

آموزش برنامه‌نویسی پایتون

برای علوم پزشکی و دارویی از صفر... به زبان خودمانی!

دکتر پدرام پریها

```
if patient.needs_python():  
    prescribe("Python")
```





دکتر پدram پربها

PythonTraining

For Medical and
Pharmaceutical Sciences

From Zero,

in Simple Language!

فارغ التحصیل دکترای عمومی داروسازی
دانشجوی دکترای تخصصی فارماسیوتیکس
محقق و مدرس برنامه نویسی و هوش مصنوعی
در علوم پزشکی و دارویی

ده سال پیش، وقتی برای اولین بار نام «پایتون» را شنیدم، هرگز تصور نمی‌کردم روزی شیفته‌اش شوم! به عنوان یک دانشجوی داروسازی که در علوم پزشکی تحصیل می‌کردم، برنامه نویسی با پایتون را دنیایی پیچیده، دور از دسترس و بدون کاربرد برای خودم می‌دانستم. اما وقتی بالاخره یادگیری آن را شروع کردم، فهمیدم که چقدر اشتباه می‌کردم و پایتون نه تنها بسیاری از کارهای خسته‌کننده‌ی روزمره و دانشگاهی‌ام را ساده‌تر کرد، بلکه به یکی از نقاط قوت من در تحصیل و کار تبدیل شد!

با گذر زمان و تجربه در برنامه نویسی و تدریس آن برای دانشجویان و اساتید علوم پزشکی، متوجه شدم که جای یک منبع آموزشی خودمانی، گام به گام و کاربردی که از صفر و با مثال‌های کاربردی علوم پزشکی این زبان را آموزش دهد، خالی است. همین ایده، پایه ی این کتاب شد: کتابی که خودم سال‌ها پیش به آن نیاز داشتم!

اگر شما نیز همیشه دوست داشتید برنامه نویسی را یاد بگیرید اما نمی‌دانستید از کجا شروع کنید و چگونه از آن در علوم پزشکی و زندگی روزمره استفاده کنید، این کتاب دقیقاً برای شماست!



نشر پید



9 786223 099687



**آموزش برنامه نویسی پایتون
برای علوم پزشکی و دارویی
از صفر... به زبان خودمانی!**

نویسنده: دکتر پدram پربها

فارغ التحصیل دکترای عمومی داروسازی

و دانشجوی دکترای تخصصی فارماسیوتیکس

محقق و مدرس برنامه نویسی و هوش مصنوعی در علوم پزشکی و دارویی

سرشناسه: پریها، پدرام، ۱۳۷۵ -
عنوان و نام پدیدآور: آموزش برنامه‌نویسی پایتون برای علوم پزشکی و دارویی: از صفر... به زبان خودمانی! / نویسنده
پدرام پریها؛ ویراستار فرزانه اعرابی.
مشخصات نشر: تهران: نشر بید، ۱۴۰۳.
مشخصات ظاهری: ۴۹۸ ص: مصور؛ ۵/۱۴ × ۵/۲۱ س.م.
شابک: ۹۷۸-۶۲۲-۳۰۹-۹۶۸-۷
وضعیت فهرست نویسی: فیپا
یادداشت: کتابنامه: ص. ۴۹۳ - ۴۹۴.
یادداشت: نمایه.
موضوع: پایتون (زبان برنامه‌نویسی کامپیوتر)
Python (Computer program language)
پایتون (زبان برنامه‌نویسی کامپیوتر) — راهنمای آموزشی
Python (Computer program language) — Study and teaching
رده بندی کنگره: Q۸۷۶/۷۳
رده بندی دیویی: ۰۰۵/۱۳۳
شماره کتابشناسی ملی: ۹۹۹۹۳۵۴
اطلاعات رکورد کتابشناسی فیپا

آموزش برنامه نویسی پایتون برای علوم پزشکی و دارویی از صفر... به زبان خودمانی!

نویسنده.....پدرام پریها
طراح جلد.....اسماعیل شاکری نژاد
صفحه آرا.....فرزانه اعرابی
ویراستار.....فرزانه اعرابی
شمارگان.....۵۰۰ نسخه
چاپ اول.....بهار ۱۴۰۴
شابک.....۹۷۸-۶۲۲-۳۰۹-۹۶۸-۷
مبلغ سرمایه گذاری.....۶۷۵ هزار تومان
کلیه حقوق برای نویسنده محفوظ می‌باشد و هر گونه تقلید و استفاده از این اثر به هر شکل، بدون اجازه کتبی ممنوع است.



انتشارات بید

تلفن: ۰۹۱۸۳۶۳۰۱۳۱ - ۴۴۲۴۲۷۵۱

مقدمه نویسنده.....	۹
--------------------	---

فصل اول: شروع دنیای زیبای برنامه‌نویسی!..... ۱۵

پایتون به زبان خودمانی چیست؟.....	۱۷
زبان‌های برنامه‌نویسی دیگری هم داریم؟.....	۱۹
پایتون را چه کسی ایجاد کرد؟.....	۲۲
زبان پایتون برای ما در علوم پزشکی و دارویی چه کاربردی دارد؟.....	۲۴
چند نکته مهم در مسیر یادگیری پایتون!.....	۳۲
اما من سیستم خوبی ندارم! آیا می‌توانم برنامه‌نویسی را شروع کنم؟.....	۳۶
نصب پایتون از صفر!.....	۳۷
باز کردن محیط پایتون و شروع برنامه‌نویسی!.....	۴۱
ظرف‌ها در برنامه‌نویسی (متغیرها).....	۴۶
کلمات و حروف در پایتون (رشته‌ها).....	۵۲
تغییر دادن یک متغیر.....	۵۶
چاپ کردن مقادیر روی صفحه.....	۵۸
تبدیل نوع متغیرها.....	۶۰
گرفتن ورودی از مخاطب.....	۶۴
نوشتن یک برنامه واقعی!.....	۶۶
نوشتن برنامه بعدی، گرفتن وزن و قد مخاطب.....	۷۵

معجزه هوش مصنوعی chatGPT در رفع ایراد کدا!..... ۷۹

فصل دوم: لذت تصمیم‌گیری با استفاده از شرطها!..... ۸۷

چطور Spyder را باز کنیم؟..... ۹۱

تغییر ظاهر اسپایدر..... ۹۲

ترکیب شرطها (مثال بررسی سن و جنسیت بیمار)..... ۱۰۸

فصل سوم: حلقه‌ها و لذت انجام کارهای تکراری و خسته‌کننده!..... ۱۲۳

حلقه‌ها به چه دردی می‌خورند؟..... ۱۲۵

شمارش با حلقه‌ها..... ۱۴۷

یادگیری مفهوم حلقه‌های تودرتو!..... ۱۵۵

استفاده از هوش مصنوعی برای رفع ایراد کدا!..... ۱۶۵

فصل چهارم: کتابخانه‌ها و متدهای رشته‌ها!..... ۱۷۵

کتابخانه چیست و در علوم پزشکی به چه دردی می‌خورد؟..... ۱۷۷

متدهای رشته‌ها..... ۱۸۶

سایر متدهای رشته‌ها..... ۱۹۶

یک مثال علوم پزشکی ترا!..... ۲۰۶

دسترسی به موارد موردنظر در رشته!..... ۲۰۸

حلقه‌ها روی رشته‌ها! یک مثال ژنتیک!..... ۲۱۰

فصل پنجم: لذت کار با داده‌های بزرگ‌تر با استفاده از لیست‌ها در پایتون!..... ۲۳۳

لیست چیست و به چه دردی می‌خورد؟..... ۲۳۵

افزایش کارایی با متدهای لیست‌ها!..... ۲۴۰

یک مثال از قبل با لیست‌ها!..... ۲۵۳

مثال بیماری‌ها!..... ۲۵۲

استخراج اطلاعات از یک متن (مثال شهر و جمعیت)..... ۲۷۲

فصل ششم: لذت دسترسی آنی به اطلاعات موردنظر با دیکشنری!..... ۲۹۱

دیکشنری چیست و به چه دردی می‌خورد؟..... ۲۹۳

۳۰۱.....متدهای دیکشنری (مثال فشار خون بیماران).....

۳۱۲.....دیکشنری‌های پیچیده‌تر (مثال فشار خون و وزن بیماران به صورت هم‌زمان).....

فصل هفتم: لذت خروج از مرزهای پایتون با خواندن فایل‌ها!..... ۳۳۵

۳۳۷.....خواندن و نوشتن فایل‌ها با پایتون برای ما چه کاربردی دارد؟.....

۳۳۹.....شروع یک مثال کاربردی! آمینواسیدها!.....

۳۵۷.....انواع فایل در کامپیوتر و نحوه باز کردن آن‌ها در پایتون.....

۳۶۱.....فهماندن آدرس فایل به پایتون!.....

۳۶۶.....خواندن فایل موردنظر در پایتون.....

۳۷۰.....استفاده از اطلاعات فایل خود در پایتون.....

۳۸۵.....نوشتن اطلاعات موردنظرمان به صورت فایل با پایتون.....

۳۹۸.....قسمت اختیاری: باز کردن فایل‌های متنی به صورت جدول در برنامه اکسل.....

فصل هشتم: تابع راهی برای کارهای تکراری و منظم شدن!..... ۴۱۷

۴۱۹.....چرا تابع؟.....

۴۲۲.....چگونه تابع بنویسیم؟.....

۴۳۰.....متغیرهای محلی و جهانی!.....

۴۳۷.....نوشتن برنامه ترجمه پروتئین.....

فصل نهم: set، حلقه while و مدیریت خطاها..... ۴۵۷

۴۵۹.....set چیستند و به چه دردی می‌خورند؟.....

۴۶۳.....متدهای set برای مثال داروها.....

۴۶۶.....حلقه while چیست؟.....

۴۷۴.....مدیریت کردن خطاها با try except.....

فصل دهم: ادامه یادگیری و منابع!..... ۴۸۷

۴۹۳.....منبع.....

۴۹۵.....واژه‌نامه.....

از همسر عزیزم بسیار تشکر می‌کنم که اگر حمایت‌های او نبود، روحیه و ارادهٔ اتمام این کتاب را نداشتم. از برادرم که توصیه‌هایش بسیار به من کمک کرد. از پدر و مادرم و خانوادهٔ همسر عزیزم که همیشه به من در این راه روحیه دادند و هیچ‌گاه مرا مجبور به انتخاب مسیر داروخانه (که کمتر موردعلاقه‌ام بود) نکردند. دوست عزیزم دکتر سید مهدی ساداتی که اگر نبود، بسیار بر من سخت می‌گذشت و از تمامی اساتید عزیزم که از آن‌ها فراوان یاد گرفته‌ام تشکر می‌کنم و از جناب محمدپیام بهرام‌پور که درست است هیچ‌گاه از نزدیک ندیدمشان ولی در مورد روش تدریس و آموزش از ایشان بسیار آموختم؛ در نهایت

این اثر را به روح پدرم تقدیم می‌کنم؛

کسی که عشق من به مطالعه و یادگیری از او نشئت گرفت...

مقدمه نویسنده

ماجرای به چند سال پیش برمی گردد؛ زمانی که همه جا و البته بیرون از رشته دانشگاهی ام، داروسازی و علوم پزشکی، نام پایتون را می شنیدم؛ اما نسبت به آن دافعه داشتم. می دیدم دوستانم با این زبان برنامه نویسی کار می کنند و یکی از اساتیدم آن را توصیه کرد؛ اما به مهارت هایی که قبلاً داشتم چسبیده بودم. هرچه به من گفته می شد که در این دوره زمانه پایتون رشد بسیار زیادی را تجربه کرده و به ناچار به آن سمت وسو خواهیم رفت، گوش نمی دادم. هوش مصنوعی؟ یادگیری ماشین؟ بیگ دیتا؟ چه کسی حوصله اش را دارد و سیستم من نمی کشد و از این جور بهانه ها می آوردم و از این حیطه ها فرار می کردم.

تا این که چند سال پیش ایمیلی تبلیغاتی از یکی از مؤسسات آموزشی معروف به دستم رسید. دوره هوش مصنوعی با پایتون؛ آن ایمیل چیزی را در ذهن ام تلنگر زد. گفتم هرچه که عقبش انداختم، الآن وقت آن است که پایتون را شروع کنم و آن دوره گران قیمت (که در آن زمان کل پس انداز من را در برمی گرفت) را ثبت نام کردم. با تأخیرات زیاد که باعث شد تنها چند نفر از ما تا جلسات انتهایی باقی بمانیم و دوره نیمه کاره رها شد؛ اما با این وجود همان دوره ناقص، کار خودش را کرد. من را علاقه مند

که هیچ، بلکه شیفته پایتون کرد! وقتی دیدم در حال فراگیری یک زبان برنامه نویسی هستم که می توانم اکثر کارهای خسته کننده خود را با آن انجام دهم شگفت زده شدم. از کارهای خسته کننده علمی گرفته تا کارهای روزمره مثل تغییر انبوه فایل هایی که در پوشه های خود داشتم تا مدیریت موسیقی هایم، دانلود موارد زیاد و استخراج اطلاعات از اینترنت با پایتون و... این موارد را که به صورت خودآموز از اینترنت و... فرا می گرفتم علاقه ام را به پایتون بسیار فراتر کرد و به حدی رسید که زندگی حرفه ای و روزمره خود را به قبل از آشنایی با پایتون و بعد از آن تقسیم می کنم.

اما نکته ای وجود داشت! اکثر دوره های آموزشی با زبانی سخت و بدون کاربرد تدریس شده بودند. باتوجه به علاقه زیاد من به تدریس و کمک به دیگران، دوره ای را برای دانشجویان تحصیلات تکمیلی دانشگاه علوم پزشکی شیراز طراحی کردم که پایتون را از صفر و به زبان خودمانی و به صورت مسئله محور آموزش دهد. مثال ها تا حد امکان در حیطه علوم پزشکی و دارویی باشد تا فراگیران بتوانند از همان ابتدا به کاربرد پایتون در حیطه خود فکر کنند و مهم تر از همه، تفکر برنامه نویسی و مهارت جستجو در این حیطه را تمرین کنند. به این شکل که موارد متعددی را خودم جلوی دوستان در گوگل جستجو می کردم که اصطلاحاً ماهیگیری را یاد بگیرند نه این که فقط از من ماهی بگیرند و تا چند سال بعد هم نتوانند خودشان را بروز کنند.

باتوجه به استقبال خوبی که از دوره شد، به من پیشنهاد شد که این دوره را مجدد برای هیئت علمی و دانشجویان تحصیلات تکمیلی برگزار کنم و از آن به بعد نیز این دوره در جاهای مختلفی برگزار شد. حاصل آن دوره، کتابی است که پیشروی شماست و امیدوارم بتواند در راستای این اهداف مفید باشد.

کتاب این گونه طراحی شده که هر هفته، یک فصل از آن به همراه تمریناتش

قابل انجام باشد. در این دوره، برنامه‌نویسی به زبان ساده و قابل فهم و از صفر به همراه مثال‌هایی در علوم پزشکی و دارویی توضیح داده شده است. در ابتدا کاربرد برنامه‌نویسی در علوم پزشکی و دارویی بیان شده و در ادامه از نصب پایتون قدم به قدم شروع کرده تا به مباحث پیشرفته‌تر برسیم. سعی شده این دوره به گونه‌ای طراحی شود تا مبنای خوبی ایجاد کنند که در ادامه دوستان بتوانند مباحث مورد علاقه خود را مانند یادگیری ماشین و هوش مصنوعی، دیتا ساینس و... را پیش ببرند. درست است که این دوره برای مخاطبین علوم پزشکی و دارویی نوشته شده است، اما سعی شده زبان تدریس و مثال‌ها به گونه‌ای باشد که حتی برای افراد خارج از این رشته‌ها نیز مفید باشد.

انشاء... اگر خوانندگان گرامی از این کتاب استقبال نموده و عمری باقی بود، جلد بعدی این کتاب با موضوع کتابخانه‌های مربوط به علوم داده مانند `numpy` و `pandas`، نحوه رسم نمودارهای زیبا با کتابخانه‌های `matplotlib` و `seaborn`، ایجاد محیط‌های مجازی و آشنایی با استخراج اطلاعات از وب با پایتون (وب اسکرپینگ) و شی گزایی نیز مورد نگارش قرار خواهد گرفت.

قطعاً این کتاب خارج از اشکال و نقص نخواهد بود و سپاسگزار خواهیم شد که مرا از پیشنهادها و نقاط بهبوددهنده که به نظر تان می‌رسد بی‌نصیب نگذارید. در این رابطه آدرس من در شبکه‌های مجازی (لینکدین، تلگرام و ایتا) و همچنین ایمیل را می‌توانید مشاهده کنید:

p.porbaha@gmail.com

@PedramPorbaha

در نهایت لازم است بگوییم که با تمامی مشغله‌های کاری و روزمره، «ریزعات» روزانه نوشتن یک کلمه بود که کمک کرد این کتاب را به پایان برسانم. بعضی مواقع پس از یک کلمه به خودم می‌آدمم و می‌دیدم که ۴۰ صفحه نوشته‌ام و بعضی مواقع از فرط خستگی آن روز، تنها همان یک کلمه را انجام داده و نرم‌افزار ورد را می‌بستم؛ به‌گونه‌ای که در حدود ۷ ماه متن اولیه این کتاب نوشته شد.

در ادامه، اگر موافق هستید؛ برویم بر قدرت یادگیری را شروع کنیم!

دکتر پدرام پربها

شهریورماه ۱۴۰۳

نکته مهم: تمامی این کتاب به صورت فیلم آموزشی به رایگان در کانال آپارات من در لیست پخش «دوره آموزشی پایتون از صفر برای علوم پزشکی» قرار دارد. کافی است که همین عنوان را در گوگل جستجو کنید تا پیدایش کنید (اگر لیست پخش مربوطه در گوگل پیدا نشد، بارکد زیر را با موبایل خود اسکن کنید یا روی لینک زیر کلیک کنید). توصیه می‌کنم در صورت نیاز حتماً در کنار کتاب، فیلم جلسه مربوطه را مشاهده کنید تا موضوع به خوبی جا بیفتد.



پایتون برای علوم پزشکی از صفر

All Images Videos Shopping News More

Tools

About 112,000 results (0.29 seconds)



آپارات

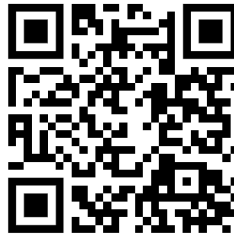
<https://www.aparat.com/playlist> · [Translate this page](#)

آموزش پایتون از صفر برای علوم پزشکی! - Playlist

آموزش پایتون از صفر برای علوم پزشکی! : 1. جلسه صفر - آموزش نصب پایتون آنکوئدا و chatGPT به راحتی آب خوردن! : 2. جلسه اول- مقدمات دوره و متغیرها- دوره پایتون از ...

شکل ۱

از طرفی همین دوره با کیفیت ویدئوی بهتر و تجارب بیشتر و همچنین با امکان ارسال تکالیف و منتورینگ شخص بنده نیز در «ایزی آکادمی» وجود داشته که با جستجو کردن «دوره پایتون برای علوم پزشکی از صفر ایزی آکادمی» در گوگل می‌توانید آن را پیدا کرده و در آن دوره نیز ثبت‌نام کنید؛ اما در عین حال دوره رایگان موجود در آپارات نیز دوره کاملی است و هیچ اجباری به ثبت‌نام در دوره همراه با هزینه نیست.



شکل ۲ https://www.aparat.com/pedram_python

راه ارتباطی من برای پاسخگویی به سوالات شما در رابطه با این کتاب، در مقدمه کتاب نوشته شده است.

ایزی آکادمی



نام آموزش مورد نظر شما...

آموزش‌های تخصصی دانشگاهی به زبان ساده و کاربردی!

ایزی آکادمی / برنامه نویسی / دوره جامع برنامه‌نویسی پایتون از صفر برای علوم پزشکی و دارویی!

رسالت مادر «ایزی آکادمی»، آموزش مباحث تخصصی و دانشگاهی به زبان خودمانی است.

فصل اول

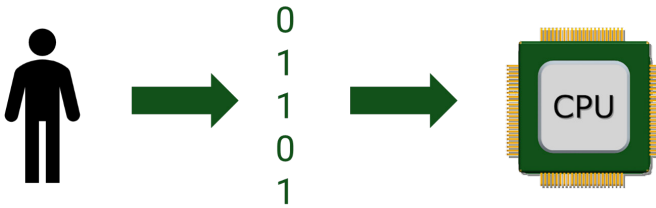
شروع دنیای زیبای برنامه نویسی!

سلام به همه شما دوستان عزیزم... اول از همه به شما تبریک می‌گویم که قصد دارید مهارتی را یاد بگیرید که بیشتر افراد در حیطهٔ ما یعنی علوم پزشکی و دارویی از آن اطلاعی ندارند، چه این که بخواهند آن را یاد بگیرند. شما در این لحظه می‌توانستید در جایی دیگر باشید؛ اما قصد دارید که این حیطه را شروع کنید و در کار و حتی مسائل روزمرهٔ زندگی خود از آن کمک بگیرید (در ادامه برایتان مثال‌های زیادی خواهم زد که پایتون چگونه در کارهایم، حتی در امور غیردانشگاهی به درد من خورده است). برنامه‌نویسی حیطه‌ای است که یادگیری‌اش به آن پیچیدگی که فکر می‌کنیم نیست؛ اما حتی اگر قصد نداشتید برنامه‌نویسی را در علوم پزشکی و روزمره ادامه دهید، به شما دید حل مسئله و دیگر مزایا را می‌دهد که همان نیز بسیار ارزشمند است.

پایتون به زبان خودمانی چیست؟

این روزها همه‌جا صحبت از پایتون است؛ اما پایتون درواقع چیست؟ به‌صورت خلاصه، پایتون یک‌زبان برنامه‌نویسی است؛ اما زبان برنامه‌نویسی چیست؟

دوستان عزیزم! فرض کنید ما می‌خواهیم با کامپیوتر ارتباط برقرار کنیم و از او بخواهیم تا کاری را برایمان انجام دهد، به همین دلیل من باید با مغز کامپیوتر (که به آن CPU گفته می‌شود) ارتباط برقرار کنم. من نمی‌توانم به آن بگویم: «ای کامپیوتر عزیز! بیا لطفاً این کار را برایم انجام بده!». تنها چیزی که مغز (همان CPU کامپیوتر) می‌فهمد، زبان صفر و یک است (شکل ۱)!



شکل ۱- CPU کامپیوتر فقط زبان صفر و یک را می‌فهمد!

دانشجو: این که خیلی سخته استاد!

استاد: دقیقاً خیلی سخت است! شاید برایتان جالب باشد که در اوایل پیدایش کامپیوترها، افراد مجبور بودند با کارت‌های سوراخ‌دار به‌مثابه صفر و یک با کامپیوترها حرف بزنند. حالا فرض کنید یک مگس یا حشره در یکی از این سوراخ‌های کارت‌ها گیر بیفتد! ممکن است خود این حشره به‌عنوان یک سوراخ در نظر گرفته شده و کل برنامه نوشته شده را به هم بریزد. اصطلاح «باگ» در برنامه‌نویسی که در لغت به معنای «حشره» هست از همین موضوع نشئت گرفته شده است!

بگذریم! دیدیم که این جور نمی‌شود و باید کاری کرد، و گرنه برای دادن یک دستور ساده به کامپیوتر پیر می‌شویم! به نظرتان چه کار می‌توان کرد؟

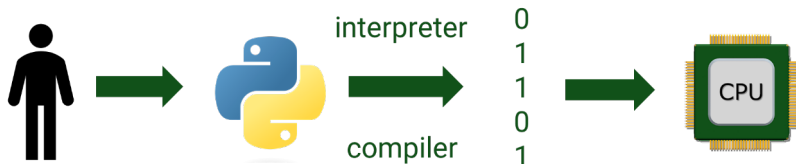
دانشجو: توی سر خودمون بزنییم و هر جور شده برنامه رو به زبان صفر و یک بنویسیم! استاد: اگر این‌طور پیش می‌رفتیم، در حال حاضر مجبور بودیم برای یک کار ساده با کامپیوتر، کلی صفر و یک تایپ می‌کردیم! بیایید این جور فکر کنیم. اگر برنامه‌نویسی چطور بود، خیلی راحت‌تر بود؟

دانشجو: اگه یه مترجم داشتیم که به زبان خودمون حرف می‌زدیم و اون می‌ومد کلام ما رو به زبان صفر و یک تبدیل می‌کرد، عالی می‌شد!

استاد: درود بر شما! رویکرد رایج کنونی برای دستور دادن به کامپیوتر، استفاده از یک «زبان برنامه‌نویسی» مثل زبان پایتون است. ما در این زبان برنامه‌نویسی به زبانی نزدیک به زبان خودمان و نه زبان صفر و یک، برنامه مدظرمان را می‌نویسیم و زبان برنامه‌نویسی برنامه نوشته‌شده ما را به زبان صفر و یک برمی‌گرداند! عالی است نه؟

دانشجو: استاد! زبان برنامه‌نویسی چطوری این کار ترجمه رو انجام می‌ده؟

استاد: در این دوره برایمان مهم نیست؛ اما اگر دوست داشتید، زبان‌های برنامه‌نویسی به‌وسیله یک ابزار خاص این کار را انجام می‌دهند. به این ابزار در زبان‌های برنامه‌نویسی، مفسر (interpreter) و در برخی زبان‌های برنامه‌نویسی کامپایلر (compiler) گفته می‌شود. شما می‌توانید هر دوی این‌ها را به یک معنا در نظر بگیرید؛ البته این دو تفاوت ریزی باهم دارند.



شکل ۲- نقش زبان برنامه‌نویس به‌عنوان یک پل!

زبان‌های برنامه‌نویسی دیگری هم داریم؟

گفتیم که زبان‌های برنامه‌نویسی آمده‌اند تا دستورات نوشته‌شده به زبان انسان را به زبان کامپیوتر تبدیل کنند. هرچه این زبان برنامه‌نویسی به زبان ما انسان‌ها نزدیک‌تر باشد (و اصطلاحاً راحت‌تر باشد) به آن زبان، سطح بالا گفته می‌شود. برای مثال زبان‌های برنامه‌نویسی پایتون و همچنین زبان برنامه‌نویسی R و زبان برنامه‌نویسی MATLAB، جزو زبان‌های سطح‌های بالا هستند و به همین دلیل یادگیری آن‌ها

(به خصوص پایتون) چالش‌های کمتری دارد.

برخی زبان‌ها سطحی متوسط دارند؛ یعنی نه به سادگی زبان‌های سطح بالا هستند و نه به پیچیدگی زبان‌های سمت پایین؛ چیزی در میان این دو. زبان بسیار محبوب ++C یکی از این زبان‌هاست که حتی پس از چند دهه هنوز به وفور استفاده می‌شود.

بد نیست یادی از یکی از زبان‌های سطح پایین کنیم. یکی از مشهورترین زبان‌های سطح پایین، زبان برنامه‌نویسی اسمبلی است که بسیار نزدیک به زبان ماشین است. این زبان بیشتر برای کسانی که علاقه‌مند به معماری کامپیوتر در سطوح بنیادی‌تر هستند کاربردی است.

اگر بخواهیم مقایسه‌ای حدودی بین این سه نوع زبان‌های برنامه‌نویسی داشته باشیم، برای مثال برای نوشتن برنامه‌ای که روی صفحه بنویسد: «من پایتون را دوست دارم»، در زبان اسمبلی نیاز به حدود ده خط برنامه‌نویسی، در زبان ++C نیاز به حدود چهار خط و در زبان پایتون تنها نیاز به یک خط کدنویسی داریم!

سطح بالا



ASSEMBLY

سطح پایین



شکل ۳- برخی زبان‌های سطح بالا و پایین

دانشجو: اگر زبان‌های برنامه‌نویسی سطح بالا این قدر کار ما رو راحت‌تر کردن، پس چرا هنوز زبان‌های سطح پایین وجود دارن؟

استاد: درست است که برنامه‌نویسی به زبان‌های سطح بالاتر بسیار راحت‌تر است، اما زبان‌های سطح پایین مزیت‌های زیادی دارند. اولاً که بسیاری از زبان‌های سطح بالا با استفاده از همین زبان‌های سطح پایین‌تر نوشته شده‌اند (مثلاً همین جناب پایتون را با استفاده از زبان ++C نوشته شده است). همچنین در رابطه با سرعت اجرای برنامه نوشته شده، زبان‌های سطح پایین سرعت بسیار بالاتری از زبان‌های سطح بالا دارند.

دانشجو: یعنی سرعت پایتون که یک زبان سطح بالاس خیلی کمه؟

استاد: نه! همان پایتون هم از نظر ما انسان‌ها خیلی سریع است؛ اما مثلاً اگر اجرای یک برنامه به زبان پایتون حدود ده هزارم ثانیه طول بکشد، اجرای معادل همان برنامه به زبان ++C حدود یک هزارم ثانیه طول می‌کشد.

دانشجو: استاد! چند تا زبان برنامه‌نویسی داریم؟

استاد: تا دلتان بخواهد زبان برنامه‌نویسی داریم! خیلی از آن‌ها دیگر منقرض شده‌اند و کسی حتی اسمشان را هم نمی‌داند! این ویدئو که الآن به شما نشان می‌دهم را خیلی دوست دارم. در این ویدئو از ۵۰ سال پیش تاکنون، تمام زبان‌های برنامه‌نویسی به همراه میزان محبوبیتشان به شکل جذاب نمایش داده شده است. اگر علاقه داشتید، می‌توانید این ویدئو را در ویدئوی جلسه اول این دوره (لینک را برایتان در صفحه اول قرار داده‌ام) ببینید. همان‌طور که در این ویدئو می‌بینید، ممکن است همان‌طور که پایتون در محبوبیت از تمامی رقبا پیشی گرفت، زبان برنامه‌نویسی دیگری نیز ده سال دیگر پایتون را کنار بزند.

دانشجو: پس هیچی! احتمالاً داریم وقت خودمون رو تلف می‌کنیم! چرا پایتون یاد بگیریم وقتی که ممکنه چند سال دیگر از رده کنار بره؟!
استاد: سؤال خیلی خوبی پرسیدی! نکته بسیار مهم اینجاست! زبان‌های برنامه‌نویسی شباهت‌های بسیار زیادی به همدیگر دارند.

بگذارید از خودم مثال بزنم! زمانی که من در دوران راهنمایی و دبیرستان به برنامه‌نویسی علاقه‌مند شدم، زبان Qbasic رایج بود! من با این زبان وارد دنیای برنامه‌نویسی شدم. سپس در دبیرستان با زبان C++ آشنا شدم و آنجا بود که دیدم چقدر این زبان‌ها باهم اشتراک دارند و اصول کلی و اساسی آن‌ها باهم مشترک است. سپس در دانشگاه در مقطع عمومی برای کار پایان‌نامه‌ام مجبور شدم با زبان MATLAB آشنا شوم. بسیار برایم جالب بود که خیلی راحت‌تر از زبان‌های قبلی توانستم با این زبان کار کنم (شاید کمتر از یکی دو هفته) و این به این دلیل بود که اکثر مفاهیم این زبان، با زبان‌های قبل یکسان بود و درنهایت هم که وارد زبان پایتون شدم، این زبان از زبان‌های قبل بسیار راحت‌تر بود، چون همان‌طور که گفتم اصول اساسی برنامه‌نویسی مثل متغیرها، شرط، حلقه و... در زبان‌های برنامه‌نویسی بسیار به هم شباهت دارند و فقط کمی در نحوه نوشتن باهم تفاوت دارند. به همین دلیل اگر با یک زبان برنامه‌نویسی آشنا شوید، آشنایی با زبان بعدی برایتان بسیار راحت‌تر خواهد بود. خب دیگر خیلی وارد این موارد شدیم! برویم ببینیم زبان برنامه‌نویسی پایتون از کجا آمده و به‌خصوص در علوم پزشکی و دارویی به چه دردی می‌خورد؟

پایتون را چه کسی ایجاد کرد؟

این زبان برنامه‌نویسی را فردی نوشت که تلفظ اسمش مقداری سخت است! «گوییدو وان روسوم» در سال ۱۹۸۹، این زبان برنامه‌نویسی را به‌عنوان یک تفریح

شخصی با استفاده از زبان ++C نوشت. هدف اصلی آن یک زبان ساده و جذاب برای همهٔ افراد بود. چه کسی فکر می‌کرد ۲۰ سال بعد، پایتون به محبوب‌ترین زبان برنامه‌نویسی جهان تبدیل شود؟



شکل ۴- گوئیديو وان روسوم، خالق زبان پایتون

همان قدر که اسم این فرد سخت است، ویژگی‌های شخصیتی او نیز عجیب است. به نظر تان چرا او اسم پایتون را بر روی این زبان برنامه‌نویسی گذاشت؟

دانشجو: حتماً به مار پایتون علاقه داشته!

استاد: نه متأسفانه!

دانشجو: شاید قصد داشته برنامه‌نویسی رو به‌مثابهٔ یه مار پایتون بزرگ تمثیل کنه. استاد: برخلاف چیزی که بسیاری از ما (و از جمله در ابتدا خودم) فکر می‌کنیم، منشأ نام پایتون هیچ ربطی به مار پایتون ندارد! بلکه گوئیديو، سازندهٔ پایتون، آن قدر

به سریال طنز «سیرک مونته پایتون» یا به نام اصلی «Monty Python's Flying Circus» علاقه داشته که این نام را روی زبان برنامه نویسی خودش گذاشته است. این روحیه طنز را در بسیاری از قسمت های پایتون می بینیم که در جاهای مختلف به آن ها اشاره خواهیم کرد.



شکل ۵- چه کسی فکر می کرد این سریال طنز، دلیل نام گذاری زبان پایتون باشد؟

زبان پایتون برای مادر علوم پزشکی و دارویی چه کاربردی دارد؟

آن قدر جامعه استفاده کنندگان از زبان پایتون زیاد شده است که عملاً نمی توان موضوعی را پیدا کرد که این زبان به دردش نخورد! از حیطه های آکادمیک گرفته تا علوم پزشکی، پردازش امواج مغز، ساخت اپلیکیشن و نرم افزار، ساخت سایت، آنالیز داده و علم داده^۱، کشیدن نمودارهای زیبا، هوش مصنوعی و یادگیری ماشین، پردازش تصویر و صوت و حتی ساخت بازی هایی متوسط را می توان با پایتون انجام داد. درست است که برای مثال، برای حیطه ساخت نرم افزار زبان هایی مثل C# یا جاوا شاید

1.DataScience

مناسب‌تر باشند، اما این که زبان برنامه‌نویسی پایتون این قدر همه‌فن حریف باشد، بسیار قابلیت جالبی است. شاید برایتان جالب باشد که قسمت‌هایی از سایت‌های گوگل، یوتیوب، نتفلیکس و بسیاری از سایت‌های محبوبی که می‌شناسید با پایتون نوشته شده است.¹

از طرفی همان‌طور که گفتیم زبان پایتون یکی از زبان‌های به‌شدت سطح بالا و نزدیک به زبان انسان است. به‌روزرسانی‌هایی که در این زبان انجام شده، روزبه‌روز کدنویسی و برنامه‌نویسی را در این زبان راحت‌تر می‌کند.

زبان‌های دیگری مانند متلب (MATLAB) یا R هم زبان‌های سطح بالایی محسوب می‌شوند؛ اما کاربرد این زبان‌ها بیشتر معطوف به کارهای علمی است. در مقایسه، پایتون زبانی است که هم در محیط آکادمیک و هم در محیط غیرآکادمیک کاربردی است؛ به همین دلیل به‌شخصه برای شروع برنامه‌نویسی، این زبان را به شما توصیه می‌کنم (پس از مدتی اگر علاقه داشتید زبان دیگری را یاد بگیرید، نگران نباشید. در صورتی که با اصول یک زبان برنامه‌نویسی آشنا شوید، آشناسدن با زبانی دیگر بسیار راحت‌تر خواهد بود).

اگر بخواهم کاربردهای یادگیری زبان برنامه‌نویسی (به‌خصوص پایتون) را برای ما به‌عنوان عضوی از جامعه علوم پزشکی و دارویی و زیستی بگویم، به نظرم می‌توان این کاربردها را در ۴ دسته کلی قرار داد:

1. امروزه پایتون یکی از زبان‌های بسیار پرکاربرد در زمینه پشته‌زمینه یا همان backend سایت‌هاست.



شکل ۶- چرا پایتون می آموزیم؟

الف) کارهای تکراری خسته کننده!

این مورد، حتی برای افرادی که نمی خواهند هیچ گونه استفاده تخصصی از زبان برنامه نویسی پایتون داشته باشند نیز بسیار کاربردی است. هر عملی که نیاز به تکرار کار خاصی با موس و کیبورد و... داشته باشد و حوصله تان نشود را می توانید به کامپیوتر بسپارید و برای این کار به یک زبان برنامه نویسی که بتواند در هر حیطه ای وارد شود (صدالبته پایتون از این دسته زبان هاست) نیاز دارید! قبلاً فکر می کردم برنامه نویسی فقط برای انجام محاسبات آکادمیک و... است؛ اما بعداً فهمیدم چقدر اشتباه فکر می کردم! اکنون هر کاری که نیاز به انجام کاری تکراری داشته باشد را با پایتون انجام می دهم! شاید نوشتن آن برنامه ساعتی از من وقت بگیرد، اما در نهایت کاری که انجامش ساعت ها (و یا حتی روزها) به طول می انجامید، در کسری از ثانیه یا نهایتاً یک شب تا صبح خود انجام شده و من از نوشیدن چای خود لذت می برم! بگذارید چند مورد را برایتان مثال بزنم:

در یکی از مقالات خود، ما مجبور بودیم که اسم تمامی چند هزار ژن‌های موجود را در بدن انسان را به همراه عبارت سرچ مربوطه در پایگاه مقالات PubMed سرچ کنیم و ژن‌هایی که بیشتر از دو مقاله برای آن‌ها موجود بود، به همراه لینک مقالات را در یک فایل اکسل بنویسیم. شاید بتوانید تصور کنید که تک‌به‌تک سرچ کردن این چند هزار ژن به‌صورت دستی، چندین هفته یا حتی ماه طول بکشد! اما من با استفاده از زبان برنامه‌نویسی پایتون یک برنامه نوشتم که این کار را خودش برای من انجام دهد و نتیجه را به‌صورت یک فایل اکسل ذخیره کند. نوشتن این برنامه یک هفته و سپس اجرای برنامه و استخراج اطلاعات مربوط به این تعداد ژن، در کل کمتر از ۲۰ ساعت وقت گرفت (من عصر برنامه را اجرا کردم و خوابیدم و ظهر فایل آماده را تحویل گرفتم!). همچنین این برنامه قابلیت این را داشت تا با کمی تغییر برای کارها و مقالات بعدی مورد استفاده قرار بگیرد. بد است؟ ☺

	A	B	C	D	E	F	G
1	idx	Symbol	Count	Article Title	Article Link	and so on...	
2	2	A2M	5	Association be	https://pubmed.ncbi.nlm.nih.gov/32792993/	FoxO1, A2M, i	https://pubmed.ncbi.nlm.nih.gov/29302075/
3	3	A2ML1	1	Borderline per	https://pubmed.ncbi.nlm.nih.gov/25612291/		
4	7	AAAS	1	Safety of elect	https://pubmed.ncbi.nlm.nih.gov/19730028/		
5	8	AACS	1	Cell surface do	https://pubmed.ncbi.nlm.nih.gov/10747186/		
6	13	AADAT	2	Variation of ge	https://pubmed.ncbi.nlm.nih.gov/29777939/	Alterations in	https://pubmed.ncbi.nlm.nih.gov/35063739/
7	16	AAMDC	2	Affective immu	https://pubmed.ncbi.nlm.nih.gov/32973758/	Acculturation	https://pubmed.ncbi.nlm.nih.gov/30456725/
8	18	AANAT	4	Resequencing	https://pubmed.ncbi.nlm.nih.gov/20459461/	Running for ti	https://pubmed.ncbi.nlm.nih.gov/24729125/
9	20	AARD	1	[Results of psy	https://pubmed.ncbi.nlm.nih.gov/3670427/		
10	26	AAS5	11	Recovery from	https://pubmed.ncbi.nlm.nih.gov/27028967/	Depression-A	https://pubmed.ncbi.nlm.nih.gov/28803091/
11	29	ABAT	7	Structural brai	https://pubmed.ncbi.nlm.nih.gov/34256069/	In pursuit of f	https://pubmed.ncbi.nlm.nih.gov/36085532/
12	30	ABCA1	5	Monocyte mit	https://pubmed.ncbi.nlm.nih.gov/34171401/	Selective activ	https://pubmed.ncbi.nlm.nih.gov/35479414/
13	31	ABCA2	2	Development	https://pubmed.ncbi.nlm.nih.gov/22860215/	Cerebral ABC	https://pubmed.ncbi.nlm.nih.gov/25446622/
14	33	ABCA4	1	Pharmacogen	https://pubmed.ncbi.nlm.nih.gov/33542445/		
15	41	ABCA13	5	Association be	https://pubmed.ncbi.nlm.nih.gov/27712136/	No evidence f	https://pubmed.ncbi.nlm.nih.gov/23250005/
16	42	ABCB1	72	Pharmacogen	https://pubmed.ncbi.nlm.nih.gov/33542445/	[Association o	https://pubmed.ncbi.nlm.nih.gov/35381649/
17	43	ABCB4	1	Case-control a	https://pubmed.ncbi.nlm.nih.gov/23273899/		
18	45	ABCB6	2	Association st	https://pubmed.ncbi.nlm.nih.gov/29929738/	ABCB6, ABCB	https://pubmed.ncbi.nlm.nih.gov/24192121/
19	49	ABCB10	1	Potential invol	https://pubmed.ncbi.nlm.nih.gov/25576703/		
20	51	ABCC1	3	Rapid evidenc	https://pubmed.ncbi.nlm.nih.gov/28456840/	Cerebral ABC	https://pubmed.ncbi.nlm.nih.gov/25446622/
21	52	ABCC2	3	Pharmacogen	https://pubmed.ncbi.nlm.nih.gov/33542445/	Transporters	https://pubmed.ncbi.nlm.nih.gov/29976647/
22	59	ABCC10	1	Pharmacogen	https://pubmed.ncbi.nlm.nih.gov/33542445/		

شکل ۷- فایل اکسل نهایی (با ۳۹۹۸ ردیف)

یا یک مثال دیگر! تعداد زیادی فایل داشتم که به این شکل نام‌گذاری شده بودند:

Text1.dot

Text2.dot

Text3.dot

و

...

متوجه شدیم که عددها اشتباه شده و باید عددها یکی کمتر باشد؛ یعنی اسم فایل‌ها تک‌به‌تک باید به نام درست تغییر می‌کرد؛ یعنی نتیجه باید می‌شد:

Text0.dot

Text1.dot

و همین‌طور تا آخر. اگر تعداد فایل‌هایی که داشتم مجموعاً سه یا چهار عدد فایل بود مشکلی نداشتم؛ اما وقتی هزاران فایل وجود دارد، همین کار به‌ظاهر ساده بسیار اعصاب‌خردکن می‌شود!

اینجاست که شما با دیگران فرق دارید! یک کد با کمتر از ۱۵ خط به زبان برنامه‌نویسی پایتون می‌نویسید و بوووم! در عرض یک ثانیه تمام فایل‌ها به اسم درست rename می‌شوند. بارها این کار را انجام داده‌ام و نتیجه بسیار لذت‌بخش بوده است.

یا یک مثال دیگر! مدتی به‌عنوان مشاور جشنواره خوارزمی در دبیرستان‌ها فعال بودم. برای این که بتوانم به دانش‌آموزانم روحیه بدهم که آن‌ها هم توانایی شرکت در این جشنواره را دارند، نیاز بود که طرح‌های برتر را به آن‌ها نشان دهم. با این کار آن‌ها متوجه می‌شوند طرح‌های برتر هم آن‌جور که فکر می‌کنند آن‌چنان طرح‌های عجیب‌وغریبی نیستند و با وجود سادگی، به یک مشکل معمول در زندگی فکر کرده و آن را حل کرده‌اند.

به همین دلیل نیاز داشتم که طرح‌های برتر جشنواره خوارزمی را دانلود کنم. این

اطلاعات در سایت جشنواره خوارزمی وجود دارد؛ اما نکته مهم این است که حدود دو هزار طرح وجود داشت. فایل‌ها نام درستی نداشتند و مجبور بودم که با دست، تک‌به‌تک، نام هر فایل را به نام طرح مربوطه تغییر دهم و از طرفی تمایل داشتم طرح‌های مربوط به هر بخش در فولدر مربوطه قرار گیرد. اگر می‌خواستم تا با دست این کار را انجام دهم، فکر می‌کنم ۶۰ ساعت باید تلاش می‌کردم تا این کار حوصله‌سر بر را انجام دهم؛ اما من رویکرد دیگری را پیش گرفتم. با پایتون برنامه‌ای حدوداً با ۲۰ خط کد نوشتم (نوشتن این برنامه یک ساعت از من وقت گرفت) و این برنامه تنها در طی نیم ساعت، تمامی این دو هزار طرح را دانلود کرده و هر کدام را در پوشه متناسب با بخش خود قرار داد و حتی رتبه فرد را در کنار طرح ذخیره کرد!

یا مثالی دیگر! برای یک مطالعه تحقیقاتی، نیاز بود حدود ۸۰۰ فایل پی‌دی‌اف که محتوی عکس بودند را به متن تبدیل کنم تا بتوانم با آن‌ها مطالعه را پیش ببرم. یک برنامه با پایتون نوشتم که این کار را برای من انجام دهد و در نهایت برنامه را اجرا کردم. در حدود ۲۰ ساعت تمام فایل‌های پی‌دی‌اف من به متن تبدیل شد و کار من را در روزهای بعد بسیار راحت‌تر کرد؛ زیرا می‌توانستم با سرچ کردن ببینم محتوای موردنظر من دقیقاً در کدام صفحه آن فایل وجود دارد و در نتیجه مستقیماً به آن صفحه مراجعه کرده و از اطلاعات آن در پروژه‌ام استفاده کنم.

ب) انجام کارهای کاربردی رشته‌مان

این مورد تقریباً شبیه مورد قبل بود، اما در رابطه با حرفه یا رشته خودمان است. تا به حال چند بار شده است که برای انجام یک کار علمی، کل اینترنت را زیرورو کرده باشید و باز هم آن برنامه‌ای که می‌خواستید را پیدا نکنید؟ در اینجا می‌توانیم خودمان آن برنامه با تعداد کمی خط کدنویسی برای خودمان بنویسیم و حتی اگر دوست

داشتیم در اینترنت بارگذاری کنیم تا بقیه هم از آن استفاده کرده، ما را دعا کنند و این عمل خیر، جزئی از اعمال ماتاخر ما شود. به قول شاعر:

دیگران کاشتند و ما خوردیم ما بکاریم و دیگران بخورند

برای مثال در حیطه داروسازی، برای محاسبه پارامترهای فارماکوکینتیکی یک دارو نیاز است که تک‌به‌تک نمودارهای خود را پرازش کرد تا این پارامترها به دست بیاید. حال اگر ۱۰۰ نمودار داشتیم چگونه؟ چقدر خوب بود که این کار را کامپیوتر خودش برایمان انجام می‌داد؛ نه؟ ما می‌توانیم با پایتون این برنامه چندخطی را خودمان بنویسیم که کار ما را انجام دهد، یا مثلاً محاسبه درصد رگ‌زایی از روی تصویر سلول‌های سرطانی. برای این کار برنامه‌های کمی وجود دارد؛ اما ما می‌توانیم خودمان با استفاده از پایتون و کتابخانه‌های آماده بسیاری که برای پایتون وجود دارد استفاده کنیم و خودمان این کار را انجام دهیم. برای هرگونه کاری (از جمله کار با تصاویر، ویدئو، صوت، اکسل و...) کتابخانه‌های مرتبط زیادی برای پایتون به‌رایگان وجود دارد که عملاً این زبان برنامه‌نویسی را به یک آچارفرانسه برای انجام بسیاری از کارها تبدیل می‌کند.

ج) داده‌کاوی و علم داده

یکی از زمینه‌هایی که می‌توان به‌خوبی از پایتون در آن بهره‌برد، داده‌کاوی و دیتاهای بزرگ است. هر چه جلوتر برویم، داده‌های موجود بیشتر و بیشتر می‌شود. داده می‌تواند هر چیزی باشد؛ اقلام نسخه‌های پزشکان، زن‌های درگیر در یک بیماری، داروهای تأییدشده FDA در ۲۰ سال اخیر، پست‌های لایک شده در یک شبکه اجتماعی، اجناس فروش رفته در یک فروشگاه اینترنتی و... همگی داده هستند. این داده‌ها درعین حال که بسیار پراکنده و به‌ظاهر بی‌معنا هستند، با تحلیل کردن آن‌ها می‌توانند به ما بینش

بسیاری بدهند و پرده از الگوهای پنهان موجود در آن داده‌ها بردارند.

در صورتی که به این حیطه علاقه داشته باشید، این مهارت جدا از این که می‌تواند یک امتیاز بسیار ویژه برای شما باشد، قادر است به‌عنوان یک شغل (تحلیل‌گر داده یا دانشمند داده) مطرح شود. کافی است که در سایت‌های کاریابی (مانند جاب‌ویژن و...) عنوان «تحلیل‌گر داده» را جستجو کنید. می‌بینید که در بین این شرکت‌ها، شرکت‌های بزرگی مانند دیجی کالا، اسنپ، ترب و... هم دیده می‌شوند و روزبه‌روز نیز بر این عنوان‌های شغلی افزوده می‌شود.

نکته جالب ماجرا این است که یکی از ابزارهای اصلی تحلیل داده، همین زبان برنامه‌نویسی پایتون است که با کتابخانه‌های قدرتمندش، امکانات بسیاری را هم در زمینه تحلیل داده‌ها و هم در زمینه بصری‌سازی داده‌ها در اختیار ما قرار می‌دهد. شاید برایتان جالب باشد که اکثر نمودارهای مقالات‌ام را با زبان پایتون رسم می‌کنم که باعث شده جلوه بسیار زیباتری داشته باشند و همچنین رسم تعداد زیادی نمودار را برای من در مقالات و ... راحت‌تر کرده است.

د) هوش مصنوعی و یادگیری ماشین

امروزه نام «هوش مصنوعی» را در هر حیطه‌ای می‌شنویم. یکی از زبان‌های برنامه‌نویسی که در این زمینه حکمرانی می‌کند، پایتون است و با کتابخانه‌های بسیار قدرتمندش در این زمینه بسیار خوب عمل می‌کند.

دانشجو: استاد هوش مصنوعی چیه؟ به درد ما که در علوم پزشکی هستیم می‌خوره؟

استاد: صد درصد! در این رابطه اگر دوست داشتید یک وینار رایگان ۳ ساعته، به نام کاربرد هوش مصنوعی در علوم پزشکی برگزار کرده‌ام و در آن به زبان ساده و خودمانی توضیح داده‌ام که هوش مصنوعی چیست و چگونه کار می‌کند. در آن وینار

مثال‌های زیادی را از کاربرد یادگیری ماشین در حیطه علوم پزشکی و دارویی ذکر کرده‌ام. کافی است در گوگل جستجو کنید: «کاربرد هوش مصنوعی در علوم پزشکی به زبان خودمانی! پدرام پربها».

چند نکته مهم در مسیر یادگیری پایتون!

الف) از سؤال کردن نترسید!

سؤالی که فکر می‌کنید بسیار سؤال سطح پایین و ساده‌ای است، در اکثر مواقع سؤال بسیاری دیگر هم بوده است؛ ثانیاً حتی اگر پاسخ به سؤال شما بخواهد وقت کلاس را بگیرد، این وظیفه استاد است که وقت را مدیریت کند و به شما بگوید که بعد از وقت کلاس به‌صورت تک‌نفره به شما پاسخ دهد و شما لازم نیست که نگران این موضوع باشید.

یادم است که در یکی از مؤسسات به‌عنوان تدریس‌یار استاد، جلسه رفع اشکال پایتون را برگزار می‌کردم. دانشجویان آن کلاس چند ماهی بود که پایتون را شروع کرده بودند. متوجه شدم یکی از دانشجویان، دو ماه است که ندانسته برنامه‌ای دیگر را به‌جای پایتون باز می‌کرده است (و با خطا روبرو می‌شده است)؛ اما چون فکر می‌کرد که سؤالش، سطح پایینی دارد این را از استاد کلاس نپرسیده بود و همین باعث شده بود که در این دو ماه نتواند تمرین کند! نکته جالب ماجرا این بود که افراد دیگری هم این مشکل را داشتند و این خطا، اشکال رایجی بود که در صورت پرسیدن از استاد، خیلی راحت قابل حل می‌بود.

به قول یکی از احادیث، دو نفر از پرسیدن سؤال سود می‌برند: هم کسی که سؤال می‌پرسد و هم کسی که از او سؤال پرسیده می‌شود. بارها شده که دانشجویانم با پرسش‌هایشان باعث شده‌اند تا نکته‌ای به ذهنم برسد و همان نکته باب آموختن نکات

بیشتر را فراهم کند. پس از سؤال کردن نترسید!

ب) هر کسی روند یادگیری خودش را دارد!

ممکن است در یک کلاس ببینید که دیگری بسیار سریع‌تر از شما پیشرفت می‌کند. این موضوع می‌تواند علت‌های بسیاری داشته باشد؛ از جمله اینکه که شاید آن فرد قبلاً زبان برنامه‌نویسی دیگری را آموخته بوده یا... خودتان را با کسی مقایسه نکنید و پیشرفت‌تان را با خودتان بسنجید. در کلاسی که من بودم، افراد از من سرعت بیشتری داشتند؛ اما چون پایتون را رها کردند و دیگر از آن در زندگی و کار خود بهره نبردند، آن را به فراموشی سپردند و منتظرند تا روزی فرارسد که وقت کنند و بخواهند پایتون را مجدداً شروع کنند. در حالی که تنها امروز دست ماست و اگر کار را به فردا موکول کنیم، شاید هیچ‌گاه دیگر نتوانیم یا نخواهیم کار را شروع کنیم. یک تمرین ناقص و اشتباه، بسیار بهتر از تمرین نکردن به‌خاطر تحویل دادن یک تمرین بی‌نقص و عالی است.

ج) از جستجو کردن نترسید!

در برنامه‌نویسی بسیار طبیعی است که چیزی را ندانید یا فراموش کرده باشید. وقتی می‌خواهید از برنامه‌نویسی در زندگی روزمره یا کاربردی استفاده کنید و قسمتی از آن را برای بار اول است که دارید انجام می‌دهید، به احتمال زیاد ندانید چگونه کار خاصی را با پایتون انجام دهید. خیلی راحت در گوگل جستجو کنید؛ اما توصیه می‌کنم به انگلیسی سرچ کنید!

دانشجو: استاد ما که زبانون خوب نیست!

استاد: همه همین را می‌گویند! همین انگلیسی دبیرستان که بلد هستید، کافی است! همان را در گوگل جستجو کنید و چون با برنامه‌نویسی آشنا هستید حتی

زبان‌تان هم بهتر خواهد شد. در این زمینه من یک ویدئوی ۱۵ دقیقه‌ای پر کرده‌ام که می‌تواند به شما کمک کند. در آن ویدئو، داستان زندگی خودم که چگونه با یک روش شیرین، از کسی که از زبان وحشت داشت، به کسی که چندین سال است اکثر مطالعاتش به زبان انگلیسی است تبدیل شدم. کافی است در گوگل جستجو کنید: «چگونه با روشی غیرمعمول ولی شیرین زبان یاد گرفتم؟ پدرام پربها»

همچنین امروزه هوش مصنوعی‌های قدرتمندی مانند چت‌جی‌پی‌تی، بینگ (که از چت‌جی‌پی‌تی نسخه ۴ استفاده می‌کند) و گوگل جمناي را داریم که می‌توانید مثل یک انسان از آن‌ها در هر رابطه‌ای سؤال بپرسید و یا حتی کدتان را کیپی کنید تا ایراد آن را به شما بگویند. البته توصیه من این است که ابتدا در خود گوگل جستجو کنید و اگر به نتیجه نرسیدید از این چت‌بات‌ها استفاده کنید.

دانشجو: چطور می‌تونیم از این چت‌بات‌های هوش مصنوعی استفاده کنیم؟ من شنیدم که از ایران نمیشه!

من نحوه استفاده از این چت‌بات‌ها در ایران را در یک ویدئو توضیح داده‌ام. کافی است جستجو کنید: «آموزش استفاده از چت‌بات‌های چت‌جی‌پی‌تی، بینگ و بارد پدرام پربها» و ویدئوی مربوطه را مشاهده کنید. برای استفاده در ایران کافی است از تحریم‌شکن استفاده کنید. من خودم از سایت «ایران ۴۰۳» استفاده می‌کنم. اگر جواب نداد سایت «شکن» یا سایر نرم‌افزارهای مربوطه نیز می‌تواند خوب باشد.

د) از خطا گرفتن نترسید!

به‌قول معروف، برنامه‌ای که بار اول بدون خطا کار کند، برنامه‌ای است که هنوز نوشته نشده! (راهنمایی: این جمله از خودم بودا). در موقع گرفتن خطا، افراد فکر می‌کنند تنها کسی هستند که در برنامه‌نویسی خود با خطاهایی روبرو می‌شوند و بقیه

نابغه برنامه‌نویسی هستند! من با برنامه‌نویسان حرفه‌ای زیادی آشنا هستم و می‌توانم به جرئت بگویم که اکثر آن‌ها در نوبت اول نوشتن یک برنامه با ارور روبرو می‌شوند؛ اما آن‌ها نمی‌ترسند؛ بلکه آن خطا را بررسی کرده و اگر متوجه نشوند، خطا را در گوگل یا هوش مصنوعی جستجو کرده و در نهایت آن خطا را رفع می‌کنند.

هرچقدر گرفتن خطا سخت‌تر باشد، لحظه حل آن خطا بسیار شیرین است. بارها شده که پروژه‌ای را شروع کرده‌ام، اما به دلیل خطایی که مواجه می‌شدم، آن را رها کردم و فکر می‌کردم تنها هستم. بعداً متوجه شدم که بقیه هم در آن پروژه به مشکل خورده و آن را رها کرده‌اند! همین موضوع «پس من هم تنها نیستم» به من نیرو داد و با جستجو کردن و آزمون و خطا و فکر کردن به مشکل، آن را برطرف کردم و اتفاقاً آن پروژه‌ها به پروژه‌های شیرین من تبدیل شده‌اند!

ه) در کامپیوتر خود منظم باشید!

همین اول کار که دارید برنامه‌نویسی را شروع می‌کنید، یک پوشه برای کدهایتان بسازید و هر جلسه که کار می‌کنیم، یک پوشه به اسم آن جلسه داشته باشید. به هیچ‌وجه برنامه خود را با نام‌های بی‌معنی مانند `testp`، `new1`، `newcode` و... ذخیره نکنید. دو هفته که گذشت و بعداً دسترسی از خاطرتان رفت، اگر نام‌های با معنایی برای فایل‌های خود گذاشته باشید، راحت‌تر کد مربوطه را پیدا کرده و از آن استفاده می‌کنید. بارها پیش می‌آید که سؤالی برایتان پیش می‌آید و می‌خواهید از کدهای قبلی‌تان چیزی را کپی کنید. در این صورت بسیار مهم است که نام‌های خوبی برای فایل‌های خود داشته باشید تا راحت‌تر بتوانید آن‌ها را پیدا کنید.

در نهایت برنامه‌نویسی را در زندگی خود به کار بگیرید!

به‌عنوان توصیه، سعی کنید هر دسترسی که در این کتاب (یا هر جای دیگر) یاد

می گیرید را در کامپیوتر خود اجرا کنید. به هیچ وجه به خودتان نگویید «خب تا آخر این مبحث را می خوانم و بعداً در اولین فرصت آن را چک می کنم!». خودتان این برنامه ها را بنویسید و با آزمون و خطا سعی کنید در آن ها مقداری تغییر ایجاد کنید.

همچنین (پس از حدود چند جلسه که با مباحث پایه آشنا شدید) اگر خواستید کاری خارج از مطالب دوره را با پایتون انجام دهید، من شما را منع نمی کنم و اتفاقاً تشویق می کنم که سرچ کنید و از آن استفاده کنید! به جرئت می توانم بگویم هر کاری که دلتان بخواهد را می توانید با پایتون انجام دهید. کار با فایل های اکسل، ورد، تصویر، ویدئو، صدا و... همه با پایتون ممکن هستند! اگر در روزمره با کاری روبرو شدید که زمان نداشتید دستی انجام دهید، سعی کنید با پایتون آن کار را انجام دهید. برای این کار سرچ کنید (آموزش های متنی و ویدئویی بسیاری پیدا خواهید کرد). حتی از چت بات های هوش مصنوعی مثل گوگل gemeni و بینگ bing و یا خود چت جی پی تی و جایگزین های آن ها نیز می توانید پرسید تا به شما انجام آن با پایتون را در حد نیاز شما یاد بدهند.

اما من سیستم خوبی ندارم! آیا می توانم برنامه نویسی را شروع کنم؟

دانشجو: استاد من خیلی به یادگیری پایتون علاقه مند شدم؛ اما من کامپیوتر خوبی ندارم! آیا می توانم با همین سیستمی که دارم، برنامه نویسی و بعداً هوش مصنوعی و یادگیری ماشین و تحلیل داده رو شروع کنم؟

استاد: در برنامه نویسی پایتون، حتی یک سیستم ضعیف یا متوسط نیز کار شما را راه می اندازد و به خوبی می توانید برنامه نویسی را یاد بگیرید. حتی اگر کسی کامپیوتر نداشته باشد هم می تواند از سرویس های آنلاین رایگان مثل Google co- lab استفاده کند (دقیقاً انگار دارید در پایتون کدنویسی می کنید، اما در مرورگر

خودتان). حتی در ادامه اگر قصد داشتید یادگیری ماشین و هوش مصنوعی را کار کنید، برای الگوریتم‌های کلاسیک یادگیری ماشین که الگوریتم‌های سبکی هستند، باز نیازی به کامپیوتر خوب ندارید. اگر خواستید یادگیری عمیق را کار کنید که نیاز به کامپیوترهای قدرتمندتر دارد، همان Google colab تا حد زیادی کار شما را راه می‌اندازد و حتی در سطوح پیچیده‌تر هم می‌توانید با صرف هزینه‌ای ماهیانه نسبتاً پایینی، از قسمت پولی Google colab یا سایر سرویس‌های آنلاین استفاده کنید که به شما قابلیت استفاده از این سیستم‌های قدرتمند را می‌دهد و در نهایت اگر بسیار نیاز به خرید سیستم داشتید، آن موقع می‌توانید به فکر خرید یک سیستم بیفتید. نگذارید این ترس‌ها شما را از وارد شدن به دنیای برنامه‌نویسی دور کند. چیزی که بیشتر اهمیت دارد، این است که تمرین کنید و سعی کنید از برنامه‌نویسی در زندگی روزمره و کاری خود استفاده کنید.

دانشجو: خیلی ممنون استاد! از نگرانی نجات پیدا کردم!

استاد: بسیار عالی! موافقید که برویم و برنامه‌نویسی را شروع کنیم؟

دانشجو: برییییییم!

نصب پایتون از صفر!

نکته: نصب پایتون بسیار ساده است اما با این وجود که سعی کرده‌ام کامل در این کتاب توضیح دهم، تمام این مراحل را به صورت فیلم نیز آموزش داده‌ام که به نظرم در کنار خواندن متن، آن ویدئو را هم مشاهده کنید تا راحت‌تر باشید. کافی است جستجو کنید: «آموزش نصب پایتون و آن‌اکنون‌دا به راحتی آب خوردن!» و هفت دقیقه اول ویدئو را تماشا کنید (به خصوص اگر ویندوزتان نسخه ۸ یا ۷ هست حتماً آن ویدئو را ببینید). البته می‌توانید در صفحات اول همین کتاب نیز لینک آن را پیدا کنید (شکل ۱).

نصب پایتون بسیار ساده است! برای این کار می‌توانید از دو راه اقدام کنید:

۱. **سرچ کردن لغت python در گوگل و دانلود پایتون از سایت رسمی:** این راه بسیار کم‌حجم است؛ اما به دلیل این که بعد از آن باید چند ابزار دیگر را خودمان نصب کنیم توصیه نمی‌شود (فقط برای مواقعی که می‌خواهیم دم‌دستی و سریع در یک کامپیوتر پایتون را نصب کنیم خوب است).

۲. **استفاده از آناکوندا:** راه راحت‌تر این است که نرم‌افزاری نصب کنید که با نصب آن تمام فایل‌های موردنیاز نصب شوند! «آناکوندا» چنین نرم‌افزاری است. یک بسته کامل که جدا از این که در آینده کار نصب کتابخانه‌ها را راحت‌تر می‌کند، بلکه خودش بسیاری از ابزارهای لازم را نیز در خود دارد. تنها مشکل آن حجم آن است که حجمی نزدیک به یک گیگابایت است (مقایسه کنید با حجم حدود ۵۰ مگابایتی خود پایتون). به نظرم ارزشش را دارد. به صورت خلاصه کافی است که در گوگل جستجو کنید Anaconda و در سایت مربوطه روی دانلود کلیک کنید. سپس در صفحه باز شده آن قدر پایین بیایید که به این قسمت برسید (شکل ۸):

Search and install thousands of packages and access community content and support



Students and Academics

Quickly get started with a no-cost, easy-to-use Python package and environment manager for educational and research use.

Download Now >



Practitioners and Hobbyists

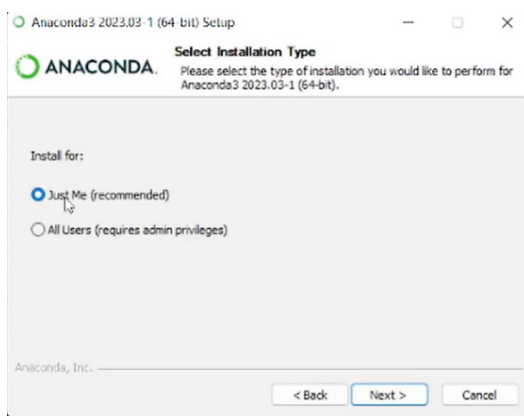
Navigate unique challenges with a robust open-source package repository and toolkit supported with technical documentation and content from our community on Anaconda.cloud.

Download Now >

شکل ۸- دانلود آناکوندا

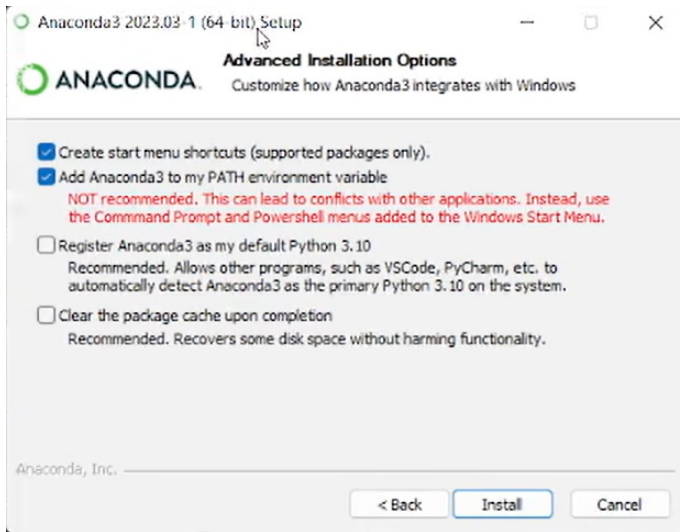
همان‌طور که می‌بینید، دو مورد نمایش داده می‌شود. من خودم هنوز بعد از این مدت

تحقیق، تفاوتی بین این دو مورد ندیدم و به‌شخصه روی همان مورد students and Academics کلیک کردم. پس از کلیک بر روی آن، از شما آدرس ایمیل می‌خواهد و درنهایت لینک دانلود به ایمیل شما ارسال می‌شود (البته می‌توانید آدرس ایمیل را وارد نکنید و روی skip کلیک کنید که در این صورت مستقیم به صفحهٔ دانلود می‌روید و کارتان راحت‌تر می‌شود). ممکن است بدون تحریم‌شکن این صفحه برایتان باز نشود؛ به همین دلیل در صورت بروز این مشکل از تحریم‌شکن یا نرم‌افزار ایران ۴۰۳ کمک بگیرید که سخت نیست و آموزش آن را در ویدئو داده‌ام (اگر ویندوزتان نسخه ۷ یا ۸ است، حواستان باشد که طبق ویدئویی که در ابتدای این مبحث توضیح دادم، ورژن مناسب ویندوزتان را باید دانلود کنید؛ اما اگر ویندوز شما نسخه ۱۰ و ۱۱ است، نیاز به اقدام خاصی نیست و راحت همین فایلی که در سایت می‌بینید را دانلود کنید 😊!) سپس فایل را دانلود و نصب کنید. در قسمت install، برای این که کارتان بعداً راحت‌تر باشد لطفاً مسیر نصب را ترجیحاً تغییر ندهید و تیک Just Me را بزنید (شکل ۹):



(شکل ۹)

توجه داشته باشید که حتماً در پنجرهٔ باز شدهٔ زیر، تیک Add Anaconda to my Path environment variable را بزنید (کلاً توصیه می‌کنم همهٔ تیک‌های موجود در این پنجره را انتخاب کنید). درست است که خود نرم‌افزار اشاره کرده است که زدن این تیک توصیه نمی‌شود، اما بعداً متوجه می‌شوید که با زدن این تیک چقدر کارتان راحت‌تر خواهد بود و نه تنها من، بلکه اکثر مدرسان پایتون، توصیه به زدن این تیک‌ها دارند (شکل ۱۰):



شکل ۱۰ - توصیه می‌کنم همهٔ تیک‌های موجود در این پنجره را علامت بزنید ☺

تمام شد. همین تیک، تمام نکتهٔ مهم نصب بود که می‌خواستیم به شما بگویم. در نهایت صبر می‌کنید تا نصب انجام شود.

دانشجو: استاد من ۳۰ دقیقه صبر کردم! چرا هنوز نصب تموم نشده؟

استاد: چون حجم فایل زیاد است، نصب آناکوندا بسته به سیستم شما ممکن است طول بکشد. مواردی داشتیم که ۴۵ دقیقه تا یک ساعت طول کشیده است. اگر دیدید که سیستم شما بسیار کند است و حتی پس از چند ساعت هم نصب تکمیل نشد، توصیه می‌کنم به جای آناکوندا، miniconda را دانلود و نصب کنید که نسخه بسیار کم‌حجم‌تری از آناکوندا است و نصب آن سریع‌تر انجام می‌شود.

دانشجو: استاد چرا من موقع نصب آناکوندا خطا می‌گیرم؟

استاد: چون سیستم شما ویندوز ۷ است و با آناکوندای جدید سازگار نیست. آناکوندا را پاک کرده و نسخه سازگار (compatible) با ویندوز خود را مطابق آموزش ویدیویی که برایتان گذاشتم، نصب کنید. به احتمال زیاد مشکلاتان رفع خواهد شد.

باز کردن محیط پایتون و شروع برنامه‌نویسی!

برویم که شروع کنیم! محیط برنامه‌نویسی پایتون به اسم IDLE معروف است.

دانشجو: IDLE مخفف چیه؟

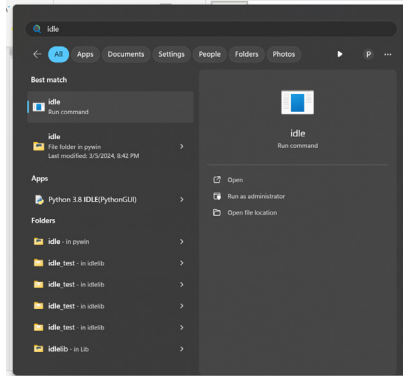
استاد: در اکثر زبان‌ها به محیط برنامه‌نویسی، IDE می‌گویند که این حروف مخفف Integrated Development Enviroment است؛ اما با توجه به این که سازنده پایتون عاشق سریال monty python بوده، اسم یکی از شخصیت‌های سریال یعنی IDLE را روی محیط برنامه‌نویسی پایتون گذاشته است!

برای باز کردن IDLE از سهرامی توان اقدام کرد:

۱. راه اول این است که در منوی استارت ویندوز، جستجو کنید idle و بعد روی آن کلیک کنید.

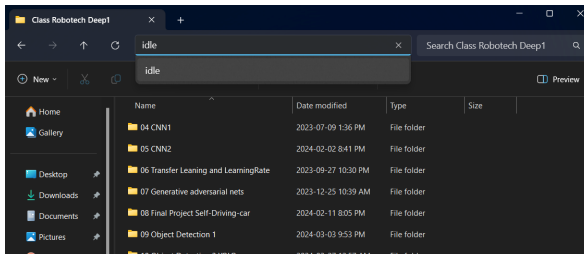
دانشجو: استاد برای من پایتون ۳.۸ باز شد. چرا قدیمیه؟

استاد: توجه کنید وقتی در استارت ویندوز، کلمه idle را جستجو کردید، دقیقاً روی idle بدون هیچ توضیح اضافای کلیک کنید، نه برای مثال روی python idle 3.8. این پایتونی قدیمی است که احتمالاً موقع نصب نرم افزارهای ویندوزی دیگر مثل SPSS روی سیستم شما نصب شده است (شکل ۱۱).



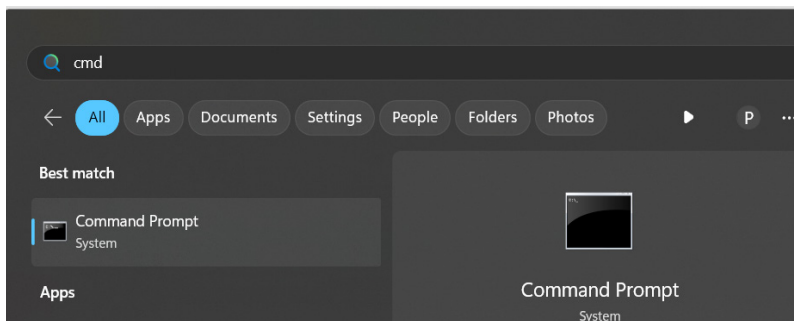
شکل ۱۱- دقت کنید دقیقاً روی idle کلیک کنید نه روی مواردی مشابه

۲. روش دوم باز کردن IDLE این است که خیلی راحت، my computer (که البته در ویندوزهای جدید نام آن به explorer تغییر کرده است) را باز کنید و در نوار آدرس بالا کلمه idle را تایپ کنید. در اکثر موارد خود من نیز از این روش استفاده می کنم.

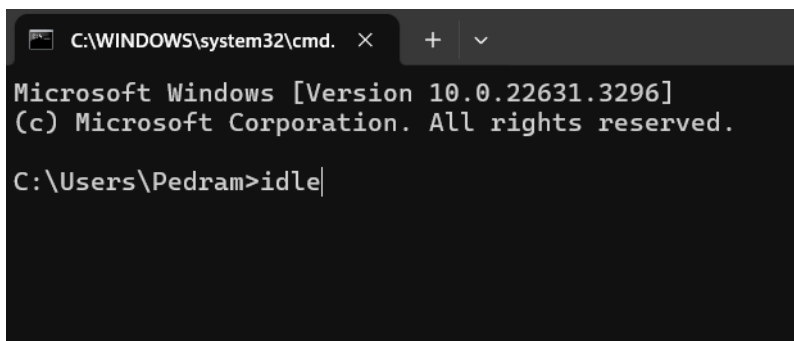


شکل ۱۲- روش دوم باز کردن IDLE

۳. در استارت جستجو کنید cmd و بعد داخل آن idle را تایپ کنید و کلید Enter را روی صفحه کلید فشار دهید (شکل ۱۴):



شکل ۱۳ - cmd



شکل ۱۴

۴. روش آخر این است که کلید `Windows+R` را روی کیبورد خود بفشارید تا پنجره Run باز شود و آنجا idle را تایپ کنید.

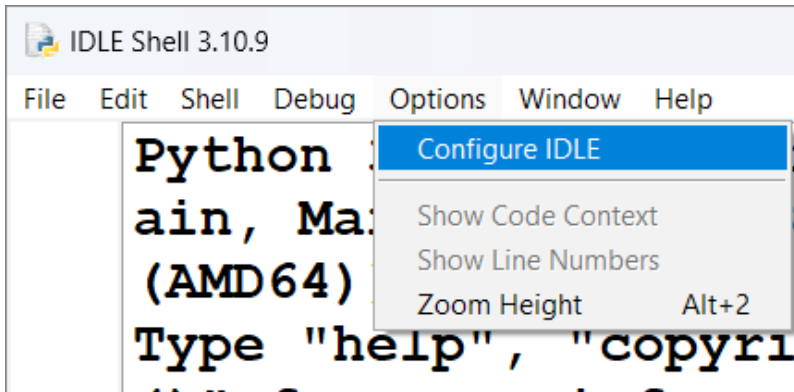
دانشجو: ویندوز، کدوم کلید کیبورد؟

استاد: همان کلیدی که روی آن عکس ویندوز کشیده شده. آن را در پایین سمت

چپ صفحه کلید پیدا خواهید کرد.

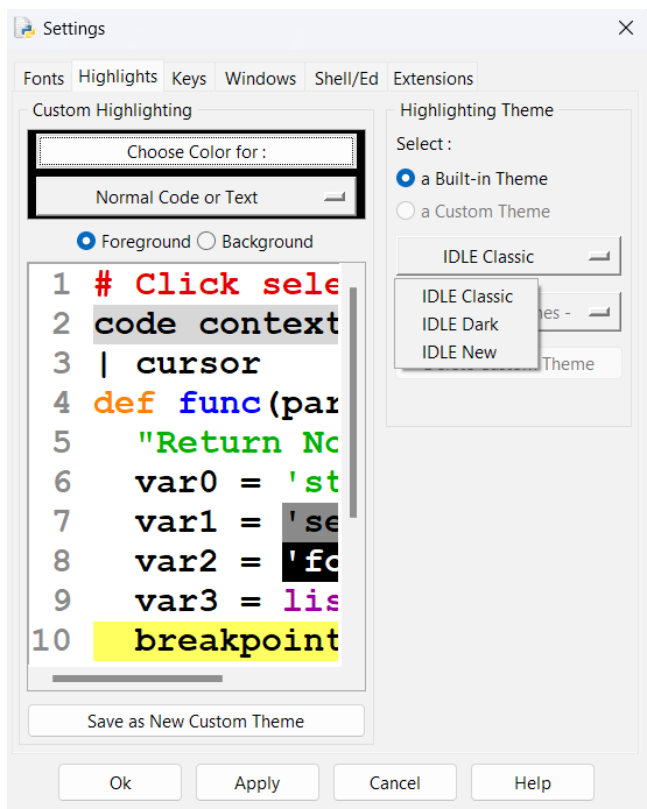
خب idle را باز کنید تا بتوانیم شروع کنیم!

قبل از شروع با کار در هر برنامه‌ای، بهتر است که ظاهر آن را متناسب با ترجیحمان تغییر دهیم تا چشم‌هایمان کمتر اذیت شود. برای تغییر ظاهر idle روی منوی option و سپس گزینه configure idle کلیک می‌کنیم (شکل ۱۵).



شکل ۱۵

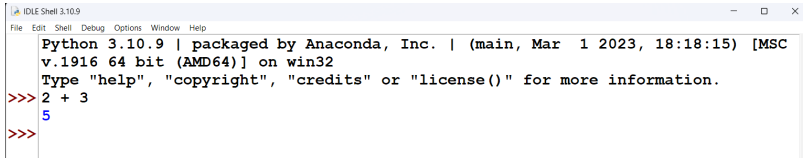
در پنجره باز شده می‌توانیم سایز نوشته (فونت) را تغییر دهیم. پیشنهاد می‌کنم فونت را بزرگ‌تر کنید (برای مثال من عدد ۱۶ یا ۱۸ را توصیه می‌کنم). اگر چشمان شما با محیط تاریک راحت‌تر است (مانند من)، روی برگه Highlights کلیک کرده و در قسمت سمت راست، روی گزینه IDLE classic کلیک کرده و IDLE Dark را انتخاب کنید (شکل ۱۶):



شکل ۱۶- تنظیم تم تاریک یا روشن

بسیار عالی! همه‌چیز آماده است. روی OK کلیک کنید تا بتوانیم برنامه‌نویسی را شروع کنیم! وقتی idle را باز کردیم، به پنجره‌ای که می‌بینید، شِل (shell) می‌گویند. مادر اینجاست می‌توانیم خط به خط دستورات را به کامپیوتر بدهیم تا آن را اجرا کند. برای مثال می‌توانید بنویسید 2+2 و سپس کلید Enter را روی کیبورد بزنید. می‌بینید که نتیجه اجرای این دستور یعنی ۴ بر روی صفحه نشان داده می‌شود و

کامپیوتر مجدد منتظر دستور بعدی ما می ماند. همین طور اگر تایپ کنید 1-5 نیز حاصل یعنی ۴ نمایش داده می شود (شکل ۱۷).



```
Python 3.10.9 | packaged by Anaconda, Inc. | (main, Mar 1 2023, 18:18:15) [MSC
v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> 2 + 3
5
>>>
```

شکل ۱۷

ظرف هادر برنامه نویسی (متغیرها):

مفهوم بسیار مهمی در برنامه نویسی داریم به نام ظرف! نام دیگر آن، متغیر (variable) است. ما می توانیم در پایتون هر ظرفی که دلمان خواست ایجاد کنیم، مثلاً اگر تایپ کنیم:

```
a = 2
```

و کلید Enter را روی کیبورد فشار دهیم، یک ظرف (یا همان متغیر) در پایتون به نام a ایجاد می شود که محتوای آن ۲ است. پس اگر آن را فراخوانی کنیم؛ یعنی به تنهایی a را تایپ کنیم و کلید Enter را فشار دهیم، روی صفحه، نمایش داده می شود:

```
2
```

«نکته بسیار مهم: اگر قصد دارید برنامه نویسی را یاد بگیرید، حتماً همراه با هر دستور، خودتان نیز آن دستور را در کامپیوتر خودتان اجرا کرده و نتیجه را ببینید. همین مورد باعث می شود که موضوع برایتان جا بیفتد. به هیچ وجه این کار به بعد از خواندن کامل فصل موکول نکنید!»

می توانیم نام های بهتری را برای متغیرهایمان انتخاب کنیم. برای مثال متغیر age برای سن:

age = 35

یا مثلاً

age2 = 35

که یک متغیر به اسم age2 تعریف می‌کند؛ اما یک نکته؛ آیا می‌توانیم اسم متغیر را 2age بگذاریم؟ مثلاً بنویسیم؟

2age = 35

دانشجو: من امتحان کردم؛ خطا داد!

استاد: دقیقاً! نام متغیر نمی‌تواند با عدد شروع شود! و همچنین فرض کنید می‌خواهیم یک متغیر تعریف کنیم به اسم

my age=23

این چطور؟

دانشجو: این هم خطا داد!

استاد: پس می‌فهمیم نمی‌توانیم بین نام متغیر فاصله یا غیره بگذاریم. پس چاره چیست؟ شاید من تمایل داشته باشم همچنین متغیری داشته باشم؟ یک روش کلک‌زدن به کامپیوتر این است که نام متغیر را برای مثال my-age بگذاریم؛ یعنی به‌جای فاصله از زیر خط (آندرلاین) استفاده کنیم. این روش هم خوانایی بالایی دارد و هم با خطا روبرو نمی‌شود.

دانشجو: میشه بنویسیم:

myAge

استاد: این روش هم خوب است؛ اما تحقیقات نشان داده که استفاده کردن از علامت آندرلاین خوانایی بیشتری دارد. از طرفی طبق استاندارد PEP8 پایتون که

دستورالعمل‌هایی برای زیبایی و خوانایی و استاندارد نویسی کدهای پایتون ارائه کرده است، بهتر است نام متغیرها و توابع با آندرلاین باشد نه به این روش که ذکر کردی.

حال نکته جالب اینجاست که باوجوداینکه چندین متغیر تعریف کردیم، تمام این متغیرها در حافظه باقی‌مانده و قابل دسترسی هستند؛ یعنی اگر بنویسیم `a`، مقدار ۲ به ما نمایش داده می‌شود. راه دیگر دسترسی به مقدار یک متغیر، استفاده از دستور `print` است؛ به این شکل که در داخل پرانتز، متغیر مدنظرمان را بنویسیم:

```
print(age)
```

یادتان باشد که در پایتون بعد از دستورات، حتماً از علامت پرانتز استفاده می‌کنیم. دانشجو: وقتی با نوشتن اسم متغیر به‌تنهایی، مثلاً نوشتن `age`، مقدار آن نمایش داده میشه، چه نیازی که کار خودمان رو سخت کنیم و از دستور `print(age)` استفاده کنیم؟

استاد: سؤال خوبی است! در اینجا (یعنی محیط `shell`) هیچ فرقی ندارد که از کدام دستور استفاده کنیم؛ اما بعداً که می‌خواهیم برنامه‌های چندخطی بنویسیم، برای نمایش مقدار متغیر حتماً نیاز به دستور `print` داریم.

راستی حواستان باشد که پایتون به بزرگی و کوچکی حروف حساس است (اصطلاحاً یک زبان `case sensitive` است) و برای همین اگر تعریف کنید:

```
Age = 13
```

یک متغیر جدید به نام `Age` داریم که مقدارش ۱۳ است و متغیر قبلی `age` نیز کماکان مقدار خود یعنی ۲۷ را دارد.

حال به نظرتان اگر بنویسیم:

```
age+2
```

یا بنویسیم

```
print(age+2)
```

چه چیزی به عنوان خروجی دریافت می‌کنیم؟

دانشجو: جالبه که هنوز مقدار `age` که برابر با ۲۷ تعریف کردیم، در حافظه کامپیوتر مونده و برای من نوشت ۲۹.

استاد: دقیقاً! این معجزه متغیرهاست! ما می‌توانیم هر چقدر متغیر دلمان خواست، تعریف کنیم و بعداً از آن‌ها استفاده کنیم و تا زمانی که برنامه بسته نشده و باز است، این‌ها در حافظه موقت کامپیوتر (یا اصطلاحاً RAM کامپیوتر) باقی می‌مانند.

یا یک مثال دیگر! اگر بنویسیم:

```
age-2
```

چطور؟

دانشجو: برای من نوشت ۲۵ که حاصل ۲۷ منهای ۲ هست.

استاد: راستی تا یادم نرفته است، حواستان باشد که نام متغیر خود را، از نام‌هایی که در پایتون مربوط به دستور خاصی هستند، انتخاب نکنید! مثلاً `print` خودش یک دستور در پایتون است (از تغییر رنگ آن می‌توانید این موضوع را متوجه شوید). برخی موارد دیده‌ام دوستان ندانسته نوشته‌اند:

```
print = 3
```

این مورد حتی اگر همان لحظه منجر به خطا نشود، بعداً در طول کار خطاهای جدی می‌دهد و شما مجبور می‌شوید پایتون را بسته و مجدداً باز کنید تا مشکل رفع شود!

حالا یک سؤال! یک متغیر به این اسم تعریف کنید:

`number = 3`

امتحان کنید و ببینید دستور:

`number*5`

چه کار می کند؟

دانشجو: حاصل شد ۱۵. انگار ستاره همون علامت ضربه!

استاد: دقیقاً! و / هم تقسیم است مثلاً:

`number/3`

۱ می شود. حال به نظرتان حاصل

`number**3`

یا همان دو ستاره چیست؟

دانشجو: لابد با این کار پایتون، ستاره بارون میشه! من زدم، 27 شد. انگار همون توان در ریاضیه.

استاد: از شوخی بی مزه شما سپاسگزاریم! ☺ درعین حال دقیقاً همان طور که گفتی `**` علامت توان در پایتون است. در بقیه زبان های برنامه نویسی، برای توان از علامت [^] استفاده می شود.

سؤال دیگر

`number%3`

نتیجه اش چه می شود؟

دانشجو: شده؛ یعنی چی متوجه نمی‌شم؟ مگه % همون درصد نبود؟

استاد: در پایتون عملگر %، باقیمانده تقسیم را حساب می‌کند. در اینجا چون 3 بر 3 بخش پذیر است و به عبارتی باقیمانده تقسیم برابر با صفر است، به ما عدد 0 نمایش داده شد و سؤال آخر! حاصل این دستور:

7//3

چه می‌شود و فرقی با تقسیم عادی یعنی 7/3 چیست؟

دانشجو: حاصلش شد 2. انگار // تقسیم صحیحه ولی 7/3 شد 2.33333333 که انگار اومده تقسیم رو با اعشار انجام داده.

استاد: دقیقاً همین طور است و جالب تر این که ما می‌توانیم مقدار به دست آمده را در یک متغیر دیگر بریزیم. مثلاً

`result = age + 2`

به این معناست که ابتدا age را به علاوه دو کن و سپس حاصل را در متغیر جدیدی به اسم result بریز. حال اگر تایپ کنیم result چه چیزی نوشته می‌شود؟

دانشجو: نوشته شد 29؛ که همان 27 به علاوه دو هست.

نکته جالب در پایتون این است که با نماد علمی هم سازگار است. برای مثال اگر یک متغیر به اسم cell-count تعریف کنیم (سعی کنیم از این به بعد دید کاربردی داشته باشیم. مثلاً فرض کنید برنامه‌ای نوشته‌ایم و تعداد سلول‌های موجود در محیط کشت را در متغیری با این اسم قرار داده‌ایم):

`cell_count = 3e6`

پایتون متوجه می‌شود که منظور ما همان 3×10^6 است؛ یعنی به عبارتی 3e6 معادل همان

$$3*10**6$$

در نظر گرفته می شود.

دانشجو: در عبارت $3*10**6$ پایتون اشتباهاً نمیدان ۳ را ابتدا ضربدر ۱۰ کنه و حاصل یعنی عدد ۳۰ رو به توان ۶ برسونه؟

استاد: در زبان های برنامه نویسی، توان بر ضرب اولویت دارد. اگر می خواستیم که ابتدا $3*10$ شده و حاصل به توان ۶ برسد باید از پرانتز استفاده می کردیم یعنی $6** (3*10)$. به صورت خلاصه یادتان باشد که در زبان های برنامه نویسی اولویت ابتدا با پرانتز، سپس با توان، سپس ضرب و تقسیم و سپس جمع و تفریق است. مثلاً اگر بنویسیم:

$$3/1+2$$

پایتون ابتدا ۳ را بر ۱ تقسیم کرده و حاصل یعنی ۳ را به دو اضافه می کند که حاصل ۵ می شود. در حالی که اگر بنویسیم:

$$3/(1+2)$$

کامپیوتر ابتدا داخل پرانتز را حساب می کند یعنی ۱ به علاوه ۲ که می شود ۳ و سپس ۳ تقسیم بر ۳ می شود؛ یعنی ۱.

کلمات و حروف در پایتون (رشته ها)

دانشجو: در متغیرها فقط می تونیم، عدد ذخیره کنیم؟ اگر خواستیم یک کلمه یا جمله رو ذخیره کنیم چیکار کنیم؟

استاد: دقیقاً به همان نکته ای اشاره کردی که می خواستم بگویم. فرض کنید بخواهیم یک متغیر به اسم name تعریف کنیم و مقدار آن را برابر با اسم یک شخص مثلاً Pedram Porbaha قرار دهیم. اگر بنویسیم:

```
name = Pedram Porbaha
```

چه مشکلی پیش می‌آید؟

دانشجو: من امتحان کردم نوشت که Pedram تعریف نشده!

استاد: همین‌طور است! پایتون به Pedram به‌مثابه یک متغیر نگاه کرده نه کلمه!
برای این‌که با خطا روبرو نشویم، باید متن خود را داخل " " یا ' ' بگذاریم، یعنی:

```
name = 'Pedram Pobaha'
```

یا

```
name = "Pedram Porbaha"
```

در پایتون هر دو مورد قبول بوده و تفاوت ندارد. به علامت "، double quotation و به علامت '، single quotation می‌گویند. یا حتی مثلاً یک جمله کامل را بنویسیم:

```
text = 'I love python programming in medical sciences'
```

وقتی متغیری دارای رشته‌ای از حروف و... است (اصطلاحاً حاوی متن است) به آن str می‌گویند (مخفف string به معنی رشته). در پایتون دستوری وجود دارد که می‌توان با آن نوع متغیر را چک کرد. نام این دستور، type است. برای مثال اگر بخواهیم ببینیم نوع متغیر name چیست کافی است بنویسیم:

```
type(name)
```

و داخل پرانتز اسم متغیر مدنظر خود را بنویسیم. اگر این دستور را اجرا کنیم نمایش داده می‌شود:

```
<class 'str'>
```

که نشان می‌دهد متغیر name از نوع str یا همان رشته است. حال بیاییم دستور type را برای متغیر age که قبلاً تعریف کردیم و مقدارش ۲۷ بود صدا بزنیم:

`type(age)`

نمایش داده شد: `int` که مخفف `integer`، یعنی اعداد صحیح است. حال اگر یک متغیر به اسم `blood-sugar = 130.5` تعریف کنیم و دستور `type` را اجرا کنیم چه نمایش داده می شود؟

`type(blood_suger)`

دانشجو: نشون داد `float`! `float` دیگه چیه؟

استاد: نظر خودت چیست؟ `۱۳۰.۵` چه فرقی با `۲۷` دارد؟

دانشجو: به نظر میاد چون اعشار داشته، نوع متغیر شده `float`.

استاد: دقیقاً! در پایتون جالب اینجاست که تعیین نوع متغیرها به صورت خودکار صورت می گیرد؛ در حالی که در بعضی زبان ها مثل `C++` باید خودمان از ابتدا نوع متغیر را مشخص کنیم. حال این دستور را اجرا کنید:

```
blood_suger = 130
```

و الآن `type` این متغیر را چک کنید.

دانشجو: جالبه الآن `int` شد؛ چون حاوی عدد اعشاری نیست. استاد راستی اگه بخوایم به متن چندخطی رو در یک متغیر بنویسیم، باید چیکار کنیم؟ من وقتی یک متن چندخطی می نویسم، با خطا روبرو میشم!

استاد: در گوگل سرچ کن و ببین چطور در پایتون می شود یک متن چندخطی را وارد کرد.

دانشجو: الآن با گرامر داغون خودم، در گوگل به زبان انگلیسی سرچ کردم:

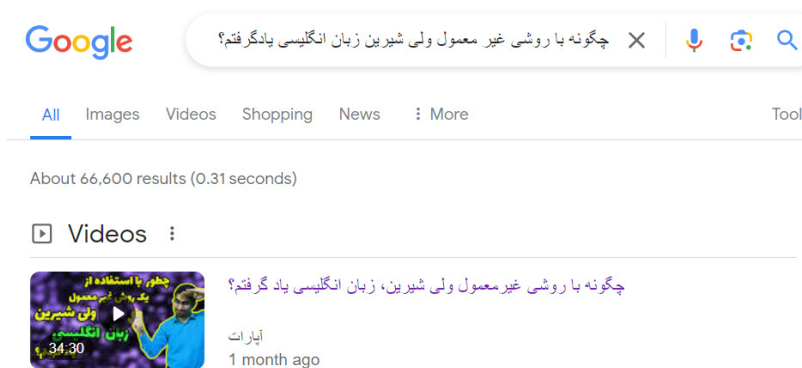
how to have a variable with multiple lines in python

نوشته انگار باید از سه تا علامت “” اول و آخر متنمون استفاده کنیم. یک مثال هم زده! مثلاً می تونیم یک متغیر به اسم `txt` تعریف کنیم:

```
txt = ''' I love Python programming
Also I love Machine learning in medical sciences
I am in term 3 of my school. I was born in 1996.
I want to achieve different skill in my field '''
```

و دیگه خطا نداد. باورم نمیشه تونستم خودم یه مطلب جدید در پایتون رو با انگلیسی دست‌وپاشکسته خودم یاد بگیرم!

استاد: درود بر تو! هیچ‌وقت یادمان نرود که هر چقدر هم که برنامه‌نویسی بلد باشیم، باز نیاز به جستجو داریم. چه بهتر به انگلیسی سرچ کنیم و همان‌طور که دیدی اکثر سایت‌ها مثال هم دارند و همین موضوع باعث می‌شود زبان انگلیسی تو هم روز به روز قوی‌تر شود! من خودم از زبان وحشت داشتم؛ اما همین‌طور با همین روش حدود ۴ سال است که حتی اکثر کتاب‌ها را نیز به زبان انگلیسی می‌خوانم. اگر تمایل داشتی، تجربه‌ام در این زمینه را در یک ویدئوی سی‌دقیقه‌ای ضبط کرده‌ام. کافی است در گوگل جستجو کنی «چطور با روشی غیرمعمول، ولی شیرین، زبان انگلیسی یاد گرفتم» تا پیدایش کنی (شکل ۱۸).



شکل ۱۸ - چگونه با روشی غیر معمول ولی شیرین، زبان انگلیسی یاد گرفتم؟

تغییر دادن یک متغیر

فرض کنید یک متغیر به اسم age تعریف کردیم:

age = 35

می‌خواهیم دو واحد به age اضافه کنیم و به همین دلیل می‌نویسیم:

age + 2

حال سؤال من از شما این است که مقدار age الآن چقدر است؟

دانشجو: من الآن نوشتم age و دیدم که مقدار age همون ۳۵ هست و تغییری نکرده!

استاد: دقیقاً! چون ما فقط به پایتون گفتیم که مقدار ۲ به‌علاوه age چقدر می‌شود و به ما حاصل را نشان داد؛ ولی به پایتون نگفتیم که این مقدار را ذخیره کند. حال فرض کنید که ما به این مقدار جدید نیاز داریم و قصد داریم که این مقدار را ذخیره کنیم. چه کنیم؟

دانشجو: می‌تونیم یه متغیر جدید مثلاً به اسم age2 بسازیم و این مقدار رو داخلش بریزیم؛ یعنی:

age2 = age + 2

استاد: درست است! این روش کار می‌کند و الآن مقدار age2 همان ۳۷ است. منتها اگر خواستیم این کار را ده بار تکرار کنیم، باید ده متغیر جدید تعریف کنیم؟ متغیر age3، age2 و...؟ آیا راهی نیست که همان مقدار متغیر age تغییر کند؟ به عبارتی حاصل را در همان خود age ذخیره کنیم؟ مثلاً بنویسیم:

age = age + 2

به عبارتی ابتدا پایتون مقدار age+2 را حساب می‌کند که می‌شود ۳۷ و سپس

حاصل را در متغیر age ذخیره می‌کند. حال چک کنید ببینید مقدار age چقدر است؟

دانشجو: چه جالب! مقدار age، ۳۷ شد و تغییر کرد؛ اما این کار حالا به چه دردی می‌خوره؟ اگر ما می‌ومدیم و یه متغیر جدید تعریف می‌کردیم، چه فرقی داشت؟

استاد: در مواقعی لازم داریم که هر بار، یک واحد به متغیر count اضافه کنیم و شاید هزار بار این کار را با کامپیوتر بخواهیم انجام دهیم. اگر این روش که به شما گفتم را بلد نباشیم، عملاً نیاز به هزار متغیر جدید داریم! بعدها در عمل، کاربرد این روش را خواهیم دید.

حال برای این که مطمئن شوم این موضوع برای شما جاافتاده است، فرض کنید یک متغیر به اسم total و برابر با مقدار ۵ داریم.

`total = 5`

اگر بخواهیم مقدار total ضربدر ۱۰ شده و مقدار total تغییر کند، باید چه کار کنیم؟

دانشجو: همون طور که دیدیم اگه بنویسیم `total*10` (یادمونه که ضرب در کامپیوتر با علامت ستاره نشون داده می‌شد)، کامپیوتر حاصل رو حساب می‌کنه ولی جایی ذخیره نمی‌کنه. ما تمایل داریم که حاصل در همون متغیر total ذخیره بشه و به عبارتی مقدار متغیر total تغییر کنه؛ پس احتمالاً باید بنویسیم:

`total = total * 10`

الآن که در خود پایتون هم امتحان کردم و مقدار total را با نوشتن نام این متغیر چک کردم:

`total`

دیدم مقدار total تغییر کرده و همون ۵۰ شد که می خواستیم.

استاد: درود بر تو؛ دقیقاً! اگر من هم بخواهم اضافه تر توضیح بدهم، ابتدا پایتون سمت راست عبارت را حساب می کند؛ یعنی مقدار total (که ۵ بود) را در ۱۰ ضرب می کند و سپس حاصل را total ذخیره می کند.

این موضوع در ابتدا ممکن است برای بسیاری از افراد عجیب باشد؛ اما پس از مدتی به آن عادت خواهد کرد. به عبارتی علامت = را در برنامه نویسی به صورت «برابر است» نخوانید؛ بلکه آن را معادل «قرار بده» بخوانید. برای مثال عبارت

```
total = total + 2
```

را به این شکل بخوانید: مقدار متغیر total را قرار بده به مقدار total به علاوه دو!

چاپ کردن مقادیر روی صفحه

فرض کنید می خواهیم متغیری به نام name داشته باشیم و اسم خودمان را در آن ذخیره کنیم؛ مثلاً:

```
name = 'Pedram'
```

(حواستان هست که Pedram را در بین دو علامت ' گذاشتم). به نظرتان دستور زیر چه می کند؟

```
print('My name is', name)
```

دانشجو: من امتحان کردم، نوشت:

```
>>> My name is Pedram
```

استاد: دقیقاً. به عبارتی شما در دستور پرینت، هر تعداد چیزی که نیت داشته باشید را می توانید، چاپ کنید. یا مثلاً دستور زیر:

```
print('Hello', name, '.How are you?')
```

که چاپ می‌کند:

```
>>> Hello Pedram.How are you?
```

دانشجو: چه جالب! ولی استاد من همین دستور رو نوشتم و خطا گرفتم. چرا؟

استاد: دستوری که نوشتی چه بود؟

دانشجو: این بود:

```
print(Hello, name, 'How are you?')
```

استاد: حواست باشد که Hello یک متن یا اصطلاحاً یک رشته است؛ پس باید حتماً بین دو علامت " قرار بگیرد. در غیر این صورت پایتون فکر می‌کند که Hello یک متغیر است در حالی که ما اصلاً همچنین متغیری را تعریف نکرده‌ایم! به خاطر همین با خطای زیر روبرو می‌شویم:

```
NameError: name 'Hello' is not defined
```

که دقیقاً به این معناست که متغیری به اسم Hello تعریف نشده است.

حال فرض کنید این متغیرها را تعریف کرده‌ایم:

```
age = 27
name = 'Pedram Porbaha'
field = 'Pharmacy'
```

به نظر تان دستور زیر چه می‌کند؟

```
print('Your name is', name, '.You are', age, 'years old', '.You study', field)
```

دانشجو: قاعدتاً باید به جای name و age و field مقدار اون‌ها رو قرار بده؛ یعنی در

نهایت خواهیم داشت:

```
>>> Your name is Pedram.You are 27 years old.You study
```

Pharmacy

تبدیل نوع متغیرها

فرض کنید متغیری به نام `age` داریم و به این صورت آن را ایجاد کرده‌ایم:

```
age = '35'
```

اگر یادتان باشد، با دستور `type` می‌توانستیم نوع متغیر را ببینیم. به نظرتان اگر نوع این متغیر را با دستور زیر مشاهده کنیم، چه چیزی می‌بینیم؟

```
type(age)
```

دانشجو: تا جایی که یادمه گفتید اعداد صحیح، یعنی اونایی که اعشار ندارن، به صورت `int` هستن.

استاد: بله، اما اینجا یک تفاوت وجود دارد. اگر به این شکل می‌نوشتیم، `age` به صورت `int` بود:

```
age = 35
```

اما ما نوشته‌ایم:

```
age = '35'
```

دانشجو: آهان الان به علامت ^{۶۰} دقت کردم؛ یعنی الان این متغیر از نوع رشته‌اس؟

استاد: درسته! و می‌بینید که خروجی دستور

```
type(age)
```

هم به شکل `str` (که همان مخفف `string` یا رشته است) نمایش داده شد؛ اما نکته این است که فرض کنید بخواهیم به `age`، دو واحد اضافه کنیم یعنی:

```
age + 2
```

دانشجو: چرا خطا داد! نوشت:

`TypeError: can only concatenate str (not "int") to str`

استاد: دقیقاً! اینجا است که صدای پایتون در می‌آید و می‌گوید که من نمی‌توانم یک عدد را به یک متن اضافه کنم! به عبارت دیگر «منی‌توانم قیমে‌ها را روی ماست‌ها بریزم!».

دانشجو: خب تکلیف چیه؟ من می‌خوام واقعاً `age` رو به شکل عدد در نظر بگیره نه به شکل یه متن!

استاد: اینجا است که باید به طریقی، نوع متغیر را به `int` تبدیل کنیم! این کار بسیار ساده است. اگر بنویسیم:

```
int(age)
```

خروجی این دستور چیست؟

دانشجو: چه جالب؛ شد 35 بدون هیچ‌گونه علامت 'اضافی'.

استاد: حال مقدار `age` را چک کن. آیا مقدار `age` هم تغییر کرده است؟

دانشجو:

```
age
```

اه... نه نشون داد '35'! یعنی انگار هنوز رشته‌اس.

استاد: دقیقاً. چون ما حاصل را در متغیر `age` ذخیره نکردیم. چطور می‌توانیم این کار را انجام بدهیم؟

دانشجو: فکر کنم همون کاری که یاد گرفتیم و من فکر می‌کردم به درد نمی‌خوره!

یعنی بنویسیم:

```
age = int(age)
```

تا پایتون ابتدا سمت راست عبارت رو حساب کنه (که میشه 35) و بعد حاصل رو در متغیر age ذخیره کنه.

استاد: درود بر تو! حالا چک کن ببین مقدار age چه شده؟

دانشجو: جالبه الآن 35 شد، بدون هیچ گونه علامت 'اضافی'.

استاد: و دستور زیر چه خروجی ای داد؟

```
type(age)
```

دانشجو: نوشت `<class 'int'>` انگار درست شد و الآن نوع متغیر از جنس `int` هست! به عبارتی الآن اگه بنویسم:

```
age + 2
```

دیگه با خطا روبرو نمی شم؟

استاد: به نظرم اگر خودت امتحان کنی، سریع تر به جواب سؤال می رسی!

دانشجو: درسته... 😊 چه جالب! الآن هیچ خطایی نگرفتم و حاصل به درستی شد 37.

استاد: درود بر تو! حالا برای این که متوجه شوم مطلب را درست آموخته اید یک سؤال می پرسم! فرض کنید یک متغیر به نام `number` داریم:

```
number = '100'
```

و گیج بازی درآورده ایم و ندانسته آن را داخل علامت " قرار داده ایم! همین باعث شده است که نتوانیم عملیات جمع و تفریق را روی آن انجام دهیم. چه کنیم؟

دانشجو: طبق چیزی که یاد گرفتیم، الآن نوع متغیر `number`، رشته یا همان `str` هست. پس باید نوع اون رو به عدد صحیح یا `int` تبدیل کنیم. پس قاعدتاً دستور زیر جواب می ده:

```
int(number)
```

استاد: حالا مقدار number هم تغییر کرده؟

دانشجو: آخر یادم رفت که باید حاصل رو در متغیر ذخیره کنم و گرنه انگار هیچ کاری نکردم! پس:

```
number = int(number)
```

استاد: بسیار عالی! و حالا یک سؤال دیگر! فرض کنید یک متغیر به نام height برای ذخیره کردن قد فرد داریم:

```
height = '1.7'
```

چطور می‌توانیم کاری کنیم که height نیز مانند یک عدد معمولی، قابلیت جمع و تفریق و... را داشته باشد؟

دانشجو: منطقاً مانند مثال قبل کافی است آن را به int تبدیل کنیم و در متغیر ذخیره کنیم، یعنی:

```
height = int(height)
```

استاد: حالا مقدار height را با دستور زیر چک کن:

```
height
```

دانشجو: عجیبه! نوشت ۱! در حالی که مقدار این متغیر ۱.۷ بود.

استاد: به نظرت چرا این‌طور شد؟

دانشجو: نمی‌دونم قاعدتاً باید درست می‌شد! ☹️

استاد: Int متغیری حاوی چه نوع اعدادی است؟

دانشجو: اعداد صحیح... صحیح! آهان! در حالی که عدد ما اعشاری بود! پس گرد شد!

استاد: و چه نوع متغیری اعداد اعشاری را ذخیره می کند؟

دانشجو: اگر اشتباه نکنم همون که اسمش جدید بود... فکر کنم float! یعنی باید به این شکل بنویسیم؟

```
height = float(height)
```

استاد: چک کن ببین جواب می دهد یا نه.

دانشجو: آره جواب داد! به درستی نوشت:

1.7

گرفتن ورودی از مخاطب

استاد: چطور از مخاطب ورودی بگیریم؟ همیشه که نمی خواهیم خودمان به متغیرها مقدار بدهیم! برای این منظور، دستور خوبی به نام input در زبان برنامه نویسی پایتون وجود دارد:

```
input('Please enter your name')
```

که روی صفحه، نمایش می دهد:

```
>>> Please enter your name
```

داخل پرانتز هر چیزی که بنویسید، به کاربر نمایش داده شده و منتظر دریافت ورودی از شما می ماند؛ مثلاً می توانید بنویسید Pedram و کلید Enter را بزنیم.

حال سؤال این است که این 'Pedram' کجا ذخیره شد؟ من باید این را بدانم؛ چون می خواهم بعداً با آن کاری انجام دهم. به عبارتی من آن را در متغیری ذخیره نکرده ام که بتوانم بعداً از آن استفاده کنیم. برای این کار می توانم بنویسم:

```
name = input('please enter your name:')
```

که حاصل را در متغیر name ذخیره کند. مثلاً اگر وارد کنیم Pedram و کلید En-

ter را بزنیم، چیزی روی صفحه نمایش داده نمی‌شود؛ اما اگر مقدار متغیر name را چک کنیم:

```
print(name)
```

می‌بینیم:

```
'Pedram'
```

حالا بیاییم سن مخاطب را هم بپرسیم. سن را چطور بپرسیم؟

دانشجو: می‌تونیم بنویسیم:

```
age = input('Please enter your age:')
```

و در متغیر age ذخیره کنیم.

استاد: همین دستور را بنویس و مثلاً عدد ۴۷ را به‌عنوان ورودی به آن بده.

دانشجو: نوشتیم. چه باحال! از من پرسید و من وارد کردم ۴۷!

استاد: حال مقدار age چیست؟

دانشجو: چک کردم نوشت:

```
'43'
```

نکنه نوع این متغیر از نوع str باشه؟ بذارین با دستور type هم مطمئن بشم:

```
type(age)
```

نوشت str؛ یعنی ما هر چیزی که ورودی بگیریم، به شکل str است؟

استاد: دقیقاً همان‌طور که خودت هم دیدی، ورودی در پایتون به شکل str یا

همان رشته ذخیره می‌شود. حال اگر بخواهیم نوع آن را تغییر دهیم چه کار کنیم تا

بعداً بتوانیم عملیات ریاضی روی آن انجام دهیم؟

دانشجو: دیگه یادمه! چون age اعشار نداره و از نوع اعداد صحیحه، از دستور `int` استفاده می کنیم و در همون age ذخیره می کنیم که بمونه، یعنی:

```
age = int(age)
```

استاد: عالی! بعضی ها هم این دو خط دستور را در یک خط خلاصه می کنند؛ یعنی این دو خط:

```
age = input('Please enter your age:')  
age = int(age)
```

را به این شکل در یک خط می نویسند:

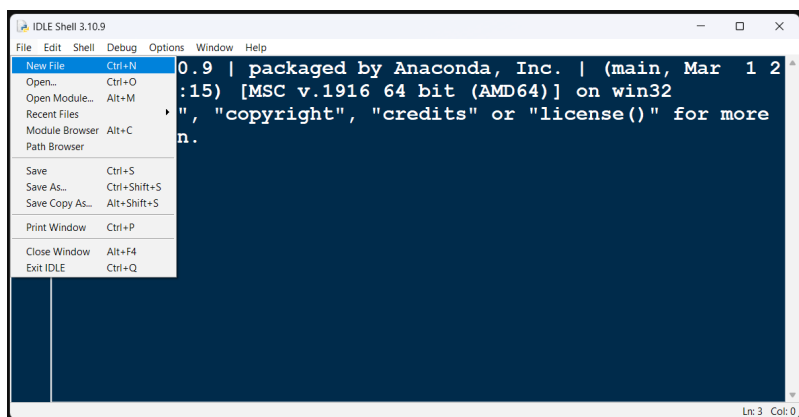
```
age = int(input('Please enter your age:'))
```

هر کدام از این دو روش را که خودتان راحت تر بودید انجام دهید. هیچ مسئله ای نیست که از روش اول استفاده کنید.

نوشتن یک برنامه واقعی!

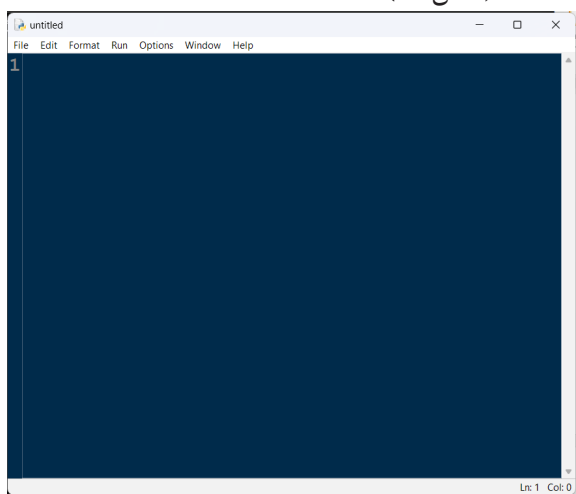
ما تا الآن دستورات را به صورت تک خطی می نوشتیم، اما اکثر برنامه های واقعی به این شکل نیستند! معمولاً به این شکل است که ما دستورات لازم را یکجا پشت سرهم می نویسیم تا با اجرای برنامه، همگی به ترتیب اجرا شوند. ممکن است بگویید از توضیحاتم چیزی متوجه نشدید! طبیعی است! اگر نشانتان بدهم، موضوع جا می افتد.

برای اینکه بتوانیم برنامه خود را بنویسیم، ابتدا از منوی `File`، گزینه `new file` را انتخاب می کنیم (شکل ۱۹):



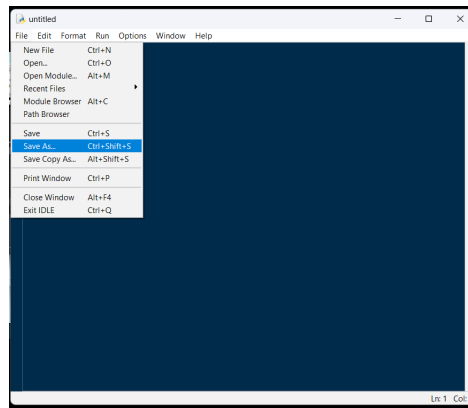
شکل ۱۹- ایجاد فایل جدید

با این کار یک فایل ایجاد می‌شود که چون هنوز نامی برای آن انتخاب نکرده‌ایم به نام untitled است (شکل ۲۰):



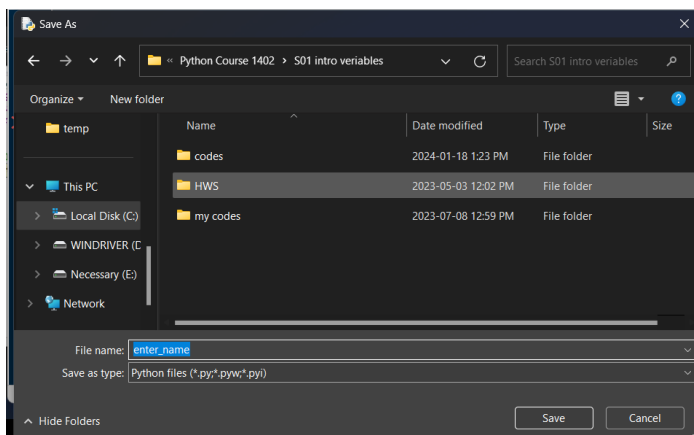
شکل ۲۰- فایل ایجادشده

خب! الآن می‌توانیم برنامه خودمان را بنویسیم؛ اما توصیه‌ی اکید می‌کنم که همین‌الآن آن را ذخیره کنید! یک برنامه‌نویس حرفه‌ای از نام‌هایی مانند untitled و... پرهیز می‌کند. برای ذخیره کردن برنامه خود روی منوی File و سپس Save As کلیک می‌کنیم (شکل ۲۱):



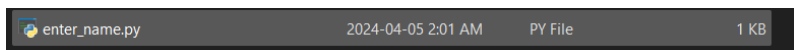
شکل ۲۱- ذخیره کردن فایل

و در پنجره باز شده نام مدنظرمان را انتخاب می‌کنیم. از گذاشتن نام‌های بی‌معنی مانند new file، test1، newprogram و... جداً پرهیز نمایید. نامی بگذارید که بعداً متوجه آن شوید. مثلاً چون این بار می‌خواهیم برنامه‌ای بنویسیم که نام مخاطب و سن او را دریافت کند، نام برنامه را enter- name می‌گذاریم (شکل ۲۲):



شکل ۲۲- سعی کنید نام‌های بامسما برای اسم برنامه‌های خود انتخاب کنید

اگر به پوشه مربوطه در کامپیوتر خود سری بزنید، می‌بینید برنامه شما ایجاد شده است. همان‌طور که احتمالاً دقت کرده باشید، معمولاً نام برنامه‌های پایتون دارای پسوند py (که مخفف Python) است (شکل ۲۳):



شکل ۲۳- برنامه ایجاد شده

دانشجو: آیا میشه در اسم برنامه فاصله گذاشت؟ مثلاً نام برنامه رو به‌جای enter_name، به‌صورت enter name گذاشت؟

استاد: بله! شما می‌توانید هر اسمی برای برنامه‌های خود بگذارید؛ اما توصیه می‌کنم حتی‌الامکان از علامت فاصله در نام برنامه خود استفاده نکنید. بعدها در برخی کاربردهای پیشرفته‌تر پایتون، بهتر است که در اسم برنامه‌های شما فاصله وجود نداشته‌باشد.

برویم سراغ نوشتن اولین برنامه خودمان! می‌خواهیم برنامه‌ای بنویسیم که نام فرد، سن و رشته تحصیلی او را پرسیده و متنی روی صفحه چاپ کند که «سلام فلانی! سن شما فلان قدر و رشته شما فلان است». اولین خط را خودم می‌نویسم. ما اول از همه باید نام فرد را دریافت کنیم و آن را در جایی ذخیره کنیم (مثلاً متغیر name) که بعداً نشان دهیم. پس من می‌نویسم:

```
name = input('Please enter your name:')
```

شما لطفاً برنامه را ادامه دهید.

دانشجو: طبیعتاً نیاز به سن اون هم داریم؛ پس می‌تونیم اون رو هم در خط بعد با دستور input از مخاطب دریافت کنیم و در متغیر دیگه‌ای (مثلاً age) ذخیره کنیم:

```
age = input('Please enter your age:')
```

استاد: بسیار عالی، برای رشته تحصیلی چه کنیم؟

دانشجو: می‌تونیم اون رو هم از مخاطب بگیریم و در متغیر دیگه‌ای مثلاً field ذخیره کنیم:

```
field = input('Please enter your field of study:')
```

استاد: درود بر تو. حال درنهایت می‌خواهیم این موارد را روی صفحه چاپ کنیم. با چه دستوری می‌تونیم این کار را انجام دهیم؟

دانشجو: تا جایی که یادمه از دستور print استفاده می‌کردیم. مثلاً می‌تونیم بنویسیم:

```
print(Hello, name, Your age is, age, Your field of study  
is, field)
```

استاد: Hello و Your age is و Your field of study is که نوشتی، متن هستند یا متغیر؟

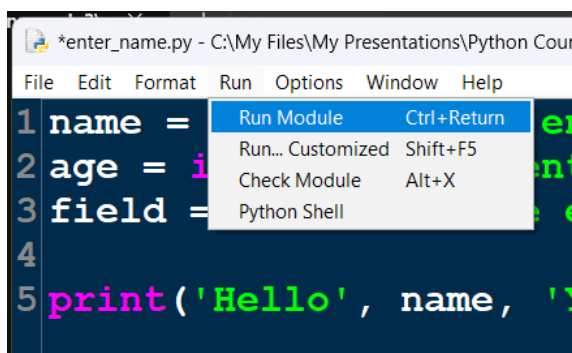
دانشجو: آخ!! برای خودم هم مایهٔ تعجب بود که چرا رنگ بعضی از کلمات مثل `is` و... این طوری شد! حواسم نبود که باید متن‌ها رو داخل دو تا علامت " بذارم وگرنه پایتون فکر می‌کنه که منظور ما یک متغیره نه یک متن:

```
print('Hello', name, 'Your age is', age, 'Your field of study is', field)
```

استاد: دقیقاً و به عبارتی کل برنامهٔ ما چهار خط شد:

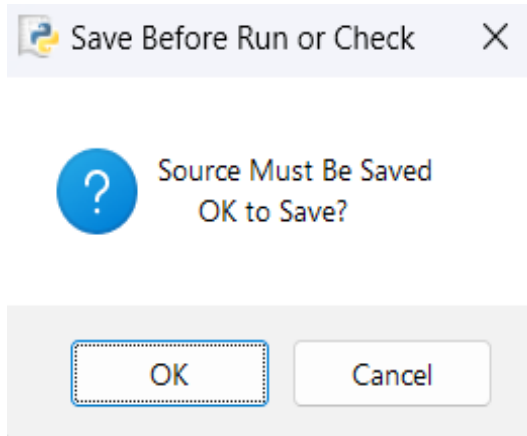
```
name = input('Please enter your name:')
age = input('Please enter your age:')
field = input('Please enter your field of study:')
print('Hello', name, 'Your age is', age, 'Your field of study is', field)
```

برای اجرا کردن کد کافی است، از منوی `Run` گزینهٔ `Run Module` را بزنید یا می‌توانید به جای این کار، کلید میان‌بر `F5` را روی صفحه کلید فشار دهید (شکل ۲۴).



شکل ۲۴- نحوهٔ اجرای اولین برنامه‌مون!

سپس برنامه از شما می‌خواهد که برنامهٔ شما ذخیره شود. شما روی `OK` بزنید (شکل ۲۵).



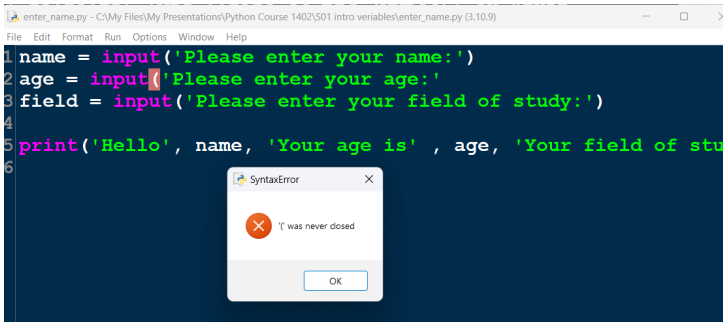
شکل ۲۵- این گزینه را OK کنید

تبریک می گویم! شما اولین برنامه خود را نوشتید.

دانشجو: چه تبریکی استاد! 😊 کامپیوتر به من خطا داد!

استاد: چه خطایی داد؟ کدی که نوشتی را به من هم نشان بده.

دانشجو: این خطا رو داد و یکی از پرانتزها رو قرمز کرد (شکل ۲۶):



شکل ۲۶- خطای نشان داده شده

استاد: خطا دارد چه می‌گویید؟

دانشجو: نوشته پرانتز هرگز بسته نشده.

استاد: به نظرت خودش به تو علت خطا را نگفته؟

دانشجو: آهان راست می‌گید! من در خط دوم، پرانتز آخر رو نگذاشته بودم!

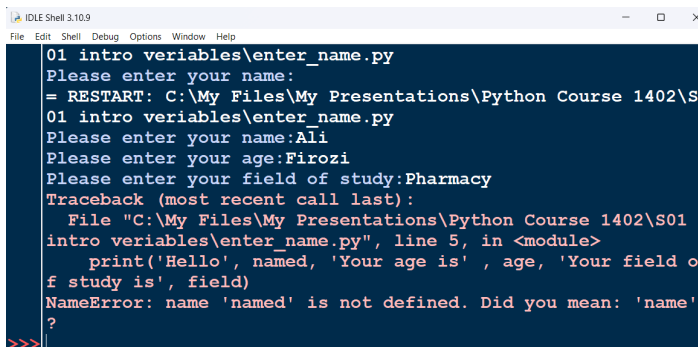
استاد: وقتی پرانتز بسته نشده یا وقتی همچنین اشکالی داشته باشیم، با این نوع از

خطاها روبرو می‌شویم. ☺

دانشجو: الآن پرانتز آخر رو گذاشتم، بذارین یه بار دیگه اجرا کنم. انگار درست شد!

چه باحال؛ اسم رو از من پرسید! سن و بعد رشته! انگار داره درست کار می‌کنه! ای‌وای

دوباره خطا داد (شکل ۲۷)!



```
01 intro variables\enter_name.py
Please enter your name:
= RESTART: C:\My Files\My Presentations\Python Course 1402\S
01 intro variables\enter_name.py
Please enter your name:Ali
Please enter your age:Firozi
Please enter your field of study:Pharmacy
Traceback (most recent call last):
  File "C:\My Files\My Presentations\Python Course 1402\S01
intro variables\enter_name.py", line 5, in <module>
    print('Hello', named, 'Your age is' , age, 'Your field o
f study is', field)
NameError: name 'named' is not defined. Did you mean: 'name'
?
>>>
```

شکل ۲۷- خطای ایجادشده در حین اجرای برنامه

استاد: خیلی خوشحال هستم که در اجرای برنامه با خطا روبرو شدی. یکی از

مهارت‌های مهم در برنامه‌نویسی (و البته در اوایل شاید اعصاب‌خردکن ☺!)، کشف

خطا و رفع آن است. خوب به متن خطا توجه کن. همان‌طور که می‌بینی در ابتدای

خطا نوشته Traceback و در خط بعد آدرس برنامه‌ات را ذکر کرده و مهم‌تر از همه خطی از برنامه که با خطا روبرو شده را بیان کرده است. مثلاً در اینجا کدام line دارای خطا بوده است؟

دانشجو: نوشته در خط ۵ در اجرای دستور زیر خطا رخ داده:

```
print('Hello', named, 'Your age is', age, 'Your field of study is', field)
```

استاد: و در ادامه علت خطا را ذکر کرده:

NameError: name 'named' is not defined. Did you mean: 'name'?

دانشجو: یعنی خطا ناشی از چی بوده؟ بذارین به برنامه‌ام نگاه کنم:

```
name = input('Please enter your name:')
age = input('Please enter your age:')
field = input('Please enter your field of study:')
print('Hello', named, 'Your age is', age, 'Your field of study is', field)
```

نوشته در خط پنجم یعنی همون خطی که دستور print هست، با خطا روبرو شدیم. به من در خطا گفته که named تعریف نشده... یعنی چی؟!... ای وای! چرا من غلط تایپی داشتم! به جای name نوشتم named!!!

استاد: درود بر تو! پایتون به تو گفته متغیری به اسم named برای من تعریف نشده تا مقدارش را نشان بدهم. راست هم می‌گویدی! نام متغیر ما name بود نه named.

دانشجو: چه جالب! الآن درستش کردم. بذارین یه بار دیگه کد رو اجرا کنم... باورم نمیشه جواب داد!!! خیلی باحال بود.

استاد: درود بر تو! نوشتن اولین برنامه خیلی لذت‌بخش است!

نوشتن برنامه بعدی، گرفتن وزن و قد مخاطب

به سراغ نوشتن برنامه بعدی برویم. همان‌طور که آموختیم با کلیک روی منوی File و زدن گزینه New File، یک فایل جدید باز می‌کنیم. هدفمان این است تا برنامه‌ای بنویسیم که قد و وزن فرد را گرفته و BMI او را حساب کند. همان‌طور که می‌دانید BMI هر فرد از رابطه زیر به دست می‌آید (وزن با واحد کیلوگرم و قد با واحد متر وارد شود):

$$BMI = \frac{\text{وزن}}{\text{قد}^2}$$

به نظرتان چگونه این برنامه را بنویسیم؟

دانشجو: قاعدتاً در ابتدا باید وزن فرد رو دریافت کنیم و در یک متغیر (مثلاً weight) ذخیره کنیم:

```
weight = input('Please enter your weight in kg:')
```

تا اینجا که اومدیم، بذارین قد رو هم از مخاطب بگیریم:

```
height = input('Please enter your height in meters:')
```

و در نهایت بیایم BMI را حساب کنیم؛ به عبارتی وزن رو تقسیم بر وزن به توان ۲ کنیم:

```
weight/(height**2)
```

حواسم بود که همیشه توان در پایتون به صورت `**` نشان داده میشه. پرانتز هم گذاشتم تا مطمئن بشم کامپیوتر اول توان رو حساب می‌کنه و بعد تقسیم رو انجام می‌ده (باوجود این که می‌تونم به ضرب و تقسیم اولویت داره، ولی خب دوست دارم احتیاط کنم!).

خب ببینم چه کاری دیگه‌ای مونده... ما این رو حساب کردیم. حالا نمی‌خوایم به جا

ذخیره‌اش کنیم؟ بذارین حاصل رو در یک متغیر مثلاً به نام bmi ذخیره کنیم:

```
bmi = weight/height**2
```

استاد: به نظرت همین کافی است یا کار دیگه‌ای هم باید انجام بدهیم؟ نمی‌خواهی

نتیجه‌ای که به زحمت حاصل شده را به مخاطب نشان بدهی؟ 😊

دانشجو: آهان! آره می‌تونیم با استفاده از دستور print نتیجه رو چاپ کنیم:

```
print(bmi)
```

البته بذارین یه کم خفن‌تر بنویسم و یه جمله هم چاپ کنم؛ مثلاً بگم BMI شما

این مقدار هست:

```
print('Your BMI is ', bmi)
```

تازه حواسم هم بود که متن رو داخل دو علامت ' بذارم. ماشاء... عجب برنامه‌نویس

خفنی هستم! این کل برنامه من شد:

```
weight = input('Please enter your weight in kg:')
height = input('Please enter your height in meters:')
bmi = weight/height**2
print('Your BMI is ', bmi)
```

استاد: درود! نظرت چیست که این برنامه را اجرا کنیم؟

دانشجو: چرا که نه! خب کلید F5 را بزنم تا برنامه اجرا بشه. وزن رو از من خواست.

۷۲ وارد کردم. قد رو هم 1.2 وارد کردم... الآن بریم که نتیجه رو نشون بده که ... ای‌وای

چی شد! 😞 خطا داد که! (شکل ۲۸)

```
Please enter your weight in kg:72
Please enter your height in meters:1.2
Traceback (most recent call last):
  File "E:/test.py", line 3, in <module>
    bmi = weight/height**2
TypeError: unsupported operand type(s) for ** or pow(): 'str' and 'int'
```

شکل ۲۸- خطای ایجادشده

این چه خطاییه دیگه!

Traceback (most recent call last):

File "E:/test.py", line 3, in <module>

bmi = weight/height**2

TypeError: unsupported operand type(s) for ** or pow(): 'str' and 'int'

استاد: در روند برنامه‌نویسی، خطاها بسیار ناخوشایند هستند؛ اما وقتی آن‌ها را رفع می‌کنیم بسیار لذت‌بخش می‌شوند. بیا باهم بررسی کنیم. گفته شده است در کدام خط با خطا روبرو شده‌ایم؟

دانشجو: گفته در خط ۳ یعنی این دستور:

```
bmi = weight/height**2
```

استاد: چه خطایی داده؟

دانشجو:

TypeError: unsupported operand type(s) for ** or pow(): 'str' and 'int'

انگار نوشته علامت توان بین str و int پشتیبانی نمیشه. یعنی چی؟

استاد: گویا یکی از متغیرهای تو از جنس رشته بوده! بیا یک کاری کنیم. قبول داری خط‌به‌خط برنامه‌تو اجرا شده تا این‌که به خط ۳ رسیده و خطا داده است؟ پس منطقاً خط ۱ و ۲ با موفقیت اجرا شده‌اند و ما الآن متغیرهای weight و height را در حافظه کامپیوتر داریم. نظرت چیست که ببینی مقدار این دو متغیر چقدر است؟ در همان محیطی که برنامه‌ما اجرا شده (یعنی محیط shell)، می‌توانیم مقدار متغیرها را با نوشتن اسم آن‌ها ببینیم. یادت است که در ابتدای این جلسه، برنامه‌های خود را به‌صورت دستور، تک‌تک اجرا می‌کردیم و می‌نوشتیم؟

دانشجو: تلاشم رو می‌کنم؛ در محیط shell می‌نویسم:

weight

که ببینم مقدار این متغیر چقدره. نوشت:

'72'

ای‌وای این دو تا علامت 'چی هستن! نکنه محتوای متغیر weight عدد نیست و رشته‌اس؟... ای‌وای! یادم رفت که ورودی‌ها در پایتون از نوع رشته هستن. پس باید تبدیلیش کنم به عدد؛ یعنی این دو خط رو هم قبل از خط ۳ اضافه کنم:

```
weight = int(weight)
```

```
height = int(height)
```

استاد: درود بر تو! اما یک سؤال! دستور int متغیر ما را به چه نوع متغیری تبدیل می‌کرد؟

دانشجو: به نوع متغیر int تبدیل می‌کرد که اعداد صحیح رو نگه می‌داشت. اعداد صحیح یعنی اونایی که اعشار ندارن. ای‌وای وزن و قد ممکنه اعشار داشته باشن! پس بهتره به float تبدیل می‌کردم؛ یعنی به جای دو خط بالا می‌نوشتم:

```
weight = float(weight)
```

```
height = float(height)
```

به عبارتی برنامه کامل ما به این شکل در می‌آید:

```
weight = input('Please enter your weight in kg:')
```

```
weight = float(weight)
```

```
height = input('Please enter your height in kg:')
```

```
height = float(height)
```

```
bmi = weight/height**2
```

```
print('Your BMI is ', bmi)
```

ایول! امتحانش کردم به‌خوبی جواب داد!

استاد: دقیقاً! چیزی که برای من ارزشمند است همین مهارت رفع اشکال است که در اصطلاح برنامه‌نویسی به آن دیباگ کردن کد می‌گویند. به‌عنوان پیشنهاد، می‌توانستیم به‌جای اینکه در دو خط بنویسیم:

```
weight = input('Please enter your weight in kg:')  
weight = float(weight)
```

در یک خط بنویسیم:

```
weight = float(input('Please enter your weight'))
```

البته این دو روش در این مثال، هیچ فرقی ندارند و هر کدام را که دوست داشتید می‌توانید انجام دهید!

معجزه هوش مصنوعی chatGPT در رفع ایراد کد!

بینیم چطور می‌توانیم از چت‌بات‌های هوش مصنوعی برای رفع ایراد کدمان استفاده کنیم. کافی است که برای هوش مصنوعی بنویسیم که این خطا را گرفتیم (خطای خود را کپی کنیم) مثلاً:

```
I got this error: Traceback (most recent call last):  
File "E:/test.py", line 3, in <module>  
bmi = weight/height**2  
TypeError: unsupported operand type(s) for ** or pow():  
'str' and 'int'
```

و در ادامه متن بنویسیم که این هم کد ما بوده است و کد خود را کپی کنیم. لازم به ذکر است که این چت‌بات‌های هوش مصنوعی به فارسی هم پاسخ می‌دهند؛ اما ترجیح من این است که انگلیسی با آن صحبت کنم:

And here is my code:


```
weight = input('Please enter your weight in kg:')  
height = input('Please enter your height in kg:')  
bmi = weight/height**2  
print('Your BMI is ', bmi)
```

من از چت بات هوش مصنوعی بینگ استفاده کردم اما شما می توانید از چت بات های دیگر مثل چت جی پی تی و ... هم استفاده کنید. در صورتی که تابه حال از این ابزارهای فوق العاده مفید در زندگی و کارتان استفاده نکرده اید و دوست دارید استفاده از آن ها را شروع کنید، در همان کانال آپارات من، آموزش استفاده از این ابزار هوش مصنوعی را پیدا خواهید کرد. این بار کد را با موبایل اسکن کنید و یا این که در گوگل جستجو کنید «آموزش استفاده از چت بات های هوش مصنوعی چت جی پی تی و بینگ، پدram پربها» (شکل ۲۹). این ابزارها در همه جا می توانند به شما کمک می کنند و فقط محدود به برنامه نویسی نیستند:



شکل ۲۹- آموزش استفاده از هوش مصنوعی

خب! ببینیم هوش مصنوعی چه پاسخی داد (شکل ۳۰):

The error you're encountering is due to attempting to perform arithmetic operations on strings. In Python, the `input` function returns data as a string, so you need to convert the inputs to floats before performing the division and exponentiation. Here's the corrected code:

Python

```
weight = float(input('Please enter your weight in kg:'))
height = float(input('Please enter your height in meters:')) # Assuming you meant to input height in
meters, not kg

bmi = weight / height**2
print('Your BMI is ', bmi)
```

AI-generated code. Review and use carefully. [More info on FAQ](#)

شکل ۳۰- پاسخ هوش مصنوعی بپینگ

دانشجو: باورم نمیشه! هم علت خطا رو نوشت و هم کد رو اصلاح کرد! پس تا خطا گرفتم، میرم ازش استفاده می‌کنم!

استاد: باوجود این که چت‌بات‌های هوش مصنوعی بسیار کمک‌کننده هستند، اما توصیه می‌کنم که در ابتدا تلاش خود را انجام دهید که خودتان علت خطا را پیدا کنید؛ در نهایت اگر نتوانستید در گوگل جستجو کنید و اگر باز هم به نتیجه نرسیدید از این‌ها استفاده کنید تا ذهنتان به این ابزارها عادت نکند و وابسته نشود؛ اما اگر جستجو کردید و به نتیجه نرسیدید، هیچ اشکالی ندارد که از این ابزارها استفاده کنید. حتی می‌توانید از خود هوش مصنوعی در ادامه بخواهید تا کدی که برایتان نوشته را توضیح دهد. شاید باورتان نشود، اما من حتی هرازگاهی در مورد مسائل زندگی هم با این ابزارها مشورت می‌کنم. این چت‌بات‌ها مانند یک دوست خوب با من حرف می‌زنند و کمک می‌کنند!

در پایان این جلسه، به سراغ بحث شیرین تمرین‌ها می‌رویم تا مطالب این جلسه برایتان تثبیت شود.

تمرین‌های جلسه اول

در تمرین هر درس ممکن است که یک نکته جدید هم آموزش داده شود، پس حتماً توصیه می‌کنم آن‌ها را ببینید و انجامشان دهید. تمرین‌ها و به خصوص خطاهایی که دریافت می‌کنید و آزمون و خطاها (و همچنین سرچ‌هایی که در صورت نیاز انجام می‌دهید)، بسیار بسیار به روند یادگیری برنامه نویسی شما کمک می‌کند. پاسخ نمونه برای تمرین‌ها نیز پس از آن‌ها قرار داده شده است.

تمرین‌ها

- ۱- برنامه‌ای بنویسید که نام، کتاب مورد علاقه کاربر و سن او را پرسیده و نتیجه را در یک جمله به کاربر نمایش دهد.
- ۲- برنامه‌ای بنویسید که سن فرد و تعداد کتابی که خوانده را پرسیده و سپس میانگین تعداد کتابی که هر سال خوانده را حساب کرده و روی صفحه چاپ کند.
- ۳- در رشته یا حرفه خود، یک تمرین، مشابه سؤال بالا، منتها کاربردی، طراحی و اجرا کنید.

اکیداً توصیه می‌کنم که ابتدا خودتان تمرین‌ها را حل کنید، سپس به پاسخ‌ها مراجعه کنید. در ضمن این پاسخ‌ها تنها یکی از حالت‌های ممکن نوشتن برنامه موردنظر هستند و شما ممکن است به شیوه‌ای دیگر برنامه را نوشته باشید و در عین حال کاملاً صحیح باشد. همچنین می‌توانید برای مشاهده توضیحات شفاهی مربوط به تمرین هر جلسه، ابتدای ویدیوی جلسه بعد را ببینید (لینک ویدئوها در صفحه اول کتاب گذاشته شده است).

پاسخ تمرین‌های جلسه اول

« پاسخ تمرین اول:

یک روش نوشتن این برنامه می‌تواند به این شکل باشد:

```
name = input('please say what is your name?')
book = input('please say what is your favorite book?')
age = int(input('please say how old are you?'))
print('You are',name,',You are',age,'years old and your
favorite book is',book,',')
```

« پاسخ تمرین دوم:

یک نوع از حل این سؤال:

```
name = input('please say your name:')
books = int(input('how many books do you read?'))
age = int(input('how old are you?'))
mean_books = books/age
print('My name is', name, 'I have read', books,
      'books, so far, as a result, I have read an
average of ', mean_books,'book/books every year')
```

سؤال تکمیلی:

چگونه می‌توان مقدار mean-books را گرد کرد؟ چه دستوری برای این کار در پایتون وجود دارد؟ (راهنمایی: نیاز به جستجو در گوگل دارید 😊)

پاسخ سؤال تکمیلی:

کافی است در گوگل مثلاً به این شکل: how to round a number in python؟

جستجو کنیم؛ می بینیم لینک های زیادی وجود دارند. من روی لینک اول زدم. می بینم که چندین روش را نوشته است. صفحه را پایین آوردم و به روش اول رسیدم و دیدم برایمان مثال کد نیز آورده شده است (شکل ۳۱):

1. Rounding to the Nearest Integer

The simplest way to round a number in Python is to use the built-in `round()` function. The `round()` function takes a number as the first argument and an optional second argument to specify the number of decimal places to round to. If the second argument is omitted, the number is rounded to the nearest integer.

```
# Example 1: Rounding to the Nearest Integer
num = 3.7
rounded_num = round(num)
print(rounded_num) # Output: 4
```

شکل ۳۱- با جستجو می توان جواب بسیاری از سؤالات را پیدا کرد

متوجه می شویم که دستوری به نام `round` برای این کار در پایتون وجود دارد که می توان از آن استفاده کرد و حتی تعداد اعشار را نیز مشخص کرد.

« پاسخ تمرین سوم:

برای نمونه، نوشتن برنامه ای که تعداد نوکلئوتیدهای یک رشته DNA را گرفته و سپس تعداد آمینواسیدهای حاصل از این DNA را به ما می دهد (همان طور که احتمالاً می دانید، معمولاً هر سه نوکلئوتید، یک آمینواسید را نتیجه می دهند و به همین دلیل می توانیم با تقسیم تعداد نوکلئوتیدها بر ۳، به صورت تقریبی تعداد آمینواسیدهای حاصل را به دست بیاوریم):

```
num_bases = input('Enter the number of DNA bases: ')
num_bases = int(num_bases)
num_amino_acids = num_bases // 3

print('The number of amino acids produced is:', num_
amino_acids)
```

نکته تکمیلی:

هر موقع بخواهیم در کد خود یک خط توضیح بنویسیم و نخواهیم پایتون آن را اجرا کند، کافی است در شروع آن خط یک علامت `#` بگذاریم؛ با این کار به پایتون می‌فهمانیم که این خط را نمی‌خواهیم اجرا کنیم و هدفمان توضیح آن قسمت از برنامه برای خودمان است. مثلاً می‌توانیم به برنامه بالا با همین روش مقداری توضیح اضافه کنیم:

```
# Prompt the user to enter the number of DNA bases
num_bases = input('Enter the number of DNA bases: ')

# Convert the input to an integer
num_bases = int(num_bases)

# Calculate the number of amino acids produced
num_amino_acids = num_bases // 3

# Print the result
print('The number of amino acids produced is:', num_
amino_acids)
```

می‌بینید که این خطوط که اول آن‌ها `#` است، به رنگ دیگری درآمده‌اند. در پایتون به این توضیحات `comment` می‌گویند. اگر این خطوط که با علامت `#` شروع شده‌اند را حذف کنید، هیچ تأثیری در اجرای برنامه ندارند. البته بهتر است آن قدر نام متغیرها را خوب انتخاب کرده باشیم و برنامه را تمیز نوشته باشیم که حتی الامکان نیازی به نوشتن توضیحات بیشتر نباشد؛ اما بعضی مواقع ناچاریم این توضیحات را بنویسیم تا بعداً یادمان باشد هدف از این قسمت برنامه چه بوده است.

• فصل دوم •

لذت تصمیم‌گیری با استفاده از شرطها!

شروع کار با محیطی زیباتر برای کار با پایتون!

استاد: سلام و درود به همه شما دوستان عزیزم! همان‌طور که دیدید، ما تاکنون با محیطی نازیبا، به نام IDLE برنامه‌نویسی می‌کردیم. ما محیط‌های کدنویسی بهتری داریم که می‌توانیم از آن‌ها استفاده کنیم. (البته حواستان باشد که این محیط‌ها تنها یک محیط کدنویسی هستند و اصل کار، اجرای برنامه و درواقع ترجمه به زبان ماشین را همان زبان پایتون انجام می‌دهد). محیط‌های مشهور کدنویسی برای پایتون که احتمالاً نام برخی از آن‌ها را شنیده باشید از این‌ها قرار هستند:

۱- IDLE: همان محیطی بود که تا الان با آن کدنویسی می‌کردیم.

۲- Spyder: محیطی که هنگام نصب آن‌کوندا بر روی سیستم ما نصب شده و آماده استفاده است.

۳- VS code: که علاوه بر پایتون می‌توان به زبان‌های دیگر برنامه‌نویسی نیز با آن کار کرد و از محیط‌های دیگر سبک‌تر است و به همین دلیل سریع‌تر باز می‌شود.

۴- Pycharm: محیط بسیار مشهوری که مختص پایتون است ولی چون امکانات

زیادی دارد، مقداری باز شدن آن سنگین تر است.

۵- Jupyter: محیطی جالبی دارد؛ به خصوص برای کارهای تحلیل و نمایش داده به کار می رود و اجازه می دهد کدمان را قسمت به قسمت اجرا کنیم و نتیجه را ببینیم.

۶- Google Colab: که این مورد شبیه همان محیط Jupyter است، منتها در اینترنت می توانید به آن دسترسی داشته باشید. به عبارت دیگر بدون نیاز به نصب پایتون، با اینترنت به پایتون دسترسی دارید و کد شما روی کامپیوترهای گوگل اجرا می شود و نتیجه به شما نمایش داده می شود.

دانشجو: تفاوت زیادی داره که از کدام محیط استفاده کنیم؟

استاد: این موضوع تا حد زیادی به سلیقه شما بستگی دارد و گرنه اصول برنامه نویسی در همه این ابزارها یکی است و همان زبان پایتون است. من خودم به شخصه برای شروع Spyder و سپس محیطی مثل VsCode را توصیه می کنم. مزیت اول این است که وقتی آنرا نصب کردید، به صورت پیش فرض Spyder و Jupyter هم روی سیستم شما نصب شدند، زیرا دیگر نیازی به نصب جداگانه آنها ندارید؛ اما می توانید اگر تمایل داشتید pycharm و vscode را بعداً امتحان کنید.

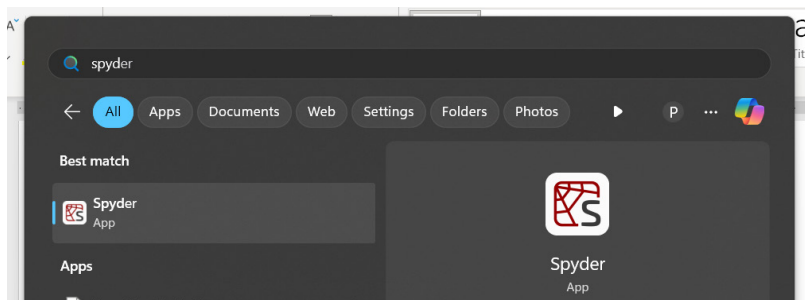
دانشجو: چرا از همون IDLE استفاده نکنیم؟

استاد: درست است که هیچ فرقی ندارد که در کدام محیط کدنویسی برنامه خود را بنویسید؛ اما عملاً کسی از IDLE برای برنامه نویسی استفاده نمی کند، زیرا محیطی است که امکانات کمی برای رفع ایراد کد شما دارد. دلیلی که من در جلسه اول از IDLE استفاده کردم این بود که بدانید با همان محیط زشت IDLE هم می توان برنامه نوشت و یک وقت این شائبه پیش نیاید که برای برنامه نویسی نیاز به سیستم

قدرتمند و عجیب و غریبی وجود دارید. خود من نیز به‌شخصه وقتی که می‌خواهم یک کد ساده را امتحان کنم یا نیاز به یک محیط سبک دارم که مثلاً بگذارم برنامه من از شب تا صبح اجرا شود، از همان IDLE استفاده می‌کنم، اما عملاً در بسیاری مواقع از محیط‌های دیگر برنامه‌نویسی استفاده می‌کنم. راستی تا یادم نرفته به محیط‌های برنامه‌نویسی، Integrated Development Environment یا همان IDE هم می‌گویند؛ به عبارت دیگر تمامی مواردی که ذکر کردیم مثل IDLE، Spyder و vs code ... نوعی IDE هستند. به‌صورت خلاصه شما از هر IDE که دلتان خواست می‌توانید استفاده کنید اما ما از Spyder در این دوره استفاده می‌کنیم.

چطور Spyder را باز کنیم؟

باز کردن Spyder بسیار ساده است. کافی است مانند IDLE، در محیط استارت ویندوز کلمه Spyder را تایپ کنیم و سپس روی آیکون آن کلیک کنیم (شکل ۱):



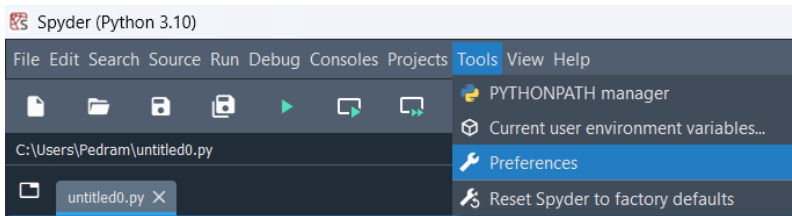
شکل ۱- باز کردن Spyder به‌وسیله استارت ویندوز

روش‌های دیگر باز کردن IDLE یادتان است؟ دقیقاً با همان روش‌ها نیز می‌توان Spyder را باز کرد؛ تنها کافی است به‌جای لغت idle کلمه spyder را بنویسیم.

نکته: در اولین بار برای اجرای اسپایدر، با یک پیغام خطا روبرو می شوید. اگر یک بار برنامه را ببندید و مجدد باز کنید، مشکل برطرف می شود و دیگر این پیغام خطا تکرار نمی شود.

تغییر ظاهر اسپایدر

برای تغییر اندازه، فونت و ظاهر برنامه مطابق سلیقه خود، از منوی Tools، گزینه Preferences را می زنیم تا پنجره مربوطه باز شود (شکل ۲):

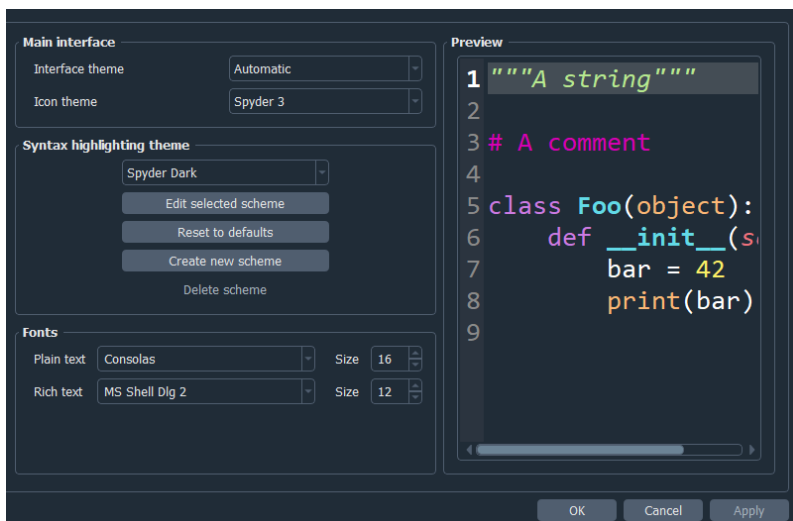


شکل ۲- باز کردن تنظیمات اسپایدر

سپس در پنجره باز شده در قسمت Fonts، اندازه نوشته شده در بالا (Plain text) مربوط به محیط کدنویسی است که من توصیه می کنم برای حفاظت از چشمانتان حتماً آن را افزایش دهید. من آن را روی ۱۶ و فونت پایینی که مربوط به قسمت shell است را روی عدد ۱۲ گذاشتم. شما بسته به سلیقه خودتان می توانید آن را به اعداد دیگری تغییر دهید (شکل ۳).

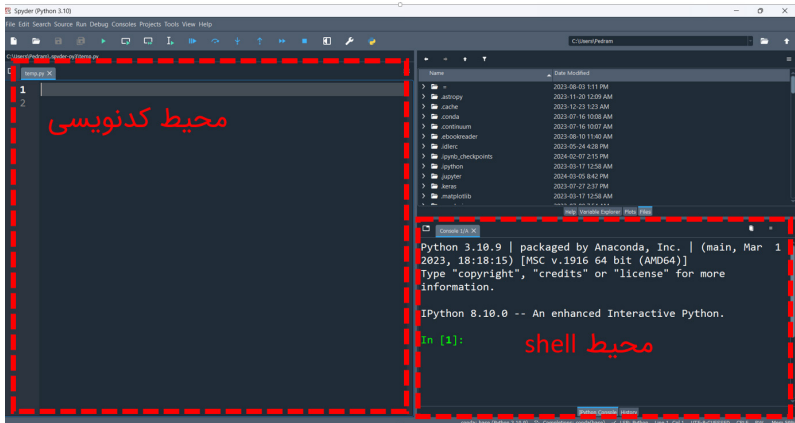
در صورتی که تمایل دارید پوسته ظاهری اسپایدر از حالت تاریک (dark) به حالت سفید (light) در بیاید، interface theme را از حالت Automatic به حالت Light و همچنین قسمت Syntax highlighting theme را به جای Spyder Dark را بر روی Spyder بگذارید و در نهایت OK را کلیک کنید. پس از بستن و باز

کردن مجدد برنامه، تغییرات اعمال خواهند شد و محیط اسپایدر به صورت سفیدرنگ و روشن در خواهد آمد. البته من توصیه می‌کنم اگر با محیط تاریک راحت هستید، آن را تغییر ندهید و بگذارید همان تم تاریک باقی بماند.



شکل ۳- تغییر سایز فونت اسپایدر

بسیار عالی! همان‌طور که خاطرتان است، ما در محیط رسمی idle پایتون، دو قسمت داشتیم: یکی محیط shell که در آن به صورت تک‌خطی برنامه‌نویسی می‌کردیم و یکی هم محیط کدنویسی که برنامه‌های بلندمان را می‌نوشتیم و دیدیم که با اجرای آن، باز هم برنامه ما در همان محیط shell اجرا می‌شد. در اسپایدر نیز همین‌طور است، با این تفاوت محیطی که برنامه خود را می‌نویسیم سمت چپ و محیط shell پایین سمت راست است که برنامه ما هم همین‌جا اجرا می‌شود (شکل ۴).



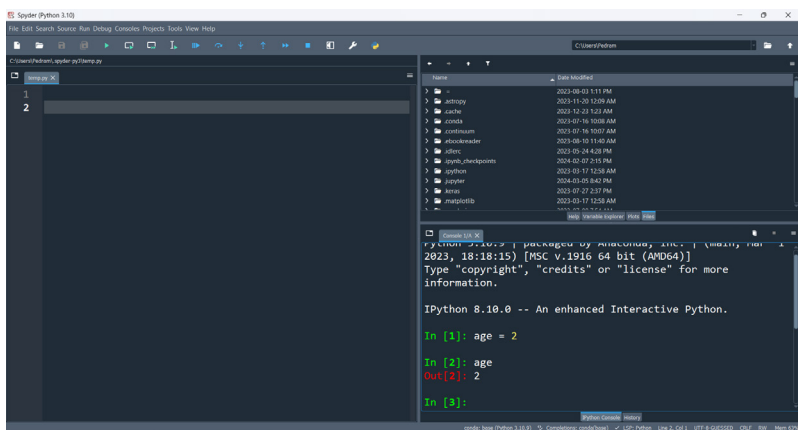
شکل ۴- محیط اسپایدر (سمت چپ محیط برنامه نویسی و پایین سمت راست محیط shell است) مثلاً اگر همانند جلسه قبل در محیط پایین سمت راست (که محیط shell است) بنویسیم:

```
age = 2
```

و کلید Enter را روی صفحه کلید فشار دهیم، متغیری به اسم age تعریف می شود. حال اگر بنویسیم:

```
age
```

مقدار age را خواهیم داد (شکل ۵).

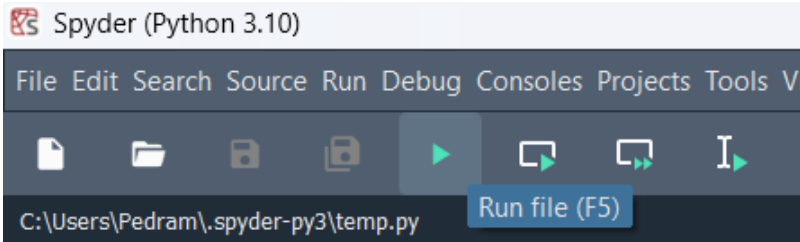


شکل ۵- استفاده از shell در اسپایدر و دیدن مقدار متغیر age

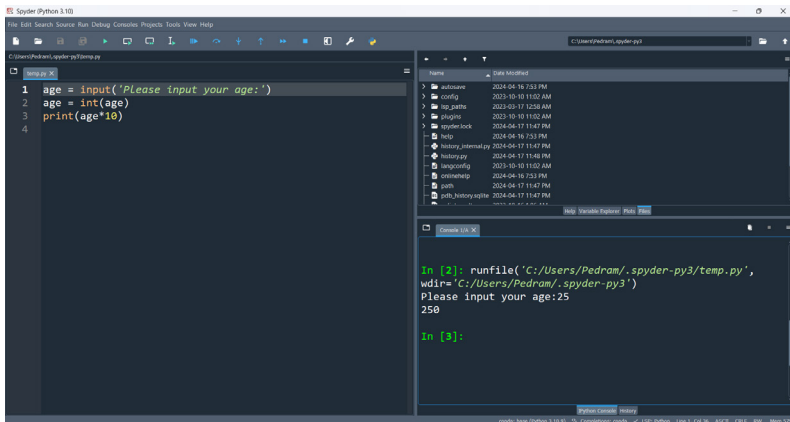
از طرفی در قسمت چپ می‌توانیم برنامه خود را بنویسیم. مثلاً اگر بخواهیم از مخاطب سن او را پرسیده و در متغیر age ذخیره کنیم و سپس آن را به int تبدیل کرده و سن را ضربدر ۱۰ کنیم و به مخاطب نمایش دهیم، می‌توانیم بنویسیم:

```
age = input('Please input your age')
age = int(age)
print(age*10)
```

حال با دکمه فلش سبزرنگ بالای برنامه که در شکل ۶ با نام Run File نمایش داده شده، می‌توانیم برنامه را اجرا کنیم و می‌بینیم که برنامه در همان قسمت shell (همان قسمت سمت راست پایین) اجرا می‌شود (شکل ۷):



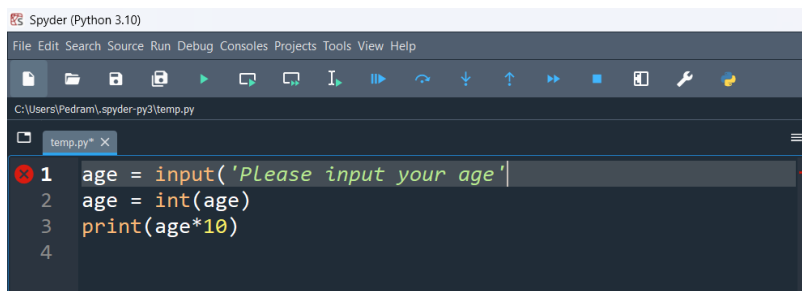
شکل ۶- دکمه سبز رنگ Run file



شکل ۷- اجرا شدن برنامه

همین کار اجرا کردن کامل برنامه را با فشردن کلید میان‌بر F5 روی صفحه کلید نیز می‌توانیم انجام دهیم.

نکته جالب ماجرا این است که اگر در نوشتن برنامه اشتباهی کرده باشیم، مثلاً پرانتز آخر را در خط اول فراموش کنیم، می‌بینیم که اسپایدر با علامت ضریب قرمز رنگ هشدار می‌دهد و قبل از این که برنامه را اجرا کنیم، به ما خطا را نشان می‌دهد، در حالی که در idle باید حتماً برنامه اجرا می‌شد تا پتک خطا بر سرمان فرود آید! این یکی از ویژگی‌های خوب اسپایدر است (شکل ۸):



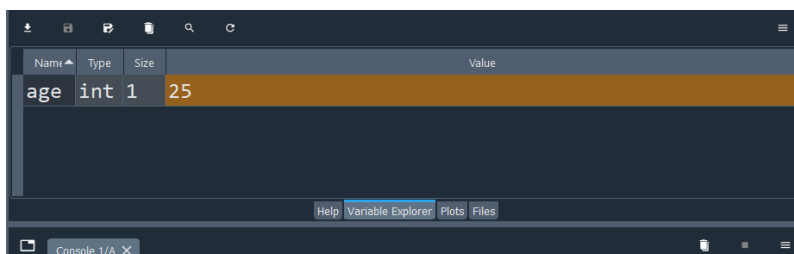
```

1 age = input('Please input your age'|
2 age = int(age)
3 print(age*10)
4

```

شکل ۸- خطاهای دستوری برنامه‌نویسی در اسپایدر با علامت ضربدر قرمز رنگ نشان داده می‌شوند و اگر با موس روی آن نگه دارید علت خطا را هم به ما نشان می‌دهد.

قابلیت خوب دیگر اسپایدر، قسمت Variable Explorer است که می‌توانیم آن را در بالای shell پیدا کنیم. با کلیک کردن روی آن، تمامی متغیرهای حال حاضر به همراه نوع و محتوای آن‌ها نمایش داده می‌شود؛ برای مثال می‌بینیم متغیر age تعریف شده و چون مقدار ۲۵ را به عنوان ورودی داده‌ایم، مقدار آن ۲۵ است (شکل ۹):

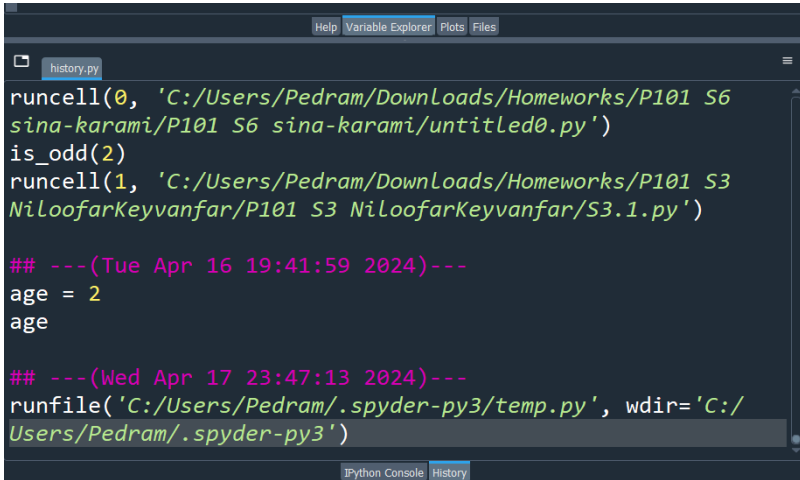


شکل ۹- Variable Explorer

اسپایدر قابلیت‌های دیگری هم دارد که در ادامه به‌مرور زمان با آن‌ها بیشتر آشنا خواهیم شد.

دانشجو: استاد من یه برنامه نوشتم و اجراش کردم، اما برخلاف شما که خیلی اجرا

شد، برای من این متن‌ها ظاهر شده (شکل ۱۰). مشکل چیه؟



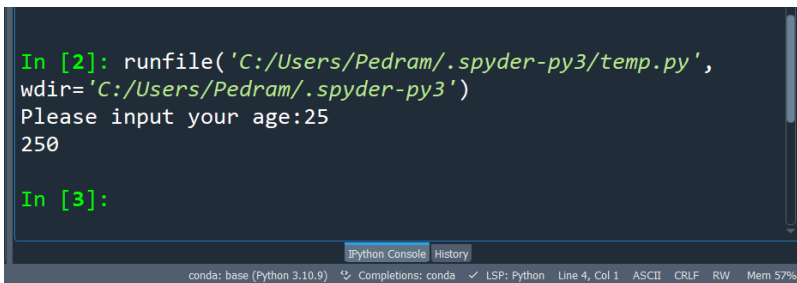
```
history.py
runcell(0, 'C:/Users/Pedram/Downloads/Homeworks/P101 S6
sina-karami/P101 S6 sina-karami/untitled0.py')
is_odd(2)
runcell(1, 'C:/Users/Pedram/Downloads/Homeworks/P101 S3
NilooFarKeyvanfar/P101 S3 NilooFarKeyvanfar/S3.1.py')

## ---(Tue Apr 16 19:41:59 2024)---
age = 2
age

## ---(Wed Apr 17 23:47:13 2024)---
runfile('C:/Users/Pedram/.spyder-py3/temp.py', wdir='C:/
Users/Pedram/.spyder-py3')
```

شکل ۱۰- اشکال پیش آمده

استاد: اگر دقت کنید می بینید که اشتباهاً روی دکمه History در پایین کلیک شده و باعث شده History (تاریخچه) تمام دستورات گذشته نمایش داده شود. کافست در شکل ۱۰ روی IPython Console کلیک کنید تا به همان محیط shell (همان محیط Console) برگردید (شکل ۱۱):



```
In [2]: runfile('C:/Users/Pedram/.spyder-py3/temp.py',
wdir='C:/Users/Pedram/.spyder-py3')
Please input your age:25
250

In [3]:
```

شکل ۱۱- برگشتن به محیط shell (یا همان console)

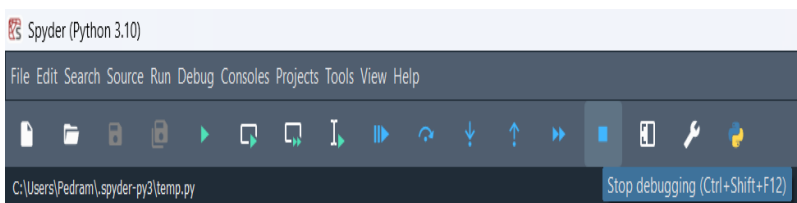
دانشجو: ممنون استاد درست شد! اما الآن که اومدم برنامه رو اجرا کنم، برخلاف شما به جای اجرای برنامه، نوشت debugFile (شکل ۱۲):

```
In [4]: debugfile('C:/Users/Pedram/.spyder-py3/temp.py',
wdir='C:/Users/Pedram/.spyder-py3')
> c:\users\pedram\.spyder-py3\temp.py(1)<module>()
----> 1 age = input('Please input your age:')
      2 age = int(age)
      3 print(age*10)

IPdb [1]:
```

شکل ۱۲- اشکال پیش آمده

استاد: زمانی که اشتباهاً به جای دکمه فلش سبزرنگ (یا همان Run File)، روی فلش آبی‌رنگ (یا همان Debug File) کلیک کنید، به این شکل درمی‌آید. برای رفع شدن این موضوع کافی است روی دکمه مربع آبی‌رنگ (یا همان Stop Debugging) کلیک کنید (شکل ۱۳):



شکل ۱۳- Stop Debugging

دانشجو: ممنونم. این حالت Debug به چه دردی می‌خوره؟

استاد: بحث امروزمان نیست اما این حالت باعث می‌شود که هر خط برنامه شما

اجرا شده و منتظر شما بماند تا بگویید که برنامه به خط بعد برود. سپس با زدن دکمه خط بعد، خط بعد اجرا شده و منتظر می‌ماند. این منتظر ماندن و همچنین قسمت Variable Explorer که مقدار متغیرها را در لحظه نشان می‌دهد، ابزار بسیار مناسبی برای رفع خطاهای برنامه (به‌خصوص برنامه‌های پیچیده) است. مافعالاً کاری با آن نداریم اما بعداً به‌خصوص در برنامه‌های پیچیده می‌توانید از آن استفاده کنید.

خب؛ باهم مروری داشته باشیم. در جلسه قبل با معجزه اول برنامه‌نویسی یعنی متغیرها آشنا شدیم و دیدیم که سه نوع متغیر داریم: int برای اعداد صحیح، float برای اعداد اعشاری (که می‌توانست اعداد صحیح را هم در خود جای دهد) و str که می‌توانست رشته‌ای از حروف و کاراکترها را در بر بگیرد.

امروز با معجزه دوم برنامه‌نویسی آشنا می‌شویم: قدرت تصمیم‌گیری! دیده‌اید که در زندگی روزمره می‌گوییم که اگر اتاق گرم بود، کولر را روشن کن. در برنامه‌نویسی هم همین‌طور است. «اگر»ها که «اگر این‌طور آنگاه فلان کار را انجام بده» و ...

هرچقدر هم توضیح بدهم، باز مثل این نمی‌شود که یک مثال بزنم! فرض کنید سن مخاطب را پرسیده و در متغیری به اسم age ذخیره کنیم:

```
age = input('Please enter your age:')
```

همان‌طور که در جلسه قبل دیدیم، چون age یک عدد صحیح است، بهتر است آن را به int تبدیل کنیم:

```
age = int(input('Please enter your age:'))
```

حال می‌خواهیم اگر سن فرد بیشتر از ۴۰ بود، روی صفحه نوشته شود که «درود بر تو، به فکر آموزش خودت هستی!»، چگونه این کار را در پایتون انجام دهیم؟ کافی است if را نوشته و پس از آن شرط مدنظرمان را بنویسیم (که در اینجا $age > 40$) است

و یک علامت دونقطه بگذاریم و سپس در ادامه، کاری که می‌خواهیم کامپیوتر انجام دهد را بنویسیم (که در اینجا چاپ کردن جمله روی صفحه است)؛ یعنی:

```
if age>40:
    print('It is so great!')
```

دانشجو: استاد چرا print رو با چند فاصله جلوتر نوشتین و اون رو هم سطح جمله بالایی قرار ندادین؟

استاد: دوستان عزیزم! حواستان باشد که پایتون به دندان‌ها حساس است! حتماً مواردی که می‌خواهید در زیرمجموعه if اجرا شوند، باید یک دندان‌ها (یا اصطلاحاً indent) داشته باشند، برای مثال کد زیر با خطا روبرو می‌شود:

```
if age>40:
    print('It is so great!')
```

زیرا دستور زیرمجموعه if باید با یک دندان‌ها جلوتر گذاشته شود (با زدن کلید Tab در سمت چپ صفحه کلید، می‌توانید یک دندان‌ها ایجاد کنید).

حالا برنامه را اجرا می‌کنیم. اگر سن را ۲۰ وارد کنیم، می‌بینیم که هیچ‌چیزی روی صفحه نمایش داده نمی‌شود، ولی اگر برنامه را مجدد اجرا کنیم و سن را مثلاً ۴۲ وارد کنیم، می‌بینیم که این بار پیغام مدنظر ما روی صفحه نمایش داده می‌شود؛ به عبارت دیگر کامپیوتر به خط if می‌رسد و شرط را بررسی می‌کند، اگر این شرط برقرار بود حالا خطوط زیرمجموعه if را اجرا می‌کند، وگرنه این خطوط را رد می‌کند تا به خطوط بعدی برنامه برسد. پس به عبارتی ساختار if در پایتون به این شکل است:

شرط if:

دستورات موردنظر

حال فرض کنید بخواهیم یک `if` دیگر بنویسیم. مثلاً اگر سن کوچک‌تر مساوی ۴۰ بود، پیام دیگری روی صفحه چاپ شود. کافی است در ادامه برنامه قبل بنویسیم:

```
if age<=40:
    print('Excellent!')
```

به عبارتی کل برنامه ما شد:

```
age = int(input('Please enter your age:'))
if age>40:
    print('It is so great!')
```

```
if age<=40:
    print('Excellent!')
```

حواستان باشد برنامه را به شکل زیر ننویسید:

```
age = int(input('Please enter your age:'))
if age>40:
    print('It is so great!')
    if age<=40:
        print('Excellent!')
```

دانشجو: چرا استاد؟ مگه این چه مشکلی داره؟

استاد: بیاید خودمان را جای کامپیوتر گذاشته و اصطلاحاً برنامه را خط به خط ردگیری (trace) کنیم. شما این کار را برای من انجام بده و بگو که چرا این برنامه مشکل دارد؟

دانشجو: در برنامه‌ای که نوشتیم، در خط اول، کامپیوتر از ما سن رو می‌خواد:

```
age = int(input('Please enter your age:'))
```

فرض می‌کنیم سن رو ۳۰ وارد کنیم. به خط بعدی می‌رسیم:

```
if age>40:
```

طبیعتاً چون این شرط برقرار نیست، خطوط بعدی که زیرمجموعهٔ `if` هستند هیچ‌کدام اجرا نشدن؛ تمام خطوط ۳ تا ۵ برنامه رد شده و خط ۶ (اگر وجود داشت) اجرا می‌شده.

حال اگر برای مثال، سن را ۴۵ وارد می‌کردیم، چون شرط سن بزرگ‌تر از ۴۰ برقراره، پس خط ۳ اجرا شده:

```
if age>40:
```

و روی صفحه پیغام «it is so great» چاپ می‌شده:

```
print('It is so great!')
```

بعدش برنامه وارد خط ۴ شده و شرط `age<=40` بررسی می‌شده:

```
if age<=40:
```

به این خاطر که سن وارد شده ۴۰ است این شرط برقرار نیست و خط ۵ یعنی دستور `print('Excellent!')` اجرا نمی‌شده. آهان! پس اشکال کار همین جاست!! درحالی‌که در برنامهٔ درست یعنی:

```
age = int(input('Please enter your age:'))
```

```
if age>40:
```

```
    print('It is so great!')
```

```
if age<=40:
```

```
    print('Excellent!')
```

اگر سن مثلاً ۳۰ وارد بشه، چون شرط `age>40` برقرار نیست، پس دستورات زیرمجموعهٔ `if` اول اجرا نشده و پایتون این خطوط رو رد کرده و وارد خط چهارم

یعنی:

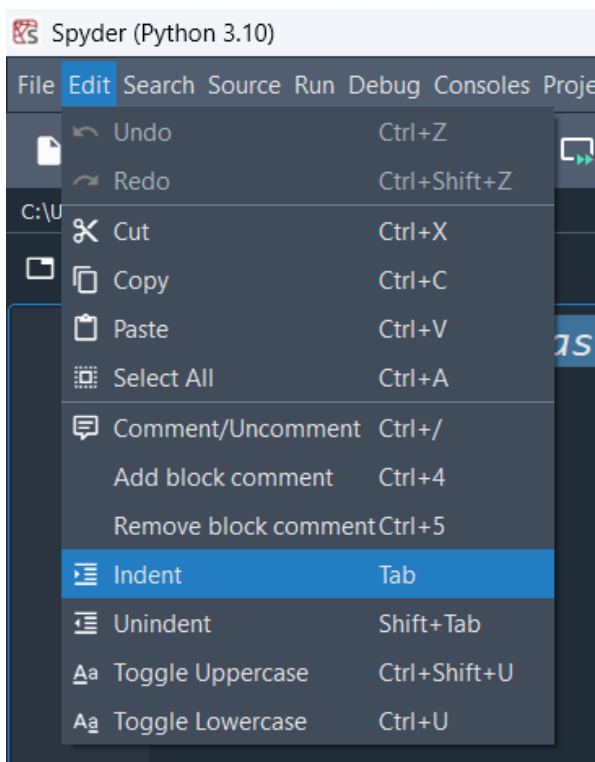
```
if age<=40
```

می‌شه، ولی چون الان شرط $age \leq 40$ برقراره، پس پایتون دستورات زیرمجموعه آن یعنی `print('Excellent!')` را اجرا می‌کنه. سؤالی که دارم اینه که آیا فقط می‌تونیم یک دستور رو زیر `if` بنویسیم یا می‌تونیم چند خط دستور بنویسیم؟

استاد: چه سؤال خوبی! شما می‌تونید حتی ده‌ها خط دستور بعد از `if` بنویسید که در صورت برقرار بودن شرط انجام شوند؛ منتها باید حتماً یک دندانۀ جلوتر باشند تا پایتون بفهمد که منظور شما این است که این دستورات زیرمجموعه `if` هستند. برای مثال فرض کنید می‌خواهیم اگر سن فرد بیشتر از ۴۰ بود، سن فرد را در ۱۲ ضرب کرده و حاصل را چاپ کنیم:

```
if age>40:  
    age = age*12  
    print('It is so great!')
```

راستی تا یادم نرفته است بگویم می‌تونید دندانۀ را با زدن کلید Tab روی صفحه کلید ایجاد کنید (حتی می‌تونید چند خط را باهم انتخاب کنید و با زدن کلید Tab تمام آن‌ها یک دندانۀ جلو می‌روند) همین کار را با زدن `Shift+Tab` می‌تونید برعکس انجام دهید یعنی خطوط انتخاب را یک دندانۀ عقب بیاورید. اگر این میانبر یادتان رفت کافیهست پس از انتخاب خطوط موردنظر، در منوی Edit، روی گزینه `indent` کلیک کنید (شکل ۱۴):



شکل ۱۴- اگر یادتان رفت می‌توانید از این قسمت دندان‌ه اضافه و کم کنید.

دانشجو: استاد من این برنامه رو نوشتم ولی با این خطا مواجه شدم (شکل ۱۵):

```
File "c:\Users\Pedram\Desktop\Untitled-1.py", line 4
    if age<=40
        ^
SyntaxError: expected ':'
PS C:\Users\Pedram>
```

شکل ۱۵- خطای نمایش داده شده

این هم برنامه‌ایه که نوشتیم:

```
age = int(input('Please enter your age:'))
if age>40:
    print('It is so great!')
if age<=40
    print('Excellent!')
```

مشکل چیه؟

استاد: خطاخوانی و رفع باگ مهارت بسیار مهمی است که به‌مرور در آن پیشرفت خواهید کرد. همان‌طور که می‌بینید در خطای نوشته‌شده، گفته‌شده در خط چهارم یعنی دستور

```
if age<=40
```

خطا ایجاد شده است. خطای ایجادشده هم از نوع `syntaxError` است و نوشته شده '؟' `expected`؛ یعنی انتظار علامت دونقطه (:) را داشته است؛ به عبارتی دارد به ما می‌گوید که یک علامت : جا مانده است.

دانشجو: آهان فهمیدم! الآن درست شد! چه باحال!

استاد: اگر هم ایراد رفع نشد، می‌توانی متن خطا را در گوگل جستجو کنی و یا حتی اگر در گوگل هم به جوابی نرسیدی، از هوش مصنوعی کمک‌گیری و خطا را به همراه کد خودت کپی کنی و از او راهنمایی بخواهی.

خب! ما تا اکنون یک برنامه نوشتیم که سن مخاطب را دریافت کرده و در صورتی که سن او از ۴۰ بالاتر بود پیامی را نمایش دهد و اگر سن او ۴۰ کمتر بود، پیام دیگری را نمایش دهد. حال اگر بخواهیم بنویسیم، اگر سن برابر با ۴۰ بود چه کار کنیم؟

دانشجو: احتمالاً به برنامهٔ بالا باید اضافه کنیم:

```
if age==40:
    print('Braov!')
```

اما من این برنامه رو اجرا کردم و با خطا روبرو شدم!

استاد: دقیقاً! حواستان باشد که در برنامه‌نویسی همان‌طور که قبلاً یاد گرفتیم علامت = به معنای «قرار بده» است و برای تعریف یک متغیر یا مقداردهی آن به کار می‌رود نه برای بررسی شرط مساوی بودن. برای اینکه به پایتون بفهمانیم منظورمان بررسی شرط مساوی بودن است، باید از عملگر == استفاده کنیم؛ یعنی:

```
if age==40:
    print('Braov!')
```

به عبارتی برنامه کامل ما شد:

```
age = int(input('Please enter your age:'))
if age>40:
    print('It is so great!')
if age<=40:
    print('Excellent!')
if age==40:
    print('Braov!')
```

دانشجو: اگر خواستیم شرط مثلاً $age \neq$ مخالف * (یعنی عکس مساوی) رو بنویسیم، چیکار کنیم؟

استاد: در این مواقع از عملگر != استفاده می‌کنیم مثلاً:

```
if age!=0:
    print('Braov!')
```

که به معنای این است که اگر سن مخالف صفر بود، این کار را انجام بده.

ترکیب شرط‌ها (مثال بررسی سن و جنسیت بیمار)

فرض کنید علاوه بر سن، می‌خواهیم جنسیت مخاطب را هم بپرسیم و در متغیری به نام gender ذخیره کنیم:

```
age = int(input('Please enter your age:'))
gender = input('Please enter your gender:')
```

حال می‌خواهیم شرط ما به این شکل باشد که اگر سن فرد بیشتر از ۴۰ باشد و برای مثال جنسیت فرد male باشد، یک پیام خاص نوشته شود. می‌توانیم دو شرط را باهم ترکیب کنیم. در این مثال چون می‌خواهیم سن فرد بیشتر از ۴۰ بوده و جنسیت فرد male باشد، می‌توانیم از and (به معنای و) استفاده کنیم یعنی:

```
if age>40 and gender=='male':
    print('It is so great!')
```

(حواسمان هست که منظورمان کلمه 'male' است نه متغیر male به خاطر همین در بین دو علامت ' قرار دادیم). همان‌طور که احتمالاً می‌دانید، زمانی شرط and یا همان «و» درست است که هر دو شرط درست باشند. اگر بخواهیم یکی از شرط‌ها، به عبارتی شرط اول یا شرط دوم برقرار باشد و دستور ما اجرا شود، از عملگر or به معنای «یا» استفاده می‌کنیم:

```
if age>40 or gender=='male':
    print('It is so great!')
```

که در این صورت چه شرط اول برقرار باشد، چه شرط دوم و چه هر دو، دستورات ما اجرا می‌شوند. حال به نظرتان دستور زیر کار می‌کند؟

```
if age>40 and not gender=='male':
    print('It is so great!')
```

دانشجو: منطقاً not به معنای «نه» هست، یعنی شرط «جنسیت برابر با مرد» برقرار نباشه. پس اگر سن بالای ۴۰ باشد و جنسیت مرد نباشه، دستور مدنظر ما اجرا می‌شه. استاد: بله دقیقاً! پس فهمیدیم با and، or و not که به آن‌ها عملگرهای منطقی بولین (Boolean) گفته می‌شود، می‌توانیم شرط‌ها را باهم ترکیب کنیم. حالا یک سؤال دیگر! به نظرتان کد زیر چه کار می‌کند؟

```
name = int(input('Please enter your name:'))
age = int(input('Please enter your gender:'))
gender = input('Please enter your gender:')
if age>40:
    if gender=='male':
        print('hello Mr.', name, 'older than 40')

    if gender=='female':
        print('hello Ms.', name, 'older than 40')
```

دانشجو: به نظرم سه خط اول برنامه به ترتیب نام فرد و جنسیت رو از مخاطب می‌پرسه. بعدش در خط چهارم یعنی:

```
if age>40:
```

فرض می‌کنیم سن فرد بالای ۴۰ سال باشه؛ پس شرط برقرار بوده و وارد دستورات زیرمجموعهٔ if میشیم؛ در خط پنجم یعنی:

```
if gender=='male'
```

بررسی میشه که آیا جنسیت بیمار، مرد هست یا نه. فرض می‌کنیم جنسیت مخاطب خانم باشه، پس دستورات زیر این شرط اجرا نشده و شرط بعدی یعنی:

```
if gender=='female':
```

بررسی میشه که چون جنسیت رو خانم وارد کردیم، این شرط برقرار بوده و در نتیجه

دستور زیر مجموعه اون یعنی:

```
print('hello Ms.', name, 'older than 40')
```

اجرا میشه. مثلاً اگه نام رو Leila وارد کرده بودیم، داریم:

```
hello Ms. Leila older than 40
```

استاد: حال اگر مخاطب سن را برای مثال ۳۵ وارد کرد، چه اتفاقی می افتد؟

دانشجو: به نظرم در این صورت چون شرط

```
if age>40:
```

برقرار نیست، کامپیوتر تمامی این خطوط زیرمجموعه رو رد می کنه تا به خطوط بعدی برسه که البته اینجا چون ادامه ای نداریم، برنامه ما تموم می شه.

استاد: امتحان بکنید، ببینید آیا در عمل هم همین جواب را می دهد؟

دانشجو: آره استاد! برنامه رو اجرا کردم و سن رو ۳۵ وارد کردم. هیچ چیزی روی صفحه چاپ نشد. خیلی جالبه که انگار دارم منطق برنامه نویسی و کامپیوتر رو آروم آروم درک می کنم.

استاد: درود بر تو!

مثال توصیه دارو با فشار و قند خون بیمار!

برویم سراغ یک مثال علوم پزشکی. فرض کنید برنامه ای نوشتیم که قندخون و فشارخون مخاطب را پرسیده است و سپس برای مثال: اگر قند خون بالاتر از 140 بود و فشارخون بالای ۱۶ بود، «داروی ۱» را توصیه کند، اگر قند خون بین ۱۲۰ و ۱۴۰ بود و فشارخون کمتر از ۱۴ بود، «داروی ۲» را توصیه کند و غیر این صورت (یعنی در صورتی که هیچ کدام از حالات بالا برقرار نبود) «داروی ۳» را توصیه کند.

دانشجو: منطقاً اول از همه باید قندخون و فشارخون رو از مخاطب بگیریم.

استاد: اگر موافق باشید فعلاً این دو متغیر را خودمان دستی تعریف کنیم و وقتی برنامه را تا آخر نوشتیم و از درست کار کردن آن اطمینان حاصل کردیم، آنگاه این خطوط از برنامه را به گونه‌ای تغییر دهیم که این دو مقدار را از مخاطب دریافت کند:

```
sugar = 160
```

```
pressure = 15
```

متغیر sugar را برای قندخون و متغیر pressure را برای فشارخون تعریف کردیم.

دانشجو: برای قسمت اول برنامه که راحت می‌تونیم بنویسیم:

```
if sugar>140 and pressure>15:
```

```
    print('drug 1')
```

چقدر من خفتم!! برای قسمت بعد هم می‌تونیم بنویسیم:

```
if 120<sugar<140 and pressure<14:
```

```
    print('drug 2')
```

ولی اینجاش گیر کردم! چطوری بنویسم در غیر این صورت؟

استاد: برای این مورد خیلی راحت می‌تونیم از لغت غیر این صورت (else) استفاده

کرده و بنویسیم:

```
if 120<sugar<140 and pressure<14:
```

```
    print('drug 2')
```

```
else:
```

```
    print('drug3')
```

پس تا اینجا برنامه کامل ما شد:

```
sugar = 160
```

```
pressure = 16
```



```
if sugar>140 and pressure>15:  
    print('drug 1')
```

```
if 120<sugar<140 and pressure<14:  
    print('drug 2')  
else:  
    print('drug3')
```

راستی، تا یادم نرفته است باید بگویم در خیلی از زبان‌های برنامه‌نویسی نمی‌شود نوشت:

```
120<sugar<140
```

و باید مثلاً نوشت:

```
120<sugar and sugar<140
```

که خوشبختانه زبان برنامه‌نویسی پایتون به این چیزها گیر نمی‌دهد و می‌توان مستقیماً نوشت:

```
120<sugar<140
```

حال اگر موافق باشی، برویم برنامه را باهم ردگیری (trace) کنیم. فرض کن قند خون فرد ۱۶۰ بوده و فشارخون او ۱۶ باشد؛ چه اتفاقی می‌افتد؟

دانشجو: بریم! در این صورت کامپیوتر به این خط می‌رسه:

```
if sugar>140 and pressure>15:  
    print('drug 1')
```

و چون هم قندخون بالای ۱۴۰ و هم فشارخون بالای ۱۵ هست، به عبارتی هردوی شرط‌ها برقراره؛ پس دستور زیرمجموعهٔ if اجرا میشه و روی صفحه چاپ میشه:

drug 1

و این یعنی برنامه ما درست کار می‌کنه!!! یوهووو!!

استاد: نظرت چیست که برنامه را اجرا کنی و نتیجه را ببینی؟

دانشجو: استاد! معلومه که این شاهکار برنامه‌نویسی من باید درست کار کنه! باشه

پس دو متغیر رو در خطوط اول برنامه عوض می‌کنم:

```
sugar = 160
```

```
pressure = 15
```

و برنامه رو اجرا می‌کنم. به‌به طبق انتظارم نوشت 1 drug. شما هم الکی گیر

می‌دی‌نا! ولی یه لحظه صبر کنید! بعدش نوشته 3 drug!!! یعنی:

```
drug1
```

```
drug3
```

آخه چرا؟؟؟؟

استاد: هیچ‌وقت زود به برنامه غره نشو! نظرت چیست ردگیری برنامه را از همان

جایی که متوقف شدی، ادامه بدهیم؟

دانشجو: حق با شماست استاد! دیدیم که شرط اول رو برنامه اجرا کرد و تا اینجا

برنامه درست کار کرد. بعدش برنامه وارد این خط میشه:

```
if 120<sugar<140 and pressure<14:
```

```
    print('drug 2')
```

```
else:
```

```
    print('drug3')
```

قاعدتاً چون قندخون بین ۱۲۰ و ۱۴۰ نیست و چون یکی از این شرطها برقرار

نیست، پس 2 drug روی صفحه چاپ نمیشه؛ اما نکته فکر کنم همین جاست. این

else مربوط به همین شرط دومه! یعنی در غیر این صورت، خُب راست هم می‌گه

کامپیوتر بیچاره! چون این شرط برقرار نبوده، رفته `else` رو اجرا کرده و چاپ کرده

!!drug3

```
if sugar>140 and pressure>15:  
    print('drug 1')
```

```
if 120<sugar<140 and pressure<14:  
    print('drug 2')  
else:  
    print('drug3')
```

درحالی که منظور ما این بوده که `else` مربوط به همه دو شرط بالا باشد نه فقط شرط دوم!

استاد: درود بر تو! از تفکر اکتشافی ات بسیار لذت بردم! حال چگونه می توانیم شرط دوم را هم به شرط قبلی متصل کنیم؟

در اینجا موارد می توانیم به جای `if` در شرط دوم، از `elif` استفاده کنیم. `elif` مخفف `else if` است، یعنی در غیر این صورت اگر؛ یعنی:

```
if sugar>140 and pressure>15:  
    print('drug 1')  
elif 120<sugar<140 and pressure<14:  
    print('drug 2')  
else:  
    print('drug3')
```

بیا یاد برنامه رازدگیری کنیم. فرض کنید قندخون همان ۱۶۰ و فشارخون فرد ۱۶ است. کامپیوتر ابتدا به شرط اول می رسد یعنی:

```
if sugar>140 and pressure>15:
```

```
print('drug 1')
```

چون قندخون بالای ۱۴۰ و فشارخون هم بالای ۱۵ است، پس دستور مربوطه اجرا می‌شود و روی صفحه چاپ می‌شود drug1. حالا نکته جالب این است که چون این شرط برقرار بود، دیگر کامپیوتر خطوط بعد را اجرا نمی‌کند، زیرا نوشته‌ایم elif، یعنی در غیر این صورت اگر. وقتی شرط قبلی برقرار بوده است پس دیگر نیازی به اجرای خطوط بعد نیست و کامپیوتر از روی تمامی این‌ها می‌پرد، یعنی:

```
if sugar>140 and pressure>15:
    print('drug 1')
elif 120<sugar<140 and pressure<14:
    print('drug 2')
else:
    print('drug3')
```

دانشجو: آهان؛ پس هر وقت شرطایی داشتیم که به جورایی به هم وصل هستن، بهتره به جای if های جداگانه از elif استفاده کنیم تا با خطاهای مشابه روبرو نشیم. استاد: دقیقاً؛ استفاده از elif به جای if در موارد مشابه، یک مزیت دیگر هم دارد. به کامپیوتر می‌فهماند که وقتی if اول چک شده و اجرا می‌شود، دیگر بقیه if ها نیازی به اجرا نداشته و کامپیوتر از تمامی آن‌ها صرف‌نظر می‌کند و به همین دلیل کامپیوتر بی‌جهت معطل نمی‌شود. همچنین elif کمک می‌کند که مخاطب بداند «آهان، پس این شرط‌ها به گونه‌ای به هم متصل هستند و از هم جدا نیستند» و به عبارتی خوانایی برنامه را بیشتر می‌کند.

دانشجو: استاد من یکم گیج شدم که چه زمانی از elif استفاده کنیم و چه زمانی استفاده نکنیم!

استاد: نگران نباش! تو اول از if استفاده کن. در طی روند برنامه نویسی وقتی با خطا روبرو شدی، خودت کاملاً متوجه می شوی که اینجا شرط های دوم و ... را با elif بنویسی یا ننویسی.

حالا حالتی دیگر را در نظر بگیریم. مثلاً فرض کنیم که قند خون ۱۳۰ و فشار خون فرد ۱۲ باشد. ابتدا کامپیوتر به این if می رسد:

```
if sugar>140 and pressure>15:  
    print('drug 1')
```

چون این شرط برقرار نیست، پس کامپیوتر دستورات را اجرا نکرده و وارد شرط بعدی می شود:

```
elif 120<sugar<140 and pressure<14:  
    print('drug 2')
```

چون هردوی این شرط ها برقرار است، پس دستورات زیر مجموعه اجرا می شود؛ یعنی چاپ می شود drug 2. چون این شرط برقرار بوده، پس در نتیجه else آخر اجرانمی شود:

```
else:  
    print('drug3')
```

به صورت کلی ساختار نوشتن شرط در پایتون به این شکل است:

شرط if:

دستورات

شرط بعدی elif:

دستورات

شرط بعدی **elif**:

دستورات

.
.
.

else:

دستورات در صورت برقرار نبودن هیچ کدام از شرطهای بالا

در کل بچه‌ها نگران نباشید! ساختار کلی و ساده را یاد بگیرید و در طول زمان مسلط می‌شوید! اگر جایی هم نیازی بود یا خطایی داشتید جستجو و همچنین هوش مصنوعی می‌توانند به شما کمک کنند تا به مرور زمان درک تان و تسلط تان بیشتر شود.

تمرین‌های جلسه دوم

در تمرین هر درس ممکن است که یک نکته جدید هم آموزش داده شود، پس حتماً توصیه می‌کنم آن‌ها را ببینید و انجامشان دهید. تمرین‌ها و به خصوص خطاهایی که دریافت می‌کنید و آزمون و خطاها (و همچنین سرچ‌هایی که در صورت نیاز انجام می‌دهید)، بسیار بسیار به روند یادگیری برنامه نویسی شما کمک می‌کند. پاسخ نمونه برای تمرین‌ها نیز پس از آن‌ها قرار داده شده است.

تمرین‌ها

۱- برنامه‌ای بنویسید که میزان چربی LDL خون و سابقه ابتلا به بیماری قلبی (مثبت یا منفی) را پرسیده و مطابق راهنمای زیر دارو پیشنهاد دهد:
اگر میزان چربی بیشتر از ۱۰۰ میلی گرم بر دسی لیتر باشد و سابقه بیماری مثبت باشد، "داروی ۱" را پیشنهاد کند و اگر میزان چربی بین ۸۰ تا ۱۰۰ باشد و سابقه بیماری منفی باشد "داروی ۲" را پیشنهاد کند و اگر غیر از این بود "داروی ۳" را پیشنهاد کند.

۲- برنامه‌ای بنویسید که از مخاطب بپرسد که چه کاری می‌خواهد انجام دهد. اگر مخاطب عدد ۱ را وارد کرد، از او تعداد کتابی که خوانده و سن او را پرسیده و میانگین کتابی که در هر سال خوانده را حساب کند و اگر مخاطب عدد ۲ را وارد کرد، از او سال تولد او و سال کنونی را پرسیده و سن او را با توجه به سال حاضر حساب کند. در غیر این صورت یک پیغام خطا به کاربر نشان دهد که عدد درستی را وارد نکرده است.

۳- در رشته یا حرفه خود، یک تمرین مشابه سؤال بالا، منتها کاربردی، طراحی و اجرا کنید.

پاسخ تمرین‌های جلسهٔ دوم

اکیداً توصیه می‌کنم که اول خودتان تمرین‌ها را حل کنید و سپس به پاسخ‌ها مراجعه کنید. در ضمن این پاسخ‌ها تنها یکی از حالت‌های ممکن نوشتن برنامه هستند و شما ممکن است به شیوه‌ای دیگر برنامه را نوشته باشید و کاملاً صحیح هم باشد. همچنین برای مشاهدهٔ توضیحات شفاهی مربوط به این تمرین‌ها، ابتدای ویدئوی جلسهٔ بعد را ببینید (لینک ویدئو در صفحهٔ اول کتاب گذاشته شده است):

« پاسخ تمرین اول:

```
LDL = float(input('please enter your LDL level(mg/dl):'))
heart_history = input('Have you ever been infected with
Heart disease? enter Positive or Negative:')
if LDL > 100 and heart_history == 'Positive':
    print('please use drug 1.')
elif 80 < LDL < 100 and heart_history == 'Negative':
    print('please use drug 2.')
else:
    print('please use drug 3.')
```

چون میزان LDL خون می‌تواند یک عدد اعشاری باشد، بهتر است نوع متغیر را به float تغییر دهیم که اینجا نیز همین کار انجام شده است.

« پاسخ تمرین دوم:

```
choice = input('Hello, what do you want to do? Enter
number 1 or 2:')
if choice == '1':
    book = int(input('How many books do you read ever?:'))
    age = int(input('How old are you?:'))
```



```
mean = round(book/age, 2)
print('hello the mean of books you read is:', mean)
elif choice == '2':
    birth_year = int(input('please say "What year
were you born?":'))
    current_year = int(input('please say "what is
the current year?":'))
    age = current_year-birth_year
    print('hello, your age is:', age)
else:
    print('You have not entered the correct number.')
```

نکته تکمیلی:

برنامه نوشته شده را به جای spyder یا idle، در محیط cmd اجرا کنید. چگونه؟ کافی است محیط cmd را باز کرده و بنویسید:

نام برنامه نوشته شده python

مثلاً در اینجا اگر اسم برنامه book.py باشد می توان نوشت:

python book.py

یا به جای اسم برنامه می توان آدرس آن را نوشت. این روش را بلد باشید؛ بعداً به

کارتان می آید. 😊

« پاسخ تمرین سوم (یک مثال):

```
print('welcome to simple statistical significancy check')
p_value = float(input('please enter your p-value of the
test:'))
if 0<= p_value < 0.05 :
```

```
print('is statistical significant')
elif p_value < 0:
    print('The p-value is not valid')
else:
    print('is not significant')
```

این برنامه، با این هدف طراحی شده است که نتیجه پارامتر p_value آزمون آماری دریافت از مخاطب دریافت شود. همان‌طور که احتمالاً بدانید، معمولاً وقتی p_value آزمون مربوطه از ۰.۰۵ کمتر است، آن تفاوت از نظر آماری قابل توجه (significant) ارزیابی می‌شود.

• فصل سوم •

حلقه‌ها و لذت انجام کارهای تکراری و خسته‌کننده!

حلقه‌ها به چه دردی می‌خورند؟

درود بر شما! تا اکنون با دو شگفتی آشنا شدیم! شگفتی اول که دیدیم با متغیرها می‌توانیم ظرف ایجاد کنیم و مقادیر را در حافظه کامپیوتر نگه داریم. شگفتی دوم که دیدیم می‌توانیم با `if` و شرط‌ها، قدرت تصمیم‌گیری به کامپیوتر بدهیم و متوجه شدیم که چگونه به کامپیوتر بگوییم: «در این صورت این کار را انجام بده و در غیر این صورت فلان کار را انجام نده». امروز می‌خواهیم با شگفتی سوم یعنی حلقه‌ها آشنا شویم.

برای این که درک کنیم حلقه چیست، با یک مثال شروع می‌کنم. فرض کنید می‌خواهیم برنامه‌ای بنویسیم که ۱۰ بار سلام کند:

```
print('hello')
```

با اجرای این خط، کامپیوتر یک‌بار سلام می‌کند، پس با چیزی که تاکنون بلد هستیم، اگر بخواهیم کامپیوتر ۱۰ بار این کار انجام دهد، باید ۱۰ بار این خط را بنویسیم؛ یعنی:

```
print('hello')
```

```
print('hello')
print('hello')
print('hello')
print('hello')
print('hello')
print('hello')
print('hello')
print('hello')
```

همه را نوشتیم تا ببینید که چقدر این کار وقت گیر است. حالا فرض کنید بخواهیم ۱۰۰ بار این کار انجام شود. آن موقع چه کار کنیم؟

دانشجو: قاعدتاً پیر میشیم که بخوایم ۱۰۰ بار این خط رو بنویسیم! البته به نظرم بتونیم کپی و پیست کنیم!

استاد: ممکن است شما بخواهید کپی و پیست کنید، اما اگر بخواهیم این کار را انجام دهیم که کنار هر بار سلام، اسم فرد را هم بنویسید، مثلاً:

```
print('Hello', 'Pedram')
print('Hello', 'Leila')
print('Hello', 'Mohammad')
print('Hello', 'Ali')
```

دیگر با کپی و پیست و از اینجور دستورها هم نمی شود به راحتی این کار را کرد!

دانشجو: پس باید چیکار کرد؟

استاد: راهکار بسیار عالی، استفاده از حلقه است. این که به کامپیوتر بگوییم به تعداد مدنظر ما این دستورات را انجام بدهد. چگونه؟ با چند مثال، مبحث حلقه برایمان جا می افتد. فرض کنید بخواهیم همان مثال بالا، یعنی سلام و به همراه نام فرد را انجام

دهیم.

کافی است بنویسیم `for` و یک متغیر را بعد از آن بنویسیم. برای مثال، چون با نام افراد سروکار داریم، به نظر می‌آید متغیر `name` مناسب باشد:

```
for name
```

و در ادامه در قالب یک لیست، اسم افراد را بنویسیم؛ یعنی:

```
for name in ['Pedram', 'Leila', 'Mohammad', 'Ali']:
```

و با یک دندانده دستوراتی که می‌خواهیم تکرار شود را بنویسیم؛ یعنی:

```
for name in ['Pedram', 'Leila', 'Mohammad', 'Ali']:  
    print('Hello', name)
```

دانشجو: استاد من درست نفهمیدم چی شد!

استاد: طبیعی است! باهم برنامه را ردگیری می‌کنیم و به‌خوبی حلقه‌ها برای شما جا می‌افتد. الآن باهم خط به خط برنامه را ردگیری می‌کنیم.

کامپیوتر موقع اجرای حلقه ما، وقتی به این قسمت می‌رسد:

```
for name in ['Pedram', 'Leila', 'Mohammad', 'Ali']:
```

یک متغیر به نام `name` تعریف کرده و آن را برابر با اولین مقدار این لیست، که در اینجا 'Pedram' است قرار می‌دهد.

```
for name in ['Pedram', 'Leila', 'Mohammad', 'Ali']:
```

سپس به خط بعد می‌رسد:

```
    print('Hello', name)
```

و چون الآن مقدار `name` برابر با 'Pedram' است، روی صفحه چاپ می‌شود:

Hello Pedram

حال نکتهٔ جالب فرا می‌رسد. کامپیوتر دوباره به اول حلقه یعنی این خط برمی‌گردد و حال مقدار متغیر `name` برابر با دومین عضو لیست ما می‌شود:

```
for name in ['Pedram', 'Leila', 'Mohammad', 'Ali']:
```

و چون الآن مقدار `name` برابر با 'Leila' است و در نتیجه در خط بعد:

```
print('Hello', name)
```

روی صفحه چاپ می‌شود:

```
Hello Leila
```

و مجدداً کامپیوتر دوباره به اول حلقه یعنی این خط برمی‌گردد و این حلقه آن‌قدر تکرار می‌شود که به آخرین عضو لیست می‌رسیم، یعنی 'Ali' و در آخر

```
Hello Ali
```

نیز روی صفحه چاپ می‌شود. به عبارتی کل خروجی ما که کاربر مشاهده کرده این است:

```
Hello Pedram
```

```
Hello Leila
```

```
Hello Mohammad
```

```
Hello Leila
```

دانشجو: استاد تا حد زیادی روشن‌تر شد! یه مثال دیگه می‌زنید که کامل‌تر جا بیفته؟

استاد: بله حتماً! فرض کنید می‌خواهیم برنامه‌ای بنویسیم که ۵ بار سلام کند؛ به عبارتی همچنین خروجی‌ای برای ما روی صفحه چاپ کند:

```
Hello 1
```

```
Hello 2
```

```
Hello 3
```

```
Hello 4
```

Hello 5

دیدیم که اگر بخواهیم با همان روشی که قبلاً بلد بودیم، یعنی بدون استفاده از حلقه‌ها، این کار را انجام دهیم، باید همچنین برنامه‌ای را می‌نوشتیم:

```
print('Hello', 1)
print('Hello', 2)
print('Hello', 3)
print('Hello', 4)
print('Hello', 5)
```

حال می‌خواهیم این کار خسته‌کننده را انجام ندهیم. چون داریم یک کار را تکرار می‌کنیم، نیاز به حلقه داریم و الآن یک حلقه بلد هستیم به نام `for`:

`for`

یادمان است که بعد از `for` نیاز به یک متغیر داریم. چه متغیری برای اینجا بگذاریم که بامسما باشد؟ چون می‌خواهیم عدد ما در حلقه عوض شود، می‌توانیم از متغیری برای مثال به نام `num` استفاده کنیم:

`for num`

البته می‌توانیم از هر متغیری که دلمان خواست استفاده کنیم. حتی متغیرهای تک حرفی برای مثال `i`؛ اما توصیه می‌کنم تا جایی که می‌توانید از نام‌های بامعنی برای حلقه‌ها (و کلاً در برنامه‌نویسی خودتان) استفاده کنید تا بعداً راحت‌تر بتوانید منظور برنامه خودتان را متوجه شوید.

حال بعد از `for num` نیاز داریم یک لیست از اعداد داشته باشیم (در مثال قبل، یک لیست از رشته‌ها داشتیم که اسامی اشخاص بودند) که در هر بار دور زدن حلقه، متغیر ما یکی از آن‌ها شود؛ به عبارتی یک لیست داشته باشیم که روی آن حرکت کنیم:

`for num in [1, 2, 3, 4, 5]:`

و در نهایت دستورات مدنظرمان را با یک دندانۀ زیر حلقه بنویسیم:

```
for num in [1, 2, 3, 4, 5]:  
    print('Hello', num)
```

اگر موافق باشید، برنامه‌ای که نوشتیم را با یکدیگر ردگیری کنیم (trace کنیم) و ببینیم که کامپیوتر وقتی با برنامه ما روبرو می‌شود، خط به خط چه می‌کند. ابتدا وقتی که کامپیوتر با:

```
for num
```

روبرو می‌شود، یک متغیر به نام `num` تعریف کرده و مقدار آن را برابر با اولین عضو لیستی که دادیم، قرار می‌دهید؛ که اینجا اولین عضو لیست ما چیست؟

دانشجو: اولین عضو ما، عدد ۱ هست که اون رو در متن زیر با رنگ زرد هایلایت کردم:

```
for num in [1, 2, 3, 4, 5]:
```

پس الان قاعدتاً متغیر `num` باید مقدار ۱ رو داشته باشه.

استاد: درود بر تو! می‌توانی بگویی که در خط بعد، کامپیوتر چه می‌کند؟

دانشجو: در خط بعد این دستور رو داریم:

```
print('Hello', num)
```

و چون مقدار `num` برابر با ۱ هست، پس روی صفحه چاپ میشه:

Hello 1

استاد: و سپس کامپیوتر چه می‌کند؟

دانشجو: اگر درست متوجه شده باشم، کامپیوتر به اول حلقه برمی‌گرده و حالا مقدار

`num` برابر با دومین عضو لیست یعنی عدد ۲ میشه:

```
for num in [1, 2, 3, 4, 5]:
```

و دوباره در خط بعد:

```
print('Hello', num)
```

چون مقدار num برابر با ۲ هست، روی صفحه چاپ میشه:

Hello 2

و مجدداً کامپیوتر به اول حلقه برمی‌گرده و این روند تکرار می‌شه.

استاد: بسیار عالی! و این کار تا کی انجام می‌شود؟

دانشجو: هوممممم... احتمالاً وقتی num برابر با آخرین عضو لیست یعنی عدد ۵

میشه:

```
for num in [1, 2, 3, 4, 5]:
```

و دستور بعدی یعنی:

```
print('Hello', num)
```

اجرا میشه؛ چون به آخر لیست رسیدیم، دیگه کامپیوتر به ابتدای حلقه برنمی‌گرده و ادامه خطوط برنامه که بعد از حلقه‌اس اجرا میشه.

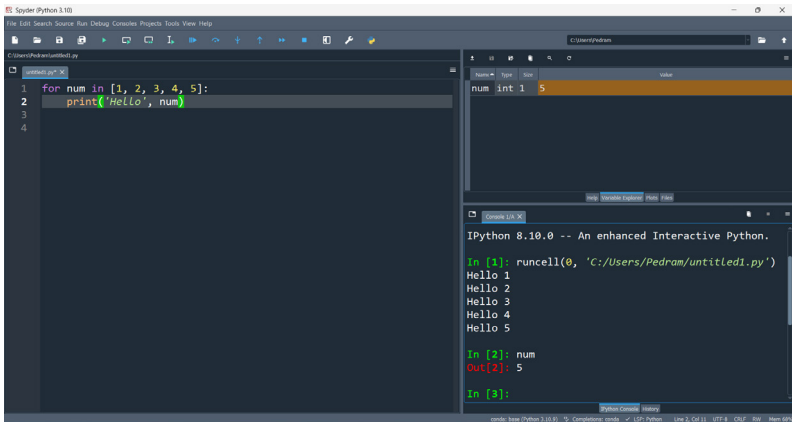
استاد: پس بعد از این که حلقه تمام شد، به نظر تو مقدار متغیر num برابر چند است؟

دانشجو: ... فکر کنم ۵؛ چون آخرین بار اجرای حلقه که ۵ بود...

استاد: نظرت چیست که بعد از اجرای برنامه که هنوز متغیرها در حافظه کامپیوتر هستند، در قسمت کنسول که پایین سمت راست وجود دارد، بنویسی num و ببینی مقدار num برابر چند است؟

دانشجو: چه جالب این جوریه هم می‌شه؟ الان تست می‌کنم! نوشتم num و دیدم

برابر ۵ هست. چه جالب (شکل ۱)!!!:



شکل ۱- می‌توان با کنسول که در پایین سمت راست برنامه قرار دارد، متغیرها را چک کرد.

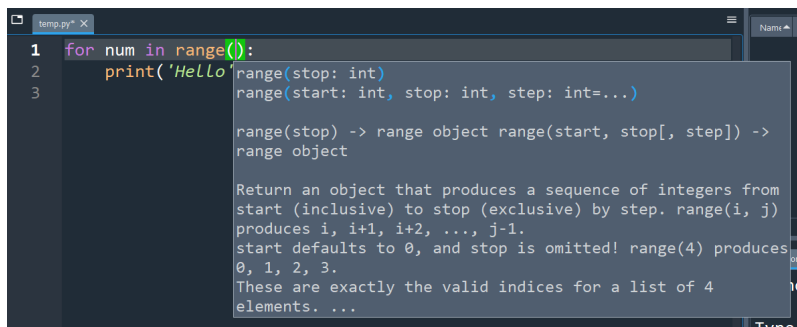
استاد: درود بر تو! همچنین در قسمت variable explorer در قسمت بالا سمت راست هم می‌توانیم آخرین مقدار متغیر num را ببینیم (شکل ۱). این روش در خیلی از مواقع که در طی اجرای حلقه به مشکل می‌خوریم و برنامه خطا داده و متوقف می‌شود، بسیار به کار می‌آید. به این شکل می‌توانیم در کنسول ببینیم مقدار متغیر مثلاً num برابر چند است و به عبارتی حلقه در کدام عضو لیست ما دچار خطا و متوقف شده است.

دانشجو: ممنون استاد! حالا سؤالی که برای من پیش آمده اینه که مگه ما از حلقه زمانی استفاده نمی‌کنیم که بخواهیم یک کار رو به تعداد زیاد تکرار کنیم؟ فرض کنید بخواهیم برنامه‌ای بنویسیم که به جای ۵ بار، ۱۰۰۰ بار سلام کنه. اونوقت باید از ۱ تا ۱۰۰۰ رو در لیستم بنویسم؟ این جوری که دوباره سخت میشه!

استاد: سؤال بسیار خوبی پرسیدی. در این مواقع می‌توانیم کاری کنیم که خود پایتون این لیست را ایجاد کند و مثلاً نخواهیم از ۱ تا ۱۰ را به این شکل بنویسیم:

```
for num in [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]:
    print('Hello')
```

برای این کار کافی است از دستور range استفاده کنیم (شکل ۲):



شکل ۲- خود اسپایدر یا محیط کدنویسی ما را راهنمایی می‌کند.

همان‌طور که می‌بینیم پایتون ما را راهنمایی کرد که می‌توانیم به دو شکل از range استفاده کنیم؛ یک حالت این است که شروع و پایان را بنویسیم (و می‌بینید که نوشته شده int، یعنی باید عدد صحیح وارد شود) مثلاً:

```
for num in range(1, 10):
    print('Hello', num)
```

این را اجرا کنید ببینید و چه چیزی را خروجی می‌گیرید؟

دانشجو: چه باحال! خودش همون کار برنامه‌ای که با دست انجام داده بودیم رو انجام داد. ولی استاد ۱۰ رو پرینت نکرد؛ یعنی همچنین خروجی‌ای گرفتیم:

```
Hello 1
Hello 2
Hello 3
Hello 4
```

```
Hello 5
Hello 6
Hello 7
Hello 8
Hello 9
```

استاد: پایتون با دستور

```
range(1, 10)
```

یک آرایه از عدد ۱ تا یکی قبل از ۱۰ (یعنی ۹) را می سازد. پس اگر بخواهیم خود ۱۰ را هم در لیست قرار بدهد باید چه کنیم؟

دانشجو: احتمالاً بشه از دستور زیر استفاده کرد؛ یعنی بنویسیم از ۱ تا ۱۱:

```
range(1, 11)
```

استاد: دقیقاً! از range به گونه ای دیگر هم می توان استفاده کرد. اگر تنها یک عدد

یعنی:

```
for num in range(11):
    print('Hello', num)
```

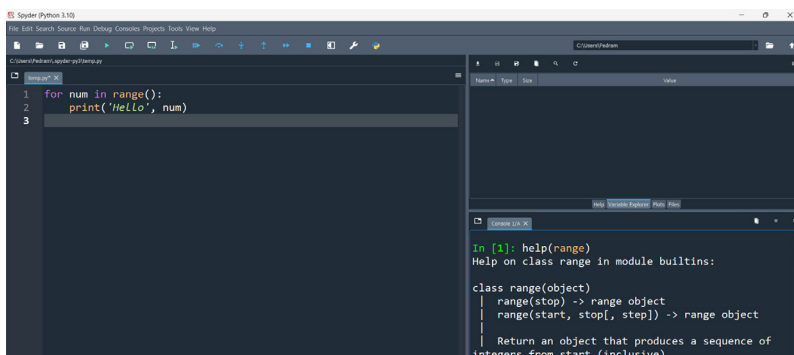
چه خروجی ای دریافت می کنیم؟

دانشجو: من امتحان کردم استاد. از عدد صفر شروع کرد تا یکی قبل از ۱۱؛ یعنی تا عدد ۱۰.

استاد: پس متوجه می شویم همان طور که در help پایتون هم دیدیم، اگر فقط یک عدد بنویسیم، پایتون از صفر شروع می کند تا یکی قبل از عدد نوشته شده. راستی اگر در کنسول بنویسید help() و در داخل پرانتز دستور مورد نظر خود را بنویسید پایتون توضیحات آن دستور را برایتان می نویسد؛ مثلاً اگر بنویسیم:

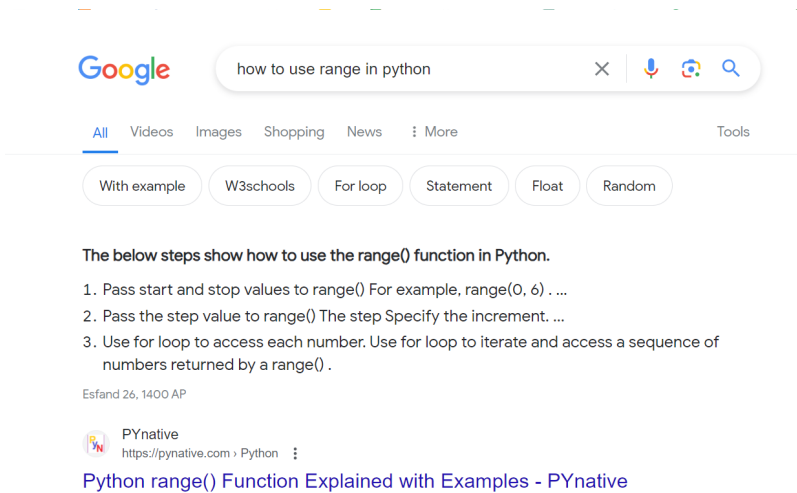
help(range)

می‌بینیم که اطلاعات آن دستور برایمان نمایش داده می‌شود (شکل ۳):



شکل ۳- استفاده از دستور help

و صدا البته اگر باز با help هم برایمان سخت بود، جستجو در گوگل را فراموش نکنید که بسیار عالی و با مثال، کد موضوع را برایتان جا می‌اندازد؛ مثلاً من جستجو کردم `how to use range in python` و می‌بینید که چه لینک‌های خوبی برای من آورده شده که کافی است یکی یا دو تا را باز کنم و ببینم که چه سایتی بهتر توضیح داده است و آن را به همراه مثال عملی آن کمی مطالعه کنم (شکل ۴):



شکل ۴- گوگل دوست خوبی است مخصوصاً به زبان انگلیسی!

و در نهایت اگر باز متوجه نشدید، می توانید از هوش مصنوعی (مثلاً بینگ یا gemeni گوگل) بپرسید و باز اگر متوجه نشدید با آن بحث کنید که کامل موضوع را شيرفهم می کند 😊!

خب برویم یک شوخی با پایتون انجام دهیم! اگر بخواهیم برنامه ای بنویسیم که ۱۰۰۰ بار سلام کند، چه کنیم؟

دانشجو: طبیعتاً چون داریم یک کار رو به تعداد زیاد تکرار می کنیم، بهتره از حلقه استفاده کنیم. تا جایی که خاطرمه باید از for و یه متغیر استفاده کنیم که من مثلاً از num استفاده می کنم:

```
for num in range(1, 1000):  
    print('Hello', num)
```

استاد: نظرت چیست که برنامه را اجرا کنیم؟

دانشجو: بله حتماً استاد! منتهای... تا ۹۹۹ نوشت! چرا!! آهان یادم رفته بود که پایتون تا یکی مونده به آخر حد پایانی range می‌ره! پس باید می‌نوشتیم:

```
for num in range(1, 1001):
    print('Hello', num)
```

استاد: بسیار عالی. به نظرت نیازی بود که در دستور print حتماً متغیر num را هم پرینت کنیم؟ مگر فقط نمی‌خواستیم که کامپیوتر ۱۰۰۰ بار سلام کند؟
دانشجو: درسته استاد! یعنی مثلاً بنویسیم:

```
for num in range(1, 1001):
    print('Hello')
```

استاد: بسیار خوب! یا نظرت چیست برای این که کدمان خلوت‌تر شود بنویسیم:
for num in range(1000):
 print('Hello')

دانشجو: استاد این که اشتباه می‌شه! البته نه... انگار درسته؛ چون پایتون از صفر شروع می‌کنه و تا ۹۹۹ پیش می‌ره، یعنی سرجمع انگار صدبار سلام کرده! چه جالب! پس من این جوری که فهمیدم نیازی نیست حتماً از متغیری که در حلقه تعریف می‌شه، استفاده کنیم.

استاد: بله دقیقاً. در این جور مواقع که از این متغیر فقط به‌عنوان یک شمارنده استفاده می‌شود و کاربرد دیگری در حلقه ندارد، معمولاً از نام‌هایی که مشخص است استفاده می‌شود مثلاً i:

```
for i in range(1000):
    print('Hello')
```

یا حتی از نام‌هایی که مشخص شود هیچ اهمیتی ندارد، مثلاً متغیری به نام _:

```
for _ in range(1000):  
    print('Hello')
```

دانشجو: متغیر به اسم - یا آندرلاین تا حالا ندیده بودم!

استاد: البته این کار برای این است که مخاطب بفهمد که این متغیر، متغیر مهمی نیست و ذهن خواننده کد درگیر نشود که این متغیر قرار است بعداً استفاده شود.

دانشجو: چه خفن! پس سعی کنم کدهام رو تمیزتر بنویسم!

استاد: دقیقاً! برنامه نویسی خوب برنامه نویسی نیست که الزاماً بتواند فقط برنامه را بنویسد، بلکه برنامه نویسی است که کد خودش را تمیزتر بنویسد تا آیندگان راحت تر بتوانند منظور او از کد را متوجه شوند و اگر خواستند از آن استفاده کرده یا آن را ارتقا دهند، راحت تر بتوانند این کار را انجام دهند. از طرفی خود شما هم وقتی بعداً خواستید کدهای قبلی خودتان را بخوانید، به این شکل راحت تر متوجه می شوید که قبلاً چه کرده اید و در وقت و انرژی شما بسیار صرفه جویی می شود.

دانشجو: استاد به دادم برس!!! من گفتم یه کم با کامپیوتر کیف کنم و اذیتش کنم! نوشتم ۱۰۰۰۰۰۰ بار سلام کنه!

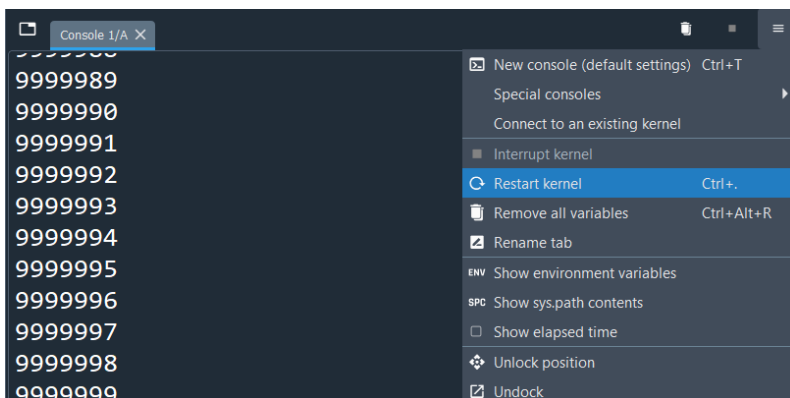
```
for num in range(1000000):  
    print('Hello', num)
```

حالا کامپیوتر داره مدام چاپ می کنه Hello و من نمی دونم چیکار کنم قطع بشه! استاد: یه نفس عمیق بکش و خونسردی ات را حفظ کن! دو راه وجود داره! یا این که خیلی راحت روی کیبورد، دکمه ترکیبی:

Cntrl + C

رافشار بدهی یا این که علامت مربع قرمز که بالای کنسول است را بزنی که اصطلاحاً

برنامه `interrupt` و متوقف شود. اگر پیدایش نکردی هم کافیست روی علامت چندخط بالای سمت راست کنسول کلیک کنی و گزینه `interrupt` را بزنی. در بسیاری از مواقع به‌خوبی کار می‌کند، اما اگر دیدی بازهم نشد (که البته خیلی بعید است)، روی همان خطوط بزن و این بار گزینه `restart console` را بزن (شکل ۵).



شکل ۵- متوقف کردن دستی برنامه (همان `interrupt` کردن)

دانشجو: آخیش استاد!! خیالم راحت شد و با همون کلید `Ctrl+C` اجرای برنامه قطع شد. یه سؤال! من در `help` پایتون برای دستور `range` دیدم که یک پارامتر دیگه هم به نام `step` نوشته شده:

`range(start, stop[, step])`

و متوجه نشدم که چرا در علامت `[]` قرار داده شده و چرا ما تا الان فقط `start` و `stop` یعنی شروع و پایان `range` رو مشخص می‌کردیم.

استاد: بسیار سؤال خوبی پرسیدی! این به این معنی است که وارد کردن پارامتر `step` اجباری نیست و اگر ما وارد نکنیم، پایتون خودش به شکل پیش‌فرض آن را پر می‌کند (که در دستور `range` آن را برابر ۱ قرار می‌دهد). همان‌طور که در راهنمای

spyder هم دیدیم، یک علامت = بعد از step قرار داده شده است. این یعنی اگر ما step را وارد نکنیم، پایتون خودش به شکل پیش فرض مقداری (که در help اگر بخوانید نوشته شده ۱) را قرار می دهد.

```
range(stop: int)
range(start: int, stop: int, step: int=...)
```

شکل ۶- نحوه خواندن help پایتون

حال بیایید این کار را انجام دهیم و ببینیم چه اتفاقی می افتد:

```
for num in range(1, 10, 2):
    print(num)
```

دانشجو: من امتحان کردم، نوشت:

1
3
5
7
9

چه جالب انگار به جای یکی، یکی با گام دو یعنی دوتا دوتا رفت!

استاد: دقیقاً! و حالا این چطور؟

```
for num in range(5, 1, -1):
    print(num)
```

دانشجو: به نظر سخت شد! اما بذارید فکر کنم. طبق توضیحی که در پایتون خواندیم، ترتیب پارامترها در دستور range می شد step stop start. پس به نظرم شروع ۵، پایان ۱ و گام -۱ یعنی یکی یکی کم بشه؛ پس همچنین چیزی باید در خروجی دریافت کنیم:

5
4
3
2
1

استاد: بسیار عالی! ولی به نظرت تا چه عددی حلقه ادامه پیدا می‌کند؟

دانشجو: تا ۱ دیگه استاد... ولی نه یادمه گفتید تا یکی قبل از پایان؛ یعنی تا ۲. یعنی باید در خروجی ببینیم:

5
4
3
2

پس اگه بخوایم خود ۱ هم چاپ بشه، باید احتمالاً تا ۰ بنویسیم که حلقه تا عدد ۱ پیش بره یعنی:

```
for num in range(5, 0, -1):  
    print(num)
```

الآن درست شد!

استاد: بسیار عالی!

حال فرض کنید که بخواهیم تمام اعداد زوج کوچک‌تر از ۱۰۰۰ را پیدا کنیم. چطور می‌توانیم برنامه‌ای بنویسیم که تمام این اعداد را به ما نشان دهد؟

دانشجو: شاید بتونیم یه حلقه بنویسیم ولی با گام ۲ که از ۲ شروع شده و دوتا دوتا بالا بره؛ یعنی:

```
for num in range(2, 1000, 2):
```

```
print(num)
```

استاد: راه حل خوبی بود! اما بیاییم از روش دیگری نیز مسئله را حل کنیم. خودمان به عنوان یک انسان چه می کنیم؟ از عدد ۱ شروع می کنیم، اگر زوج بود (یعنی باقیمانده تقسیم آن بر ۲، برابر با صفر بود) آن را چاپ کنیم وگرنه آن را رها کرده و به سراغ عدد بعدی برویم.

اول از همه نیاز داریم که برنامه ما یکی یکی به اعداد دسترسی پیدا کند. برای این کار چه چیزی را می شناسیم؟ حلقه! پس تا اینجا:

```
for num in range(1, 1000):
```

به عبارتی در هر بار اجرای حلقه، num یکی از اعداد ۱ تا ۹۹۹ می شود. چگونه می توانیم به کامپیوتر بگوییم که اگر باقیمانده تقسیم بر ۲ برابر صفر بود، آن عدد را برآیمان چاپ کن؟

دانشجو: تا جایی که یادمه، برای اگرها و شرطها از if استفاده می کردیم.

استاد: بسیار عالی! چگونه می توانستیم در پایتون، باقیمانده تقسیم را حساب کنیم؟

دانشجو: فکر کنم یادمه! یه علامت عجیب داشت؛ آهان علامت % . بذارین داخل کنسول چک کنیم. مثلاً باقیمانده ۴ بر ۲ باید صفر باشه. ببینم جواب می ده:

```
2%2
```

آره! جوابش شده صفر!

استاد: و حالا چطور می توانیم آن را به صورت شرط بنویسیم؟

دانشجو: ما می خواهیم اگه باقیمانده تقسیم عدد ما بر ۲ برابر صفر بود، عدد ما چاپ بشه. پس احتمالاً:

```
for num in range(1, 1000):
    if num%2 == 0:
        print(num)
```

عجب برنامه‌نویسی خفنی هستم!! بذارید اجراش هم بکنم! ای‌وای خطای زیر رو داد!

```
In [1]: runfile('C:/Users/Pedram/.spyder-py3/
temp.py', wdir='C:/Users/Pedram/.spyder-py3')
File <unknown>:2
    if num%2 == 0:
        ^
SyntaxError: cannot assign to expression here.
Maybe you meant '==' instead of '='?
```

شکل ۷- خطای نمایش داده‌شده

استاد: به نظرت خطا دارد به ما چه می‌گوید؟

دانشجو: نوشته که احتمالاً منظور ما == به جای علامت = نبوده... آهان حواسم نبود. برای شرط مساوی، در جلسه قبل یاد گرفتیم که باید == به کار ببریم و علامت = در پایتون به معنای قرار بده است، پس:

```
for num in range(1, 1000):
    if num%2 == 0:
        print(num)
```

الآن جواب داد و تمام اعداد زوج کوچک‌تر از ۱۰۰۰ رو به من نشون داد! چه جالب!

استاد: درود بر تو! مهارت رفع خطای برنامه‌نویسی، مهارت مهمی است که به‌مرور زمان و البته با دقت به پیام‌های خطا و همچنین جستجو کردن، روزبه‌روز بهتر می‌شود. نظرت چیست که یک‌بار خودمان را جای کامپیوتر بگذاریم و برنامه را

ردگیری کنیم تا بهتر یاد بگیریم؟

دانشجو: چرا که نه! وقتی کامپیوتر به این خط می‌رسد:

```
for num in range(1, 1000):
```

یه متغیر به اسم num تعریف می‌کنه و چون عضو اول لیست ما ۱ است، مقدار num رو برابر با عدد ۱ قرار می‌ده. سپس به این خط می‌رسد:

```
if num%2 == 0:
```

و چک می‌کنه که آیا باقیمونده num (که اینجا عدد ۱ است) بر ۲، صفر است؟ باقیمونده عدد ۱ بر ۲ صفر نیست، پس دستور شرطی زیر اجرا نمی‌شه و کامپیوتر از روی اون می‌پره:

```
print(num)
```

و دوباره کامپیوتر به ابتدای حلقه برمی‌گرده:

```
for num in range(1, 1000):
```

و این بار num برابر با عدد ۲ می‌شه. حال کامپیوتر به خط بعد می‌رسد:

```
if num%2 == 0:
```

آیا باقیمانده ۲ بر ۲، صفره؟ بله! پس دستورات زیرمجموعه if یعنی این دستور اجرا می‌شه:

```
print(num)
```

و در نتیجه مقدار num یعنی عدد ۲ روی صفحه چاپ می‌شه. حالا دوباره کامپیوتر به ابتدای حلقه برمی‌گرده و این بار num برابر با عدد ۳ می‌شه و ...

استاد: و آن قدر این حلقه تکرار می‌شود که به آخرین عضو می‌رسیم و num برابر با آخرین عضو یعنی ۹۹۹ می‌شود و حلقه برای آخرین بار اجرا می‌شود. پس از آن حلقه

تمام شده و کامپیوتر از حلقه خارج می‌شود و دستورات بعد از حلقه را اجرا می‌کند (که البته برنامه ما بعد از حلقه تمام شده و خطوط دیگری برای اجرا ندارد).

حال اگر بخواهیم که کامپیوتر، به جای اعداد زوج، اعداد فرد کوچک‌تر از ۱۰۰۰ را به ما نشان دهد، باید چه کنیم؟

دانشجو: فکر می‌کنم هیچ فرقی با برنامه قبلی نداره؛ منتها عدد فرد باقیمانده‌اش بر ۲ برابر با ۱ است؛ پس به نظرم برنامه ما به این شکل درمیاره:

```
for num in range(1, 1000):
    if num%2 == 1:
        print(num)
```

استاد: دقیقاً! یا مثلاً یک روش دیگر این بود که بنویسیم باقیمانده، مخالف با صفر باشد (یادمان است که در پایتون $!=$ به معنی بررسی مخالف بودن مقدار دو چیز است و $=$ به معنای بررسی مساوی بودن مقدار دو چیز است):

```
for num in range(1, 1000):
    if num%2 != 0:
        print(num)
```

دانشجو: استاد مهمه که حتماً فاصله بذاریم؟ مثلاً اگر این جور بنویسیم مشکلی پیش میاد؟

```
for num in range(1,1000):
    if num%2!=0:
        print(num)
```

استاد: می‌توانی امتحان کنی!

دانشجو: الان امتحان کردم هیچ خطایی نداد! پس مشکلی نداره؟

استاد: پایتون در این زمینه اصلاً سخت گیر نیست و چه فاصله بگذاریم و چه نگذاریم، پایتون کد ما را اجرا می کند؛ اما اگر فاصله بگذاریم، از لحاظ زیبایی کد، اصول PEP8 که اصول زیبایی کد است را رعایت کرده ایم و کد ما خواناتر می شود.

حال به عنوان تمرین، برنامه ای بنویسید که تمام اعدادی که مضرب ۳ یا مضرب ۷ هستند را پیدا کند.

دانشجو: به نظرم همان برنامه قبلی است ولی در قسمت شرط باید تفاوت بدهیم، به نظرم:

```
for num in range(1, 1000):  
    if num%3==0 and num%7==0:  
        print(num)
```

استاد: درود بر تو؛ منتها گفتیم and یا or؟

دانشجو: آهان حواسم نبود! این برنامه من، اعدادی که هم بر ۳ و هم بر ۷ بخش پذیر باشند رو پیدا می کنه. ما می خواستیم اعدادی که یا بر ۳ یا بر ۷ یا بر هر دو بخش پذیر هستن رو پیدا کنیم؛ پس به جای and، از or استفاده می کنم:

```
for num in range(1, 1000):  
    if num%3==0 or num%7==0:  
        print(num)
```

استاد: درود بر تو! نظرت چ یست ابتدا از ۱ تا ۱۰ را چک کنیم و سپس وقتی که دیدیم برنامه درست جواب داد، بنویسم از ۱ تا ۱۰۰۰؟

دانشجو: این هم فکر خوبیه؛ پس برای این که بتونم برنامه رو چک کنم:

```
for num in range(1, 10):  
    if num%3==0 or num%7==0:  
        print(num)
```

خب برنامه رو اجرا می‌کنم. این خروجی رو به من داد:

4

7

که کاملاً درسته که اعدادی رو به من نشون داده که یا بر ۴ یا بر ۷ هر هردو، بخش پذیر هستن. پس با خیال راحت تر می‌تونم برنامه رو به همون ۱ تا ۱۰۰۰ تغییر بدم.

استاد: دقیقاً! به قول شعاری که هکرها دارند، هر برنامه پیچیده، زمانی برنامه ساده‌ای بوده که کار می‌کرده است! همیشه سعی کنید برنامه را برای هدف ساده‌تر بنویسید و بعد آن را توسعه دهید. این کار خیلی بهتر از آن است که از همان اول بخواهید یک برنامه پیچیده را به همان پیچیدگی بنویسید و به هم بپیچید و در آخر هم موضوع را بیپیچانید و بی خیال شوید!

دانشجو: کاملاً شیرفهم شدم!

استاد: حالا برویم سراغ توسعه برنامه قبلی!

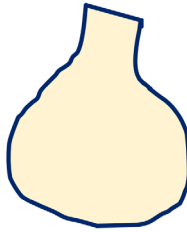
شمارش با حلقه‌ها

می‌خواهیم بدانیم چند عدد کوچک‌تر از ۱۰۰۰ وجود دارند که زوج هستند. چه کنیم؟

دانشجو: بذارین الآن می‌گم؛ نمی‌دونم!!!

استاد: وقتی نمی‌دانید که چه کار کنید، مسئله را ساده کنید؛ مثلاً برای اعداد کوچک‌تر از ۱۰، ببینید که خودمان به عنوان یک انسان چطور مرحله به مرحله این کار را انجام می‌دهیم و تعداد اعداد زوج را می‌شماریم؟

اعداد ۱ تا ۱۰ را مدنظر قرار داده و یک کیسه خالی را در نظر می‌گیریم (شکل ۸):



شکل ۸- کیسه خالی!

اول از همه از ۱ شروع می‌کنیم. اگر ۱ زوج بود، یک سنگ در کیسه می‌اندازیم و گرنه بی‌خیال می‌شویم. آیا ۱ بر ۲ بخش‌پذیر است؟ نه؛ پس بی‌خیال شده و به سراغ عدد بعدی یعنی ۲ می‌رویم.

آیا ۲ بر ۲ بخش‌پذیر است؟ بله؛ پس یک سنگ در کیسه می‌اندازیم. قبل از شروع شمارش، در کیسه ما چند سنگ وجود داشت؟

دانشجو: هیچی! به عبارتی ۰ تا!

استاد: و الان چند تا؟

دانشجو: یه سنگ انداختیم و شد ۱ عدد. ریاضی اول دبستانه استاد؟ 😊

استاد: خب! می‌رویم به سراغ عدد بعد که ۳ است. آیا ۳ بر ۲ بخش‌پذیر است؟ نه؛ پس بی‌خیال شده و به سراغ عدد بعدی یعنی ۴ می‌رویم.

آیا ۴ بر ۲ بخش‌پذیر است؟ البته! پس یک سنگ دیگر به داخل کیسه می‌اندازیم و همین‌طور پیش می‌رویم تا به عدد آخر برسیم. در نهایت تعداد سنگ داخل کیسه نشان‌دهنده چیست؟

دانشجو: تعداد عددهای زوج؟ آهان...

استاد: حالا می‌بینیم با همین کیسه چطور می‌توانیم برنامه‌مان را راحت‌تر بنویسیم! اسم این کیسه را مثلاً بگذاریم count و چون این کیسه اول کار خالی است، پس مقدار آن را صفر می‌گذاریم، یعنی:

```
count = 0
```

حالا چه کار کنیم؟

دانشجو: به نظر الان باید همون طور که خودمون به عنوان یه انسان از ۱ تا ۱۰ رو در نظر می‌گرفتیم، از یه حلقه استفاده کنیم؛ مثلاً:

```
for num in range(1, 10):
```

حالا برای هربار اجرای حلقه، به یکی از اعداد ۱ تا ۱۰ دسترسی داریم؛ بار اول num میشه ۱، بار بعد num میشه ۲ و بار بعد num میشه ۳ و ... ما در برنامه قبلی یه شرط نوشتیم که اگه باقیمانده تقسیم بر ۲ صفر بود، برامون عدد رو چاپ کنه، یعنی:

```
for num in range(1,10):
```

```
    if num%2==0:
```

```
        print(num)
```

ولی ما الان قصدمون چاپ کردن این اعداد نبود، بلکه می‌خواستیم تعداد اون‌ها رو به دست بیاریم.

استاد: بسیار عالی! با روشی که خودمان به عنوان یک انسان کار می‌کردیم، یعنی ایده کیسه، چیزی به ذهنمان می‌رسد. ما یک کیسه داشتیم به اسم count که اول خالی بود؛ یعنی مقدار آن صفر بود:

```
count = 0
```

حال هربار که به یک عدد زوج رسیدیم، نظرت چیست که یکی به count اضافه کنیم؟ به عبارتی انگار یک سنگ در کیسه بیندازیم؟

```
count = count + 1
```

دانشجو: آهان!؛ یعنی برنامه ما به این شکل می‌شه:

```
count = 0
for num in range(1,10):
    if num%2==0:
        count = count + 1
```

استاد: درود بر تو! فقط این که برنامه درنهایت قرار است چه کار کند؟

دانشجو: قرار بود تعداد اعداد زوج رو نشان بده... آهان درسته؛ بعد از تموم شدن حلقه، تعداد count را پرینت کنه، یعنی:

```
print(count)
```

به عبارتی کل برنامه ما شد:

```
count = 0
for num in range(1, 10):
    if num%2==0:
        count = count + 1
print(count)
```

استاد: بسیار عالی! فقط به نظرت اینکه اگر برنامه ما بدون هیچ توضیحی فقط یک عدد را پرینت کند، برای مخاطب مفهوم است؟ بهتر نیست یک جمله بنویسیم و درنهایت آن عدد را پرینت کنیم؟ مثلاً این جمله که تعداد اعداد زوج برابر است با فلان.

دانشجو: فکر خوبیه، یعنی:

```
count = 0
for num in range(1, 10):
    if num%2==0:
        count = count + 1
```

```
print('Count of even numbers:', count)
```

اینم نتیجه‌ای که به من نشون داد:

Count of even numbers: 4

استاد: درود بر تو! حالا که دیدیم برنامه ما تا عدد ۱۰ به خوبی کار کرد، می‌توانیم حالا آن را تا عدد ۱۰۰۰ بگذاریم و از اجرای آن لذت ببریم:

```
count = 0
for num in range(1, 1000):
    if num%2==0:
        count = count + 1
print('Count of even numbers:', count)
```

و این هم نتیجه برنامه:

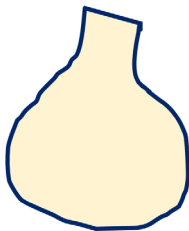
Count of even numbers: 499

مثالی دیگر

حال فرض کنید بخواهیم مجموع اعداد زوج کمتر از ۱۰۰۰ را پیدا کنیم. ببینم مثلاً برای اعداد کوچک‌تر از ۱۰، ما به عنوان یک انسان چه کار می‌کنیم؟

یکی از راه‌حل‌ها این است که ابتدا مانند مثال قبل، یک کیسه خالی را در نظر بگیریم

(شکل ۹):



شکل ۹- کیسه خالی

و حال از عدد ۱ شروع کنیم. اگر عددی زوج بود، کل آن را به کیسه اضافه کنیم. آیا ۱ زوج است؟ پس از آن می‌گذریم.

به عدد ۲ می‌رسیم. آیا عددی زوج است؟ بله، پس الآن این عدد را در داخل کیسه می‌اندازیم. الآن مجموعاً داخل کیسه ۲ است. همین‌طور عدد بعدی و ... در نهایت پس از اتمام کار، داخل کیسه چیست؟

دانشجو: در مثال قبلی، به‌ازای هر عدد زوج، یه سنگ داخل کیسه می‌نداختیم و به‌عبارتی در انتهای کار، تعداد اعداد زوج رو داشتیم؛ اما اینجا هربار به عدد زوج می‌رسیم، کل عدد رو به کیسه اضافه می‌کردیم و قاعدتاً مجموع تمام اعداد زوج رو داریم.

استاد: درود بر تو! آیا می‌توانی چیزی که گفتی را در پایتون پیاده کنی؟

دانشجو: به نظرم خیلی شبیه مثال قبلیه. کافیه یه کیسه خالی رو در نظر بگیریم و اسم اون رو مثلاً `count` بذاریم که چون در ابتدای کار کیسه خالیه، مقدار اون در ابتدا صفره، یعنی:

```
count = 0
```

استاد: چیزی که نوشتی درست است، اما به نظرت چون هدف ما از این کیسه شمارش نیست بلکه هدف مجموع است، بهتر نیست نام مناسب‌تری برای این کیسه انتخاب کنیم؟ مثلاً `total`:

```
total = 0
```

دانشجو: منم موافقم! بعد نیاز داشتیم که هربار به یه عدد از ۱ تا ۱۰ دسترسی داشتیم که این کار رو با حلقه `for` انجام می‌دادیم:

```
for num in range(1, 10):
```

سپس باید هر عدد رو چک کنیم که زوج یا نه. می‌تونیم از شرط استفاده کنیم:

```
if num%2==0:
```

حالا اگه شرط برقرار بود، یعنی عدد زوج بود، باید عددمون رو به کیسهٔ total اضافه کنیم:

```
total = total + num
```

به عبارتی برنامهٔ ما شد:

```
total = 0
```

```
for num in range(1, 10):
```

```
    if num%2==0:
```

```
        total = total + num
```

عجب چیزی نوشتیم! الآن اجراش می‌کنم تا لذت ببرید! ... اجراش کردم! چرا هیچ اتفاقی نمی‌افته؟ حتماً اشتباه اجراش کردم! بذارین دوباره اجرا کنم... نه انگار هیچ اتفاقی نمی‌افته؟

استاد: هدف ما از برنامه چه بود؟

دانشجو: می‌خواستیم مجموع اعداد کمتر از ۱۰ به کاربر نشون داده بشه؟

استاد: آیا ما وقتی کار محاسبه تمام شد، این کار را انجام دادیم؟

دانشجو: آهان! دوباره یادم رفت 😊! وقتی حلقه تموم شد، باید در نهایت این total که این همه به زحمت به دست آمده رو به کاربر نشان بدیم؛ مثلاً بنویسیم حاصل جمع اعداد، فلان مقدار است:

```
print('Sum of the even numbers is', total)
```

یعنی برنامهٔ ما شد:

```
total = 0
```

```
for num in range(1, 10):  
    if num%2==0:  
        total = total + num  
print('Sum of the even numbers is', total)
```

و این هم نتیجه اجرای برنامه:

Sum of the even numbers is 20

استاد: حالا که دیدیم برنامه ما به خوبی کار کرد، می‌توانیم به جای اعداد ۱ تا ۱۰، اعداد ۱ تا ۱۰۰۰ را به برنامه بدهیم:

```
total = 0  
for num in range(1, 1000):  
    if num%2==0:  
        total = total + num  
print('Sum of the even numbers is', total)
```

نتیجه اجرای برنامه:

Sum of the even numbers is 249500

راستی در پایتون می‌شود به جای:

```
total = total + num
```

نوشت:

```
total += num
```

می‌توانید اگر خواستید از این نوع نگارش هم استفاده کنید. حالا یک سؤال، دستور زیر معادل چیست؟

```
total += num
```

دانشجو: به نظرم شبیه قبلیه، یعنی:

```
total = total - num
```

استاد: و این نگارش معادل چیست؟

```
= num*total
```

دانشجو: به نظرم:

```
num*total = total
```

و احتمالاً دستور:

```
total /= num
```

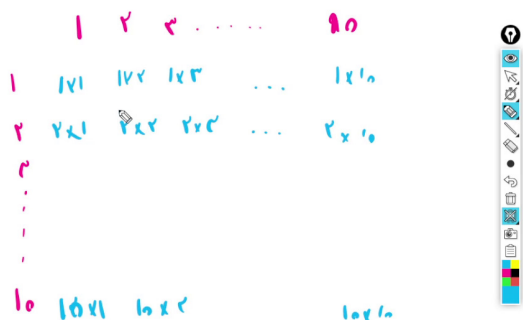
معادل این دستور باشه

```
total = total / num
```

استاد: درود بر تو!

یادگیری مفهوم حلقه‌های تودرتو!

خب برویم سراغ مفهومی جالب و مهم. فرض کنید بخواهیم یک برنامه پایتون بنویسیم که جدول ضرب را برایمان بنویسد. همان طور که خاطرتان است، جدول ضرب به این شکل بود (با خط میخی برایتان روی تخته در شکل ۱۰ نوشتم!):



۱	۲	۳	...	۱۰
۱	۲	۳	...	۱۰
۲	۴	۶	...	۲۰
۳	۶	۹	...	۳۰
...	...	...	...	...
۱۰	۲۰	۳۰	...	۱۰۰

شکل ۱۰- جدول ضرب... یادش به خیر!

دانشجو: چه سخت شد ددد! اصلاً نمی‌فهمم باید چیکار کنیم!

استاد: آن جمله مشهور هرکرا یادت نرود! هر برنامه پیچیده، زمانی یک برنامه ساده بوده که کار می‌کرده است! اگر نمی‌توانیم کل این برنامه را بنویسیم، نظرتان چیست برنامه‌ای بنویسیم که فقط سطر اول را حساب کند؟ یعنی این سطر که دورش مستطیل کشیدم (شکل ۱۱):



شکل ۱۱- سطر اول جدول ضرب

متوجه الگوی خاصی می‌شوید؟

دانشجو: استاد دلت خوشه‌ها! چه الگویی!!! اما... یه لحظه صبر کنید ببینم... انگار عدد اول ثابت همون ۱ هست و عدد دوم دازه از ۱ تا ۱۰ تغییر می‌کنه.

استاد: وقتی یک کار تکراری داریم، ذهن ما به چه سمتی می‌رود؟

دانشجو: حلقه؛ آهان مثلاً یه همچنین حلقه‌ای رو بذارید بنویسم:

```
for i in range(1, 11):
    print(1*i)
```

بذارید امتحانش هم کنم... آهان، چه خوب جواب داد:

- 1
- 2
- 3
- 4

5
6
7
8
9
10

عجب چیزی نوشتم. اما برای خطوط بعدی چیکار کنم؟ یه راهی که به ذهنم می‌رسه، برای هر سطر، یه حلقه جدا بذارم؛ مثلاً برای سطر دوم چون عدد ۲ ثابت:

```
for i in range(1, 11):
    print(2*i)
```

و همین‌طور برای ردیف‌های دیگه؛ همیشه همچنین چیزی:

```
for i in range(1, 11):
    print(1*i)
```

```
for i in range(1, 11):
    print(2*i)
```

```
for i in range(1, 11):
    print(3*i)
```

```
for i in range(1, 11):
    print(4*i)
```

تا آخر.

استاد: درود بر تو! بسیار زیبا به مسئله فکر کردی. سؤالی از تو دارم؛ باز کد ما پر از تکرار نیست؟ نظرت چیست که من اسم متغیر انتخابی تو را به col تغییر دهم؟

مخفف کلمه column یا همان ستون که نمایانگر ستون‌های ما در جدول ضرب باشد:

```
for col in range(1, 11):  
    print(1*col)
```

و حال قبول داری که انگار کل همین حلقه هم دارد به ازای هر ردیف تکرار می‌شود؟
پس نظرت چیست که یک حلقه بزرگ‌تر به اسم for row تعریف کنیم و در نهایت
row*col چاپ کنیم؟

```
for row in range(1, 11):  
    for col in range(1, 11):  
        print(row*col)
```

دانشجو: استاد می‌فهمم ولی متوجه نمی‌شم!!! می‌شه یه بار مثل قبل، خودمون رو
جای کامپیوتر بنذاریم و برنامه رو ردگیری کنیم که من راحت‌تر برنامه‌ای که نوشتم
رو درک کنم؟

استاد: چرا که نه! برو تا برویم! خودمان را به جای کامپیوتر می‌گذاریم. کامپیوتر به
خط اول یعنی این خط می‌رسد:

```
for row in range(1, 11):
```

و به همین دلیل یک متغیر به اسم row می‌سازد. مقدار آن را برابر با چه قرار
می‌دهد؟

دانشجو: برابر با ۱ تا ۱۱؟

استاد: نه یادمان است که متغیر ما می‌شد اولین عضو که اینجا عدد ۱ است، به
عبارتی الان انگار:

```
row = 1
```

حال کامپیوتر وارد این حلقه شده و به شکل این خط می‌شود:

```
for col in range(1, 11):
```

الآن چه اتفاقی می‌افتد؟

دانشجو: دیگه الآن حواسم هست. 😊 به متغیر به نام col و مقدار اون رو برابر با اولین عضو یعنی ۱ قرار می‌ده، انگار:

```
col = 1
```

استاد: درود بر تو! حال کامپیوتر وارد حلقهٔ دوم شده و به این خط می‌رسد:

```
print(row*col)
```

مقدار row و col در حال حاضر چقدر است؟

دانشجو: استاد ۱ هستن.

استاد: درواقع انگار دستور زیر دارد اجرا می‌شود:

```
print(1*1)
```

حال نکتهٔ مهم ماجرا! الآن داخل حلقهٔ دوم هستیم و چون تمام شد، کامپیوتر به شروع حلقهٔ دوم برمی‌گردد (شروع حلقه‌ای که الآن داخلش هستیم)؛ یعنی به این خط برمی‌گردد (با فلش در شکل ۱۲ نشان دادم):

```
for row in range(1, 11):
    for col in range(1, 11):
        print(row*col)
```



شکل ۱۲

یعنی به این خط:

```
for col in range(1, 11):
```


و حال مقدار متغیر col می شود عدد ۲ و کامپیوتر وارد خط بعد می شود:

```
print(row*col)
```

و مقدار col برابر ۲ و مقدار row هم که تغییر نکرده برابر ۱ است؛ پس انگار همچنین دستوری را داریم:

```
print(1*2)
```

حال دوباره مطابق شکل 12 به ابتدای حلقه ای که در آن هستیم، یعنی این خط برمی گردد:

```
for col in range(1, 11):
```

و حال مقدار متغیر col می شود عدد ۳ و کامپیوتر وارد خط بعد می شود:

```
print(row*col)
```

و مقدار col برابر ۲ و مقدار row هم که تغییر نکرده برابر ۱ است؛ پس انگار:

```
print(1*3)
```

و دوباره این موضوع تکرار می شود تا جایی که col برابر ۱۰ می شود:

```
print(1*10)
```

و به عبارتی تا الآن به خوبی سطر اول جدول ضرب را چاپ کرده ایم؛ اما حال که حلقه دوم تمام شده، چه اتفاقی می افتد؟ کامپیوتر نفس راحتی کشیده و از دست حلقه داخلی خلاص شده و ادامه می دهد. اما یادمان است که یک حلقه بزرگ تر داریم، پس کامپیوتر به ابتدای این حلقه بیرونی برمی گردد (شکل ۱۳):



```
for row in range(1, 11):
    for col in range(1, 11):
        print(row*col)
```

شکل ۱۳

و حال row تغییر کرده و برابر با دومین عضو یعنی عدد ۲ می‌شود. سپس وارد خط بعد می‌شویم:

```
for col in range(1, 11):
```

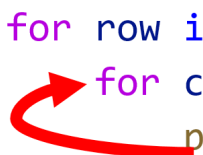
توجه دارید که مجدداً داریم حلقه دوم را اجرا می‌کنیم؟ یعنی الآن انگار تازه حلقه دوم شروع می‌شود؛ پس col برابر با اولین عضو یعنی عدد ۱ می‌شود و در نتیجه در خط بعد:

```
print(row*col)
```

مقدار row برابر با عدد ۲ و مقدار col برابر با عدد ۱ است؛ یعنی:

```
print(2*1)
```

توجه دارید که در حلقه دوم هستیم، یعنی کامپیوتر به ابتدای همین حلقه برمی‌گردد:



```
for row in range(1, 11):
    for col in range(1, 11):
        print(row*col)
```

یعنی این خط:

```
for col in range(1, 11):
```

و الآن مقدار col برابر با دومین عضو یعنی عدد ۲ شده و در نتیجه در این خط:

```
print(row*col)
```

مقدار row برابر با عدد ۲ و مقدار col برابر با عدد ۲ است، یعنی:

```
print(2*2)
```

و دوباره به ابتدای حلقه برمی گردیم، یعنی:

```
for col in range(1, 11):
```

و الآن مقدار col برابر با سومین عضو یعنی عدد ۳ شده و در نتیجه در این خط:

```
print(row*col)
```

مقدار row برابر با عدد ۲ و مقدار col برابر با عدد ۳ است، یعنی:

```
print(2*3)
```

و به همین ترتیب تمام جدول ضرب چاپ می شود. این برنامه تا جایی ادامه دارد که به آخرین عضو row یعنی برابر با ۱۰ و col برابر با ۱۰ برسیم و آنگاه کامپیوتر یک نفس راحت کشیده و برنامه تمام می شود.

دانشجو: استاد بسیار عالی! الآن حس می کنم متوجه شدم! فقط سؤالی که داشتم، وقتی این برنامه رو اجرا می کنم، همه اعداد، پشت سرهم، هر عدد در یک خط جدید ظاهر می شه، در حالی که ما مدنظرمون چیزی شبیه جدول ضربه که اعداد هر سطر جدول ضرب در یه سطر باشن و برای سطر بعدی وارد خط بعدی بشیم.

استاد: نکته این است که دستور print به صورت پیش فرض، هر بار اجرا شود، یک خط جدید ایجاد می کند. نظرت چیست در گوگل جستجو کنیم، مثلاً:

How to not print new line after each print?

دانشجو: فکر خوبیه استاد!

استاد: همان طور که می بینیم، دستور print یک آرگومان اختیاری دارد به نام

end، که به صورت پیش فرض برابر با '\n' است. \n در دنیای کامپیوتر به معنای خط جدید (همان مخفف new line) است؛ برای مثال اگر بنویسیم:

```
print('Salam Python \nI am Pedram Porbaha\nHello')
```

و اجرا می‌کنیم، می‌بینیم در خروجی نوشته می‌شود:

```
Salam Python
I am Pedram Porbaha
Hello
```

حال به صورت پیش فرض هر موقع ما می‌نویسیم، مثلاً:

```
print(1)
```

انگار نوشته‌ایم:

```
print(1, end='\n')
```

و چون این پارامتر end به صورت پیش فرض برابر با \n است، ما نیازی نداشتیم که این پارامتر را وارد کنیم.

حالا برگردیم به کد جدول ضرب:

```
for row in range(1, 11):
    for col in range(1, 11):
        print(row*col)
```

همان طور که می‌بینیم، چون در دستور print، پارامتر end را وارد نکرده‌ایم، انگار پایتون هربار که پرینت می‌کند، یک خط جدید ایجاد می‌کند. نظرتان چیست که end را برابر با "، یعنی یک فاصله بگذاریم تا بین هر مورد پرینت شده، تنها یک فاصله قرار گیرد؟ یعنی:

```
for row in range(1, 11):
    for col in range(1, 11):
```

```
print(row*col, end=' ')
```

دانشجو: به نظر که عالیه استاد! جالب بود؛ من هم کاراکتر 'n' رو بلد نبودم و هم این که یادم می‌ره که برای مشکلاتم در برنامه‌نویسی، در گوگل جستجو کنم. حالا استاد بذارید کد رو اجرا کنم... خیلی بهتر شد اما... انگار... دیگه همه اعداد شدن دنبال هم و بین سطرها در جدول ضرب فاصله‌ای نیست!!

استاد: درسته! چون ما هربار از دستور print در برنامه استفاده کردیم، پارامتر end آن برابر با ' ' است. باید چه کار کنیم؟

دانشجو: نمی‌شه استاد هربار که یه سطر پرینت شد، یه خط جدید پرینت کنیم؟ مثلاً یه دستور پرینت الکی بذاریم؟

```
print(' ', end='\n')
```

استاد: ایدهٔ خلاقانه‌ای بود. به نظرت این دستور رو کجای برنامه باید بذاریم؟

دانشجو:... منطقاً... بعد از وقتی که یه سطر از جدول ضرب چاپ شد.

استاد: و کجا این اتفاق می‌افتد؟

دانشجو: آهان! بعد از اتمام حلقهٔ دوم! یعنی:

```
for row in range(1, 11):
    for col in range(1, 11):
        print(row*col, end=' ')
        print(' ', end='\n')
```

استاد: بسیار نزدیک شدی اما حواست باشد چون دستوری که نوشتی از لحاظ جلوآمدگی (همان دندانه) هم‌تراز با دستور پرینت قبلی است، هربار که یک عدد پرینت می‌شود، یک خط جدید هم پرینت می‌شود. جای درست این دستور اینجاست:

```
for row in range(1, 11):
    for col in range(1, 11):
        print(row*col, end=' ')
    print('', end='\n')
```

و چون وقتی پارامتر end را نگذاریم، پایتون به‌صورت پیش‌فرض خودش end را برابر با '\n' در نظر می‌گیرد، می‌توان نوشت:

```
for row in range(1, 11):
    for col in range(1, 11):
        print(row*col, end=' ')
    print('')
```

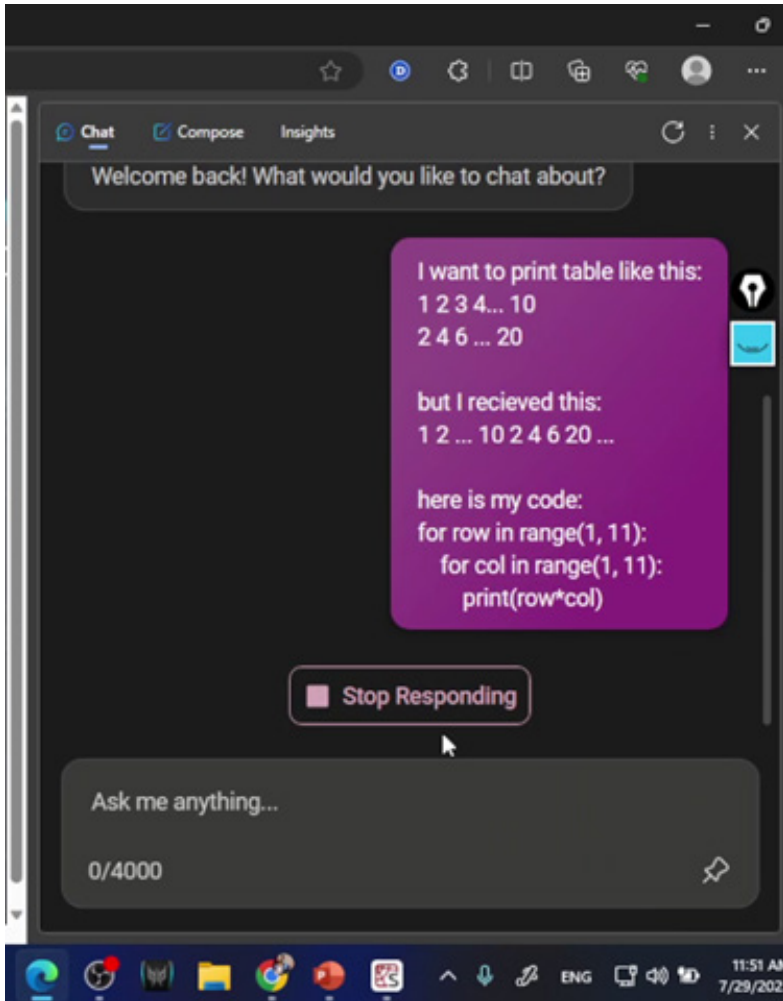
دانشجو: آهان الان گرفتم چی شد!! البته مقداری خاص بود اما این که مرحله‌به‌مرحله پیش رفتیم برام خیلی بهتر جا افتاد.

استاد: ممکنه در ابتدا، این مفاهیم مقداری جدید باشند، اما هربار سعی کنید خودتان را به‌جای کامپیوتر بگذارید و خطبه‌خط برنامه را ردگیری کنید.

استفاده از هوش مصنوعی برای رفع ایراد کد!

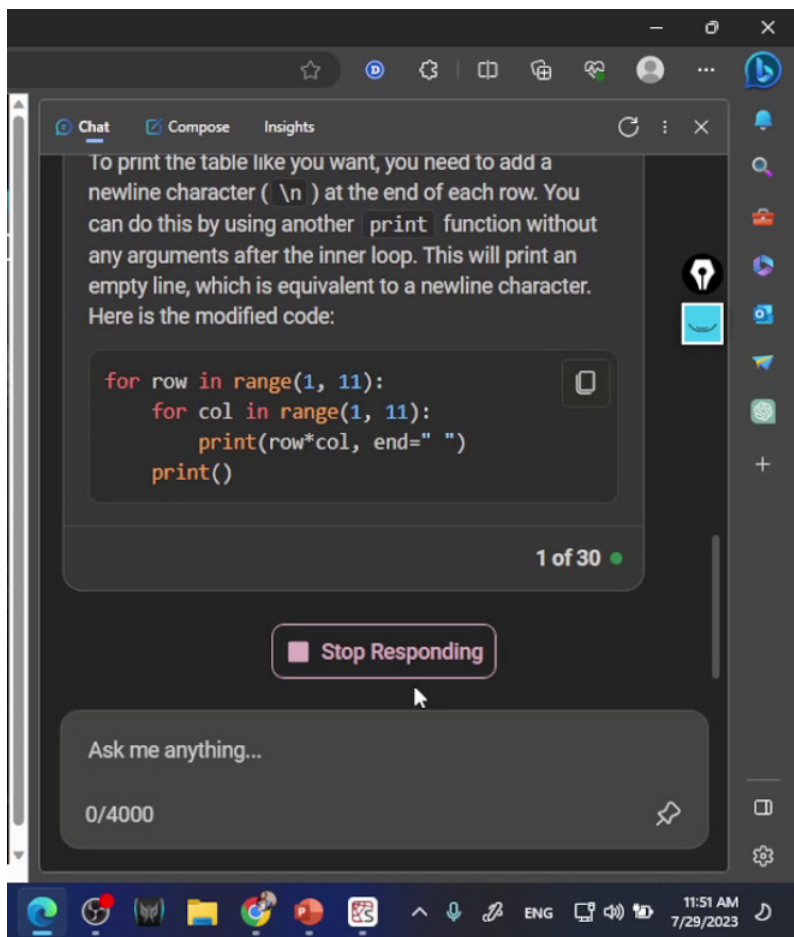
اگر یادتان باشد، یک روش رفع ایراد کد (وقتی با جستجو در گوگل به نتیجه رسیدیم)، استفاده از هوش مصنوعی است. خب، من یک هوش مصنوعی را باز کردم (من از copilot استفاده کردم اما شما می‌توانید از هر هوش مصنوعی که خواستید مانند chatGPT استفاده کنید).

در قسمت پیام، مشکلی که دارم را نوشتم (به زبان انسانی نوشتم!). در ادامه پیام نوشتم که این کد من است و کد نوشته‌شده را کپی کردم (شکل ۱۴):



شکل ۱۴- پیام نوشته شده برای هوش مصنوعی در مرورگر edge

می بینید باوجوداینکه من خیلی کلی توضیح دادم، اما چه زیبا چت جی پی تی مشکل را توضیح داده و درنهایت کد اصلاح شده را قرار داده است!!! (شکل ۱۵):



شکل ۱۵- پاسخ دریافت‌شده از هوش مصنوعی!

این هم کدی که هوش مصنوعی برای ما اصلاح کرد و نوشت!

```
for row in range(1, 11):
    for col in range(1, 11):
```



```
print(row*col, end=' ')\nprint()
```

و شما حتی در ادامه می‌توانید در مورد پاسخی که داده است مجدداً بحث کنید یا راهنمایی بخواهید یا اگر اشتباه کرده به او بگویید که اشتباهش را اصلاح کند. حیرت‌انگیز نیست؟!

دانشجو: عجب!!!! پس استاد برای چی ما داریم برنامه‌نویسی رو یاد می‌گیریم؟
استاد: ما چون در حال یادگیری اصول برنامه‌نویسی هستیم، متوجه منظور هوش مصنوعی می‌شویم و می‌توانیم حتی کد نوشته‌شده به وسیله آن را مطابق نیازمان تغییر دهیم. اگر برنامه‌نویسی بلد بودیم (همانند بسیاری از افرادی که با برنامه‌نویسی آشنا نیستند) تبدیل به یک مصرف‌کننده می‌شدیم و نمی‌توانستیم از هوش مصنوعی آن‌جور که باید و شاید بهره ببریم؛ اما کسی که با برنامه‌نویسی آشناست، آنگاه هوش مصنوعی به جای این که سریعاً جای او را بگیرد، کار او را به عنوان یک انسان بسیار تسریع می‌کند و او را فرد موفق‌تری می‌سازد؛ به عبارتی مانند یک دستیار یا کمک‌خلبان به ما کمک می‌کند.

برویم سراغ بحث شیرین تمرین‌ها! در تمرین هر درس، ممکن است که یک نکته جدید هم آموزش داده شود؛ پس حتماً توصیه می‌کنم آن‌ها را ببینید و انجامشان دهید. تمرین‌ها، به خصوص خطاهایی که دریافت می‌کنید و آزمون و خطاها (و همچنین سرچ‌هایی که در صورت نیاز انجام می‌دهید)، بسیار به روند یادگیری برنامه‌نویسی شما کمک می‌کنند. پاسخ نمونه برای تمرین‌ها نیز پس از آن‌ها قرار داده شده است.

تمرین‌های جلسه سوم

در تمرین هر درس ممکن است که یک نکته جدید هم آموزش داده شود، پس حتماً توصیه می‌کنم آن‌ها را ببینید و انجامشان دهید. تمرین‌ها و به‌خصوص خطاهایی که دریافت می‌کنید و آزمون و خطاها (و همچنین سرچ‌هایی که در صورت نیاز انجام می‌دهید)، بسیار بسیار به روند یادگیری برنامه‌نویسی شما کمک می‌کند. پاسخ نمونه برای تمرین‌ها نیز پس از آن‌ها قرار داده شده است.

تمرین‌ها

۱- برنامه‌ای بنویسید که مجموع و تعداد اعداد کوچک‌تر از ۱۰۰۰ که مضرب ۳ یا ۵ هستند را حساب کرده و چاپ کند.

۲- برنامه‌ای بنویسید که از مخاطب بپرسد چه کاری می‌خواهد انجام دهد. اگر ۱ را وارد کرد، ۱۰۰ بار به او سلام کند و اگر ۲ را وارد کرد به او ۵۰ بار امید بدهد و اگر عددی غیر از این دو را وارد کرد، پیغام خطایی با این مضمون را نمایش دهد: (خطا به جای ۱ یا ۲، اشتباهاً این کاراکتر وارد شده است)

۳- برنامه‌ای بنویسید که عددی را از مخاطب گرفته و تمام مقسوم‌علیه‌های آن را نمایش دهد (تمام اعداد کوچک‌تر از آن که عدد به آن بخش پذیر است). مثلاً اگر مخاطب وارد کرد ۶، برنامه چاپ کند: ۱، ۲، ۳ و ۶، زیرا عدد ۶ به این اعداد بخش پذیر است.

پاسخ تمرین‌های جلسه سوم

اکیداً توصیه می‌کنم که اول خودتان تمرین‌ها را حل کنید و سپس به پاسخ‌ها مراجعه کنید. در ضمن این پاسخ‌ها تنها یکی از حالت‌های ممکن نوشتن برنامه هستند و شما ممکن است به شیوه‌ای دیگر برنامه را نوشته باشید و کاملاً صحیح هم باشد. همچنین برای مشاهده توضیحات شفاهی مربوط به این تمرین‌ها، ابتدای ویدئوی جلسه بعد را ببینید (لینک ویدئو در صفحه اول کتاب گذاشته شده است):

« پاسخ تمرین اول:

```
count = 0
total = 0
for num in range(1, 1000):
    if num%3==0 or num%5==0:
        count = count + 1
        total = total + num

print('The count is as follow:', count)
print('The sum is as follow:', total)
```

به عبارتی انگار دو کیسه داریم؛ یکی برای تعداد (به نام کیسه count) و یکی برای مجموع اعداد (به نام کیسه total) که مقدار اولیه هر دو، برابر با صفر است.

تمرین تکمیلی:

اگر بخواهیم به جای مجموع، حاصل ضرب این اعداد را به دست بیاوریم، به نظرتان مقدار اولیه این کیسه را چقدر در نظر بگیریم؟ اگر دوست داشتید این برنامه را بنویسید.

راهنمایی:

مثلاً کیسه‌ای به اسم multiply تعریف می‌کنیم، ولی مقدار اولیه آن را ۱ در نظر می‌گیریم؛ چون اگر صفر در نظر بگیریم، هر عددی را در این عدد ضرب کنیم، حاصل برابر صفر می‌شود.

« پاسخ تمرین دوم:

```
name = input('What is your name?\n')
choice = input('What do you want to do? (1 or 2)\n')
if choice == '1':
    for num in range(1, 101):
        print(num, 'Hello', name)

elif choice == '2':
    for num in range(1, 51):
        print('I wish you a' , num, 'times of
        better days, my dear', name)

else:
    print('You entered wrong choice', choice)
```

سؤال تکمیلی:

هدف از نوشتن \n در انتهای متن داخل دستور input چیست؟ یک‌بار آن را بردارید و یک‌بار بگذارید و تفاوت این دو را در عمل ببینید.

نکته تکمیلی:

در نسخه‌های اخیر پایتون، یک syntax جدید اضافه شده است که کار پرینت

کردن را راحت می‌کند. به عبارتی خروجی این دو دستور معادل هم هستند:

```
print('You entered wrong choice:', choice, 'Your  
name is', name)
```

```
print(f'You entered wrong choice: {choice} Your  
name is {name}')
```

به حالت دوم، f-string می‌گویند و کافی است یک حرف f قبل از رشته خود بگذارید و متغیر یا محاسبات مدنظر خود را بین دو علامت آکولاد {} قرار دهید. پایتون خودش مقدار آن متغیر را جایگزین می‌کند که کد را خواناتر و زیباتر می‌کند.

سؤال تکمیلی:

ما در این برنامه از متغیر num استفاده‌ای کرده‌ایم؟ اگر نه، به نظرتان بهتر بود چه نامی برای متغیر حلقه خودمان انتخاب می‌کردیم؟

پاسخ:

در این موارد که از متغیر حلقه ما هیچ استفاده‌ای نمی‌شود و فقط نقش شمارنده دارد، معمولاً از متغیری با نام _ (همان آندرلاین) استفاده می‌شود که به خواننده بفهماند که این متغیر، استفاده‌ای در برنامه ندارد و کار خواندن و درک برنامه را برای خواننده انسانی راحت‌تر می‌کند، یعنی:

```
name = input('What is your name?\n')
choice = input('What do you want to do? (1 or 2)... \n')
if choice == '1':
    for _ in range(1, 101):
        print(num, 'Hello', name)
```

```
elif choice == '2':
    for _ in range(1, 51):
        print('I wish you a' , num, 'times of
              better days, my dear', name)

else:
    print(f'You entered wrong choice: {choice} Your
          name is {name})
```

« پاسخ تمرین سوم:

ما به عنوان یک انسان چه کار می‌کنیم؟ از عدد ۱ تا همان عدد پیش رفته و هربار چک می‌کنیم که عدد ما بر آن عدد بخش‌پذیر است یا نه و در صورت بخش‌پذیر بودن، آن عدد را چاپ می‌کنیم:

```
num = int(input('What number do you consider?\n'))
for i in range(1, num+1):
    if num%i == 0:
        print(i)
```

سؤال تکمیلی برای علاقه‌مندان:

عدد اول عددی است که غیر از عدد ۱ و خودش، بر هیچ عددی بخش‌پذیر نباشد (مثلاً عدد ۵ عددی اول است زیرا فقط بر ۱ و خودش بخش‌پذیر است؛ اما مثلاً عدد ۴ اول نیست، زیرا علاوه بر ۱ و خودش، بر ۲ هم بخش‌پذیر است). چطور می‌توان یک برنامه نوشت که تمام اعداد اول کوچک‌تر از مثلاً ۱۰ را چاپ کند؟ برای مثال اعداد ۲، ۳، ۵، ۷، اعداد اول کوچک‌تر از عدد ۱۰ هستند.

راهنمایی:

به عنوان یک انسان ببینید خودتان چگونه به مرحله به مرحله این کار را انجام می دهید. سپس همان کار را به زبان پایتون بنویسید. ابتدا توصیه می کنم برنامه را ساده شده بنویسید، یعنی برنامه ای که فقط چک کند یک عدد اول است یا نه. وقتی این کار را کردید، آنگاه برنامه را ارتقا دهید به شکلی که این کار را برای تمام اعداد کوچک تر از ۱۰ انجام دهد.

• فصل چهارم •

کتابخانه‌ها و متدهای رشته‌ها!

درود بر شما! تا اینجا با متغیرها آشنا شدیم، دستورات شرطی را دیدیم و متوجه شدیم که با حلقه‌ها، چگونه کارهای تکراری را انجام دهیم. البته همان طور که تا الآن هم دیدیم، هرچه جلوتر می‌رویم، با ترکیب این قابلیت‌ها قدرت بیشتری به دست می‌آوریم و روی مباحث قبلی هم مسلط‌تر می‌شویم.

کتابخانه چیست و در علوم پزشکی به چه دردی می‌خورد؟

ممکن است ما در برنامه‌ای که نوشتیم، بخواهیم با فایل اکسل کار کنیم؛ یک فایل ورد را باز کنیم و بخوانیم؛ یک عملیات خاص ریاضی را انجام دهیم؛ یک طیف EEG مغزی را بخوانیم؛ بخواهیم صوت گرفته‌شده را پردازش کنیم؛ یک مولکول شیمیایی را بخوانیم و ساختار آن را بررسی کنیم؛ یک سایت را باز کرده و اطلاعات آن را دریافت کنیم؛ از هوش مصنوعی و یادگیری ماشین بهره ببریم؛ یک جستجوی پیشرفته انجام بدهیم؛ با تصاویر یا ویدئو کار کنیم و ... برای هر کدام از این کارها فردی وجود داشته که لطف کرده و دستورات لازم را در قالب یک کتابخانه (library) به زبان پایتون نوشته و ما کافی است که آن کتابخانه را به راحتی به پایتون وارد کرده و از آن دستورها استفاده کنیم.

پایتون به کتابخانه‌های بسیار وسیع و کاربردی‌اش مشهور است. به جرئت کمتر موضوع یا کاری را می‌توانیم پیدا کنیم که کسی قبل از ما در آن رابطه، یک کتابخانه ننوشته و کار ما را راحت‌تر نکرده است. حتی اگر حیطه ما بسیار تخصصی هم باشد، باز کتابخانه‌هایی هستند که کار ما را راحت‌تر می‌کنند و می‌توانیم خودمان کسی باشیم که آن کتابخانه را نوشته و در دسترس سایرین بگذاریم تا دیگران از آن بهره ببرند. به قول شاعر:

دیگران کاشتند و ما خوردیم ما بکاریم و دیگران بخورند

برای این که کار با کتابخانه‌ها برایمان جا بیفتد، باهم یک مثال می‌زنیم. بعضی کتابخانه‌ها بسیار ساده و کوچک هستند و شاید در زمانی کمتر از یک ربع، بتوان استفاده کردن از آن‌ها را یاد گرفت و بعضی از کتابخانه‌ها بسیار وسیع و پر از امکانات هستند و حتی دوره‌های چندساعته و کتاب‌هایی برای امکانات متفاوت آن‌ها وجود دارد که در طول زمان، هر فرد می‌تواند به آن‌ها مسلط‌تر شود. از نمونه این کتابخانه‌های بسیار وسیع (به قولی آبر کتابخانه!)، می‌توان به سه کتابخانه پایتون که به قول یکی از اساتیدم به نام سه‌تفنگدار پایتون می‌توان از آن‌ها ذکر کرد: `numpy`، `pandas` و `matplotlib` هستند که مفصل برای کار با داده‌ها به کار می‌روند و بسیار قدرتمند هستند که انشا... در فصل‌های انتهایی یا جلد بعدی کتاب به آن‌ها می‌پردازیم.

برای این که کار با کتابخانه‌ها را شروع کنیم، بیایم کاری را انجام دهیم که تا الآن با دستوراتی که در خود پایتون یاد گرفتیم، نمی‌توانیم این کار را انجام دهیم. فرض کنید بخواهیم لگاریتم عدد ۵ را حساب کنیم. برای عملیات ریاضی، یک کتابخانه در پایتون وجود دارد به نام `math`. این کتابخانه نیاز به نصب ندارد به عبارتی یک کتابخانه `built-in` است و تنها کافی است که آن را وارد کنیم. برای وارد کردن (`import`)

کردن) کتابخانه کافی است بنویسیم `import` و نام کتابخانه مدنظرمان را پس از آن بنویسیم:

```
import math
```

الآن پایتون به دستورات موجود در کتابخانه `math` دسترسی دارد و می‌توانیم از دستورات آن استفاده کنیم. مثلاً یکی از دستورات این کتابخانه، دستور `log` است که لگاریتم طبیعی را حساب می‌کند، منتها چون دستور جزوی از کتابخانه `math` است نمی‌توانیم بنویسیم:

```
log(5)
```

بلکه باید بنویسیم:

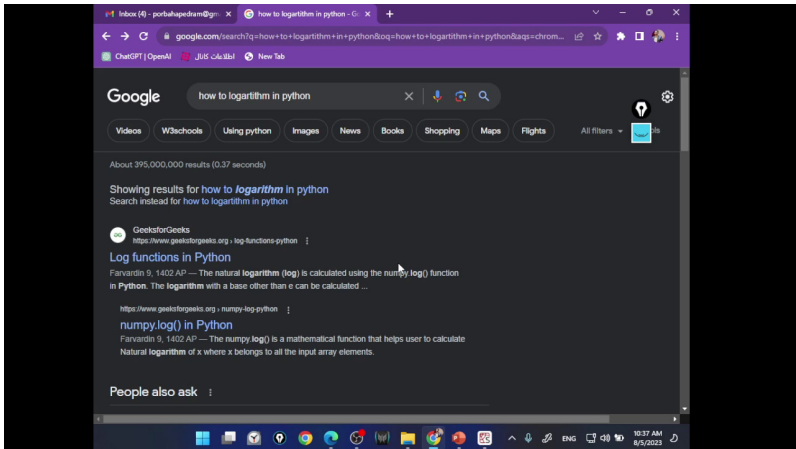
```
math.log(5)
```

تا لگاریتم طبیعی عدد ۵ را حساب کند.

دانشجو: استاد ما چه جوری می‌تونیم اسم کتابخونه‌ها رو حفظ کنیم؟ یادمون می‌ره و دستوراتش ممکنه از خاطرمون بره.

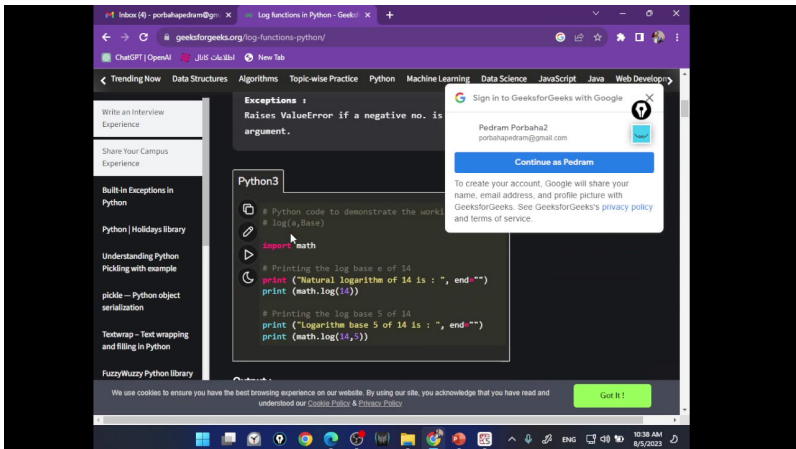
استاد: خیلی ساده! کافی است در گوگل جستجو کنیم. مثلاً من جستجو کردم که چگونه می‌توان در پایتون، لگاریتم گرفت (شکل ۱):

how to logarithm in python



شکل ۱- با جستجو در گوگل به زبان انگلیسی، معمولاً خیلی سریع تر به پاسخ می‌رسیم.

خوبی سایت‌های آموزش برنامه‌نویسی این است که در کنار آموزش، یک کد نمونه هم قرار داده شده که موضوع را به‌خوبی جا می‌اندازد (شکل ۲):



شکل ۲- صفحهٔ پیداشده به‌وسیلهٔ جستجوی گوگل

همان طور که می‌بینید، دقیقاً همان دستوری که نوشتیم را اینجا نوشته و همچنین توضیحات اضافه‌تری هم داده؛ مثلاً گفته است که در پرانتز، بعد از کاما، اگر می‌توانید حتی مبنای لگاریتم را بنویسید؛ مثلاً اگر بخواهیم لگاریتم عدد ۵ را در مبنای عدد ۲ بگیریم، کافی است بنویسیم:

```
math.log(5, 2)
```

که البته چون می‌خواهیم حاصل آن چاپ شود و به ما در خروجی نشان داده شود، مانند قبل از `print` استفاده می‌کنیم:

```
print(math.log(5, 2))
```

این نکته را از من به خاطر داشته باشید که تقریباً هر کاری که فکرش را بکنید، می‌توان با پایتون انجام داد. کافی است جستجو کنید (اگر جستجو کردید و پیدا نکردید، از هوش مصنوعی بپرسید) و دقیقاً راه درست مسلط شدن در برنامه‌نویسی همین است، زیرا به این شکل با کاربردهای عملی که موردنیاز شخص شماست، برنامه‌نویسی را یاد گرفته و به همین دلیل علاقه و اعتماد به نفس شما روز به روز افزایش پیدا می‌کند.

خب تا بحث کتابخانه `math` باز است، برویم چند دستور دیگر این کتابخانه را ببینیم. مثلاً اگر بخواهیم تانژانت یک زاویه را حساب کنیم، کافی است از دستور `tan` استفاده کنیم:

```
print(math.tan(45))
```

یا مثلاً برای محاسبه `exp` یا همان `e` به توان عدد موردنظرمان کافی است:

```
print(math.exp(3))
```

یا مثلاً شاید بایتان جالب باشد که:

```
math.pi
```

همان عدد پی را به ما می‌دهد:

```
print(math.pi)
```

دانشجو: ما که همیشه بعد از دستورات در پایتون، علامت پرانتز می‌داشتیم. اینجا نیاز نیست؟

استاد: چه سؤال خوبی! چون عدد پی یک دستور نیست و به عبارتی یک عدد است، نیازی به اجرا شدن ندارد و کافی است با همان `math.pi` مقدار آن را ببینیم.

دانشجو: آهان! و استاد یه سؤال دیگه؛ فرض کنید من چندین بار نیاز باشه که از دستور لگاریتم در برنامه استفاده کنم. نمیشه به جای این که هر بار کلمه `math` را بنویسم، یعنی:

```
import math
result = math.log(10) + math.log(4)
print(result)
```

راحت‌تر باشم و نخوام هر بار این قدر بنویسم؟

استاد: یک روش این است که کتابخانه را به این شکل `import` کنیم:

```
import math as m
result = m.log(10) + m.log(4)
print(result)
```

با استفاده از کلمه کلیدی `as` در اینجا کتابخانه `math` را به عنوان `m` وارد کردیم.

دانشجو: این هم روش خوبیه، اما استاد روشی نیست که بتونیم همون دستور مدنظرمون رو به صورت ساده بدون اسم کتابخانه بنویسیم؟ یعنی:

```
import math
result = log(10) + log(4)
```

```
print(result)
```

من این کار رو کردم، اما با این خطا روبرو شدم که دستور `log` تعریف نشده است. استاد: برای انجام این کار باید به جای این که کل کتابخانه مدنظر را `import` کنیم، تنها دستورات موردنیاز خود را `import` کنیم. برای این کار کافی است بنویسیم:

```
from math import log
result = log(10) + log(4)
print(result)
```

و به این شکل، دیگر می‌توانیم بدون نیاز به ذکر اسم کتابخانه، مستقیماً از آن دستور در برنامه خود استفاده کنیم. حتی می‌توانیم چند تا دستور را یک کتابخانه `import` کنیم:

```
from math import log, tan, exp, pi
result = log(10) + exp(4)*tan(pi/4)
print(result)
```

دانشجو: چه جالب! استاد این کتابخانه `math` کاربردی بود اما کتابخانه‌های دیگه رو هم به ما معرفی می‌کنید؟

استاد: تا انتهای دوره با کتابخانه‌های متعددی برای مدیریت فایل، خواندن اکسل و ... آشنا می‌شویم، اما یادمان نرود، هر موقع خواستید کاری را با پایتون انجام دهید، جستجو در گوگل و یا پرسیدن از چت‌بات‌های هوش مصنوعی را فراموش نکنید. من در یک پروژه، قصد داشتم که به جای تصویرها، با موسیقی‌ها کار کنم و هیچ‌چیزی در این رابطه نمی‌دانستم؛ اما با جستجو کردن و همچنین پرسش از هوش مصنوعی‌ها، توانستم دستورات موردنیاز کار خودم را در یک یا دو نیمروز یاد بگیرم و برنامه‌ای را بنویسم که موسیقی‌ها را از یک پوشه در کامپیوتر من می‌خواند و از آن‌ها الهام

می‌گرفت تا با یک روش یادگیری ماشین بتواند شبیه آن‌ها را تولید کند. من در دوره‌ای که آموزش دیده بودم، یاد گرفته بودم که با تصاویر این کار را انجام دهم اما به دلیل علاقه، همین کار را با موسیقی انجام دادم و باعث شد که با برخی کتابخانه‌های کار با صوت آشنا شوم و در همین راستا بسیار یاد بگیرم و مهم‌تر از همه، لذت بسیار بیشتری برده و اعتماد به نفس‌ام بیشتر از قبل شود.

دانشجو: خیلی ممنونم! من جدیداً خیلی به برنامه‌نویسی علاقه‌مند شدم و به خودم که میام می‌بینم دارم پشت کامپیوتر برنامه می‌نویسم، اما جدیداً گردنم درد می‌کنه! چون مجبورم خیلی روی کیبورد خم بشم! چیکار کنم؟

استاد: درود بر تو! در رابطه با این مورد که ذکر کردی، یکی از دلایل اصلی که به نظرم می‌رسد، نگاه کردن به صفحه کلید موقع تایپ است که فشار زیادی بر گردن وارد می‌کند. در این زمینه، یادگیری تایپ ده انگشتی کمک بسیار زیادی به من کرده و برخلاف انتظارم، یادگیری‌اش ساده‌تر از چیزی بود که فکر می‌کردم. سایت ایران تایپو یک از سایت‌هایی است که در این زمینه فوق‌العاده است. به صورت رایگان و به شکل یک بازی است (کافی است در گوگل جستجو کنید ایران تایپو).

هر درس‌اش، درست مثل یک بازی بوده و بیشتر از پنج دقیقه هم طول نمی‌کشد. من هم برای تایپ فارسی و هم برای انگلیسی، دوره‌شان را گذراندم (البته من آن موقع این سایت رایگان را نمی‌شناختم و در عوض از سایت یوتایپ، اکانتی یک‌ماهه خریده بودم و روی آن انجام می‌دادم). اگر به مدت چهار هفته، هر روز پنج تا ده دقیقه، تمرین این سایت را انجام بدهید و خودتان را ملزم کنید که در زندگی روزمره هم از این مهارت استفاده کنید، واقعاً یادگیری این مهارت بیشتر از یک ماه طول نمی‌کشد (حتی من آن موقع برای تفریح حین کار، به شکل یک بازی از آن استفاده می‌کردم). وقتی

هم که دوره‌اش تمام شد، برای مرور و افزایش تسلط از سایت Ten Fast Fingers استفاده کردم که تقریباً هر زبانی را پشتیبانی کرده و یک متن تصادفی را جلوی شما می‌گذارد. من پس از اتمام دوره، روزی یکی، دو سه بار (هر بار حدود یک دقیقه بیشتر طول نمی‌کشد) این بازی انجام می‌دادم.

من توانستم به همین صورت سرعتم را به پنجاه کلمه در دقیقه بدون نگاه به صفحه کلید برسانم (دوستانی هم داشتم که به ۷۰ یا ۱۰۰ هم رسیدند). این حس که می‌دانی دیگر مجبور نیستی به صفحه کلید نگاه کنی و می‌توانی به جای قوز کردن به صندلی تکیه بدهی و تایپ کنی، واقعاً لذت‌بخش است و این که می‌شود در حدود دو هفته تا یک ماه به آن رسید. از آن لذت‌بخش‌تر، پرستیز برنامه‌نویسی بهتان می‌دهد که آن هم بسیار جذاب است! (۵)

همچنین توصیه می‌کنم بیشتر از ۲۵ دقیقه نیز نشسته باقی نمانیم و یک تایمر گذاشته (خود ویندوز یک تایمر درون برنامه clock خود دارد) و در صورت تمام شدن تایمر، برای دو دقیقه هم شده از جایمان بلند شویم و قدمی بزنیم. بحث سلامتی بحثی بسیار جدی است و توصیه می‌کنم همان‌طور که به فکر افزایش مهارت و سرمایه‌گذاری روی بخش کاری و حرفه‌ای زندگی خود هستید، حتماً به بخش سلامت جسمی خود نیز فکر کنید (مخصوصاً الآن که به سن بالاتر و شروع بیماری‌های مزمن نرسیده‌ایم و وقت بسیار مناسبی است؛ البته حتی اگر به این شرایط بیماری هم رسیده باشیم، باز هم می‌توان با رعایت نکاتی ساده در سبک زندگی، خود را ارتقا دهیم). اگر به این موضوع علاقه دارید، توصیه می‌کنم پادکست «جافکری» فصل «حال خوب بدن من» را گوش دهید (این پادکست را می‌توانید با دائلود نرم‌افزارهای پخش پادکست مانند castbox یا گوگل پادکست در حین مسیر رفتن به سرکار و برگشت گوش دهید

البته فکر کنم اگر جستجو کنید هم بتوانید فایل‌های صوتی هر قسمت را از سایت خود جافکری نیز دانلود کنید). این فصل ۱۰ قسمت بیشتر نیست و به‌شخصه به من کمک کرد که رعایت نکات ساده‌ای را در سبک زندگی روزانه‌ام، نسبت به قبل انجام دهم.

خب اگر موافق هستید برویم سراغ موضوع بعدی!

متدهای رشته‌ها

فرض کنید می‌خواهیم نام و جنسیت مخاطب را دریافت کنیم:

```
name = input('Please enter your name:\n')
gender = input('Please enter your gender:\n')
```

حالا می‌خواهیم کاری کنیم که اگر مخاطب آقا بود (به عبارتی جنسیت برابر با male بود)، چاپ کند:

```
Hello Mr Pedram
```

و اگر خانم بود (یعنی جنسیت برابر با female بود)، چاپ کند:

```
Hello Mr Leila
```

برای این کار می‌توانیم این برنامه را بنویسیم:

```
name = input('Please enter your name:\n')
gender = input('Please enter your gender:\n')
```

```
if gender == 'male':
    print('Hello Mr', name)
elif gender == 'female':
    print('Hello Ms.', name)
```

برنامه را هم اجرا می‌کنیم و می‌بینیم به خوبی کار می‌کند؛ اما نکته‌ای وجود دارد! اگر مخاطب کلمهٔ Male را با حروف بزرگ وارد کرد چطور؟ بیا ببینیم برنامه را ردگیری کنیم ببینیم چه اتفاقی می‌افتد. برنامه به این خط می‌رسد:

```
if gender == 'male':
```

آیا این شرط برقرار است؟ یعنی آیا gender برابر با 'male' است؟ خیر! اکنون gender برابر با 'Male' است. به همین دلیل این شرط برقرار نیست و پایتون وارد شرط بعدی می‌شود:

```
elif gender == 'female':
```

که این شرط هم برقرار نیست و در نتیجه هیچ چیزی چاپ نمی‌شود! چیزی که ما نمی‌خواهیم! به نظر تان برای حل این مشکل چه کنیم؟

دانشجو: استاد می‌تونیم به مخاطب بگیم حتماً male یا female را با حروف کوچک تایپ کنه. مثلاً در توضیحات input برای متغیر gender بنویسیم:

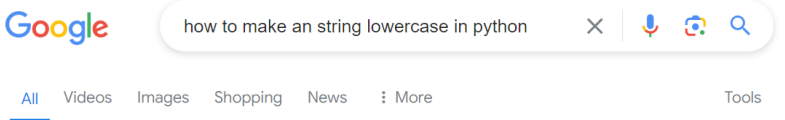
```
gender = input('Please enter your gender(male or female):\n')
```

استاد: این روش خوبی است؛ اما مشکل آن این است که شاید مخاطب به توضیحات ما توجه نکرد و باز کار خودش را کرد و نوشت Male یا اصلاً شاید ناامدانه این اشتباه را کرد که اشتباه خیلی خاصی هم نیست. آیا راهی وجود دارد که بتوان کاری کرد که برنامه در این شرایط هم درست کار کند؟ به عبارت دیگر اگر مخاطب وارد کرد Male یا MALE یا mALE یا هر شکل دیگری، باز هم برنامه ما درست کار کند؟

دانشجو: نمی‌دونم واقعاً! البته شاید... شاید بشه یه کاری کرد... به نظرم اگه می‌شد که کاری کنیم که gender حروفش به حروف کوچک تبدیل بشه، اون موقع دیگه مخاطب هر جوری male را وارد کنه، آخرش به male با حروف کوچک تبدیل بشه...

استاد: چه ایدهٔ خلاقانه‌ای؛ دقیقاً و حالا چطور می‌توان یک رشته را به حروف کوچک تبدیل کرد؟

دانشجو: نمی‌دونم... بذارین این بار نوبت منه که جستجو کنم! 😊 شاید بتونم پیداش کنم... الآن جستجو کردم (شکل ۳):



شکل ۳

استاد: هیچ ایرادی ندارد که با غلط تایپی یا گرامری به انگلیسی جستجو کنید! فقط جستجو کنید. بعد از مدتی همین مورد، زبان انگلیسی شما را نیز بسیار بهبود می‌دهد. دانشجو: خب؛ لینک اول رو باز کردم، باز نشد. الآن میرم سراغ لینک دوم (شکل ۴):

Python String lower()

The `lower()` method converts all uppercase characters in a **string** into lowercase characters and returns it.

Example

```
message = 'PYTHON IS FUN'

# convert message to lowercase
print(message.lower())

# Output: python is fun
```

شکل ۴- سایت دوم

طوری که اینجا نوشته و مثال زده، کافیه یه نقطه بعد از متغیرمون بذاریم و بعدش از دستور `lower()` استفاده کنیم یعنی:

`gender.lower()`

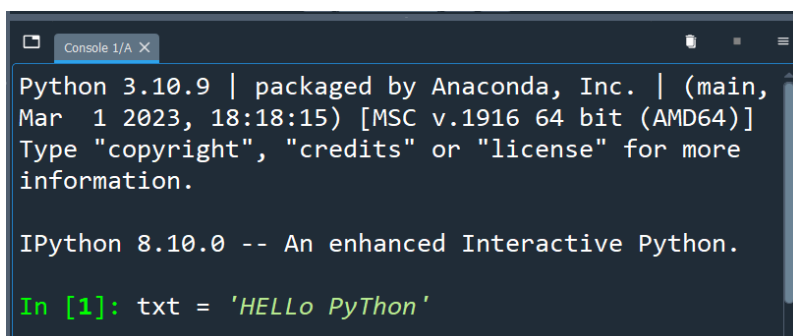
استاد: درود بر تو! کیف کردم که با جستجو کردن توانستی دستور را پیدا کنی. نظرت چیست که برویم در همان قسمت پایین سمت `spyder`، یعنی همان قسمت `Console` این دستور را چک کنیم؟ خاطرت است که `Console` یا همان `Shell` در اسپایدر کجاست؟

دانشجو: ...آره یادمه... همون جایی که دستورها رو تک خط تک خط می‌دادیم و اجرا می‌کرد.

استاد: دقیقاً! برای یادگیری دستورات جدید و امتحان کردن آن‌ها، این قسمت بسیار مناسب است، چون سریعاً خروجی آن دستور را نشانمان می‌دهد و اگر یادت باشد نیازی هم به دستور `print` نداشت.

خب پس بیا یک متغیر دلخواه تعریف کنیم، مثلاً:

`txt = 'HELLO PyThon'`



```

Python 3.10.9 | packaged by Anaconda, Inc. | (main,
Mar 1 2023, 18:18:15) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more
information.

IPython 8.10.0 -- An enhanced Interactive Python.

In [1]: txt = 'HELLO PyThon'
  
```

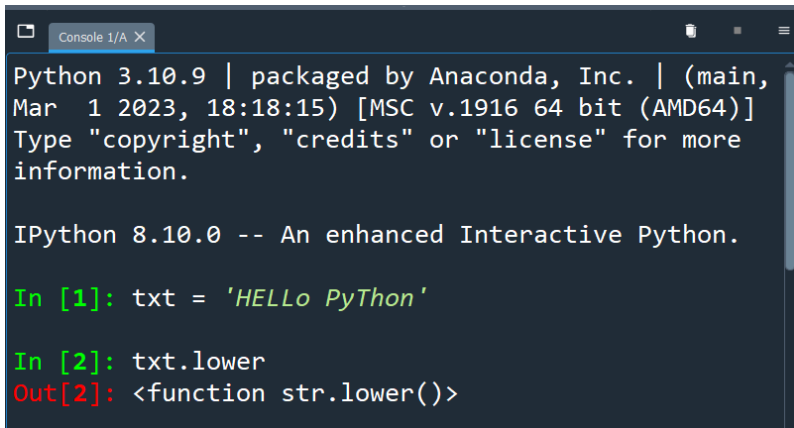
شکل ۵- قسمت کنسول یا همان `shell` در اسپایدر، در پایین سمت راست `spyder` قرار دارد و برای اینکه دستوری که جدیداً یاد گرفته‌ایم را امتحان کنیم و یاد بگیریم بسیار مناسب است.

همان طور که خودت جستجو کردی و یاد گرفتی، رشته‌متدهای متفاوت زیادی دارند که یکی از آنها، متد `lower` است. برای این که ما بخواهیم از متدها استفاده کنیم، کافی است یک نقطه بعد از اسم آن رشته بگذاریم و نام متد را بنویسیم.

دانشجو: استاد من این دستور رو نوشتم:

`txt.lower`

اما حروف رشته من کوچک نشد و در عوض یک پیام دیگه دریافت کردم:



```
Python 3.10.9 | packaged by Anaconda, Inc. | (main, Mar 1 2023, 18:18:15) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 8.10.0 -- An enhanced Interactive Python.

In [1]: txt = 'HELLO PyThon'

In [2]: txt.lower
Out[2]: <function str.lower()>
```

شکل ۶- پیام دریافت شده

استاد: متد `lower` یک دستور است و دستورها در پایتون..؟

دانشجو: آهان دستورها در پایتون نیاز به علامت پرانتز برای اجرا دارن؛ یعنی درستش اینه:

`txt.lower()`

چه جالب؛ الآن درست شد و با حروف کوچک نوشت:

`'hello python'`

```

In [1]: txt = 'HELLO PyThon'

In [2]: txt.lower
Out[2]: <function str.lower()>

In [3]: txt.lower()
Out[3]: 'hello python'

In [4]: |

```

شکل ۷ خروجی متد lower()

استاد: وقتی دستورها یا متدها یا ... را بدون علامت پرانتز می‌گذاریم، پایتون آن‌ها را اجرا نمی‌کند و به عبارتی انگار خود دستور را به ما نشان می‌دهد.

حالا یک درخواست از تو دارم؛ در همین محیط shell لطفاً بنویس txt و به من بگو که محتوای متغیر txt چیست:

txt

دانشجو: الآن امتحان می‌کنم... نوشتم استاد... ولی همون رشته قبلی رو چاپ کرد نه رشته‌ای که حروفش کوچک شده:

'HELLO PyThon'

استاد: به عبارتی یعنی متد lower()، متغیر رشته مدنظر را با حروف کوچک برمی‌گرداند ولی متغیر را تغییر نمی‌دهد و متغیر ما همان قبلی است. چطور می‌توانیم این را در متغیر ذخیره کنیم؟

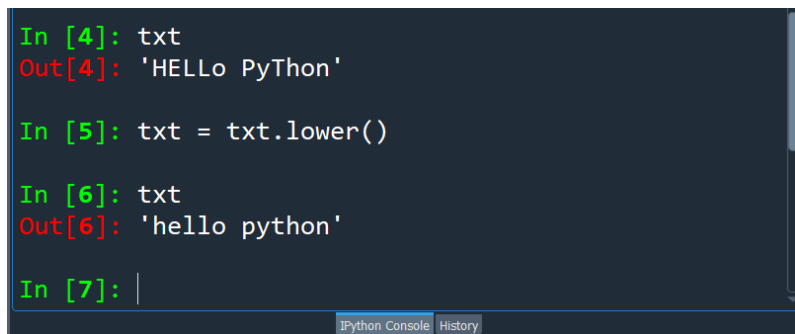
دانشجو: فکر کنم باید در همان متغیر txt ذخیره‌اش کنیم، یعنی:

```
txt = txt.lower()
```

ببینیم محتوای txt الآن چیه؟

```
txt
```

چه جالب، درست شد و الآن محتوای متغیر txt با حروف کوچکه (شکل ۸):
'hello python'



```
In [4]: txt
Out[4]: 'HELLO PyThon'

In [5]: txt = txt.lower()

In [6]: txt
Out[6]: 'hello python'

In [7]: |
```

شکل ۸- نتیجه دستور در Console پایتون

استاد: درود بر تو! می‌بینی چقدر راحت‌تر در قسمت shell می‌توانی کاربرد یک دستور را مرور کنی و با آن آزمون و خطا کنی و آن را بهتر یاد بگیری؟ حالا با دانشی که به دست آوردیم، چطور می‌توانیم برنامه قبلی را که نوشته بودیم و نام و جنسیت را از مخاطب می‌گرفت، اصلاح کنیم؟

دانشجو: خیلی راحت! کافیه بعد از دریافت متغیر gender اون رو با متد lower به حروف کوچک تبدیل کنیم:

```
name = input('Please enter your name:\n')
gender = input('Please enter your gender:\n')
```

```
gender.lower()
```

```
if gender == 'male':
    print('Hello Mr', name)
elif gender == 'female':
    print('Hello Ms.', name)
```

استاد: متد lower آیا مقدار متغیر را هم تغییر می‌داد یا نیاز بود که آن را در متغیر هم ذخیره کنیم؟

دانشجو: ای‌وای حواسم نبود! باید خروجی رو در همان متغیر gender ذخیره می‌کردم، یعنی:

```
gender = gender.lower()
```

پس برنامه ما کامل شد:

```
name = input('Please enter your name:\n')
gender = input('Please enter your gender:\n')
gender = gender.lower()
```

```
if gender == 'male':
    print('Hello Mr', name)
elif gender == 'female':
    print('Hello Ms.', name)
```

بنارین امتحانش کنم... چه جالب چند بار امتحان کردم. یک‌بار جنسیت را وارد کردم MALE و بار دیگر وارد کردم mAle و بار دیگر MALE و در همه‌حال درست کار کرد و نوشت:

```
Hello Mr. Reza
```

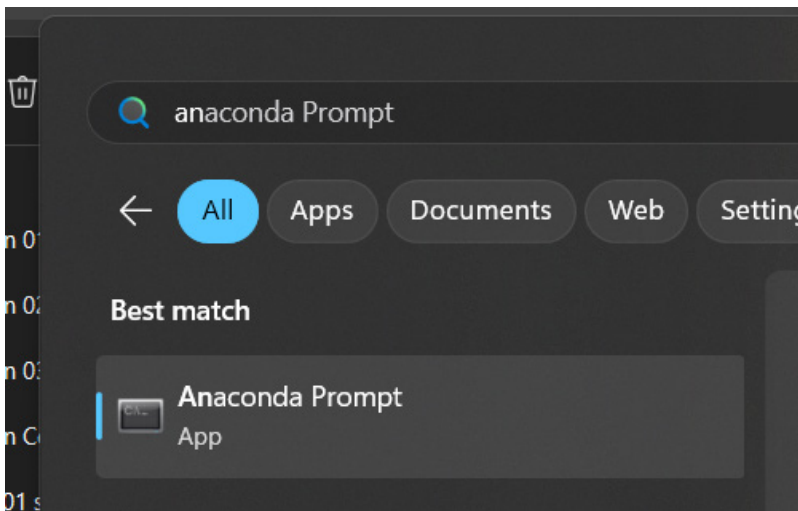
استاد: بسیار عالی! حال اگر خواستیم که کتابخانه‌ای که در پایتون به شکل پیش فرض وجود ندارد را نصب کنیم و از آن استفاده کنیم، چه کنیم؟ نصب کتابخانه در پایتون بسیار راحت است. مثلاً فرض کنید می‌خواهیم کتابخانهٔ `pyautogui` را که یک کتابخانه برای کنترل خود کار موس و کیبورد به وسیلهٔ پایتون است را نصب کنیم. اگر در محیط پایتون بنویسیم:

```
import pyautogui
```

با این خطا روبرو می‌شویم:

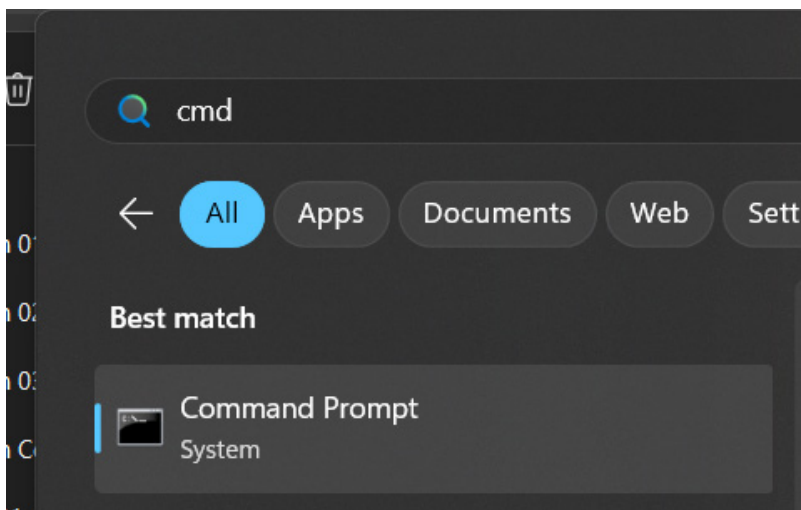
`ModuleNotFoundError: No module named 'pyautogui'`

که به این معناست که کتابخانه (یا به عبارت تخصصی، ماژول `pyautogui`) پیدا نشد و نیاز دارد که نصب شود. برای نصب یک کتابخانه کافی است ابتدا در محیط استارت ویندوز جستجو کنیم، `Annaconda prompt` (شکل ۹):



شکل ۹- استارت ویندوز و جستجو `Annaconda prompt`

می‌توانید به جای `cmd`، `Annconda prompt` را جستجو کنید (شکل ۱۰) و فرقی ندارد؛ اما توصیه می‌کنم اگر مثل فصل اول، پایتون را از طریق آناکوندا نصب کرده‌اید، همان `anaconda prompt` را جستجو کنید.



شکل ۱۰- می‌توانید به جای `Annaconda prompt`، از `cmd` یا همان `command prompt` استفاده کنید. پس از آنکه محیط `Annaconda prompt` (یا محیط `cmd`) باز شد، کافی است دستور زیر را بنویسید:

`pip install` نام کتابخانه

مثلاً چون ما می‌خواهیم کتابخانه `pyautogui` را نصب کنیم، کافی است بنویسیم:
`pip install pyautogui`
و تمام! اگر اینترنت وصل باشد، خود `pip` شروع به دانلود و نصب کتابخانه می‌کند و وقتی تمام شد، به ما پیام `successful` یا همان موفقیت را نشان می‌دهد (معمولاً

نیازی به تحریم‌شکن نیست اما اگر با خطای connection روبرو شدید، تحریم‌شکن خود را متصل کنید). پس از اتمام نصب، کافی است که محیط `Annaconda prompt` یا `cmd` را ببندید و سپس در پایتون اگر یکبار دیگر این دستور را بنویسید:

```
import pyautogui
```

می‌بینید که دیگر با خطا روبرو نمی‌شوید و به این معناست که کتابخانه مد نظر ما به خوبی نصب شده است و می‌توانید از دستورات این کتابخانه (شبه همان کتابخانه `math` که باهم مرور کردیم استفاده کنید).

اصول نصب کتابخانه به همین شکل است که به شما گفتم؛ اما برای هر کتابخانه بد نیست که یک سرچ در گوگل کنید و می‌بینید که دستور دقیق نصب آن کتابخانه که کافی است آن دستور را در `prompt` خود کپی کنید نوشته شده است و کافی است همان دستور را کپی کنید.

سایر متدهای رشته‌ها

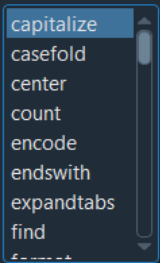
استاد: رشته‌ها متدهای زیادی دارند که هر کدام کار خاصی را انجام می‌دهند. برای این که با این متدها بیشتر کار کنیم و راحت‌تر آزمون و خطا کنیم، دوباره به همان محیط کنسول اسپایدر (پایین سمت راست اسپایدر) برویم. یک متغیر دلخواه (مثلاً `txt`) تعریف می‌کنیم، مثلاً:

```
txt = 'Hello pYthon'
```

حالا اگر پس از نوشتن اسم متغیر که اینجا `txt` است، یک نقطه بگذاریم، لیست متدهای آن برایمان نمایش داده می‌شود (اگر نمایش داده نشد، کلید `Tab` یا اگر باز هم نشد کلید `Shift + Space` را روی کیبورد فشار دهید):

```
In [7]: txt = 'Hello pYthon'

In [7]:
In [7]:
In [7]:
In [8]: txt.
```



IPython Console History

شکل ۱۱- پس از نوشتن نقطه، با زدن کلید Tab روی کیبورد یا در بعضی سیستم‌ها با زدن دکمه Shift + Space، لیست متدها برآید نمایش داده می‌شود.

همان‌طور که می‌بینید، یک رشته متدهای زیادی را دارد. می‌توانید خودتان آن‌ها را امتحان کنید و ببینید هر کدام چه می‌کند یا این که جستجو کنید و ببینید. مثلاً به نظرتان متد زیر چه می‌کند؟

```
txt.upper()
```

دانشجو: چه جالب حروف کلمه رو بزرگ کرد:

```
'HELLO PYTHON'
```

استاد: یا مثلاً متد زیر؟

```
txt.title()
```

دانشجو: امتحان که کردم مثل تیترها، حرف اول کلمات رو بزرگ و بقیه حروف رو کوچک نگه داشت:

```
'Hello Python'
```

استاد: بسیار عالی! یا مثلاً متد

```
txt.islower()
```

چه کار می کند؟ همان طور که از اسم متدهایی که با `is` شروع می شوند مشخص است، این متدها `True` یا `False` برمی گرداند (مثلاً این متد در صورتی که تمامی حروف کوچک باشند `True` برمی گرداند و در غیر این صورت `False`). این متدها به نظرتان کجا به درد می خورند؟

دانشجو: دارن من رو یاد شرطها می ندازن.

استاد: درود بر تو! می توانیم در `if`ها از آنها بهره ببریم، مثلاً:

```
if txt.islower():
    print('Your text is lower cased')
else:
    print('Your text is NOT lower case')
```

که در صورتی که این شرط برقرار بود، برای مخاطب این جمله را می نویسد یا مثلاً متد:

```
txt.isalpha()
```

که در صورتی که اگر رشته ما فقط از حروف تشکیل شده بود (نه اعداد)، `True` برمی گرداند. یا مثلاً متد `endswith`:

```
txt = 'word.docx'
txt.endswith('.docx')
```

که بررسی می کند اگر رشته ما با چیزی که نوشتیم تمام شده، `True` برمی گرداند که همان طور که می بیند به خصوص برای بررسی پسوند فایل ها می تواند بسیار مفید باشد (آنچه در نام فایل ها پس از نقطه می آید، پسوند آن است و نوع فایل را مشخص می کند).

یا مثلاً متد `count` که برای ما می شمارد که چند بار از یک کاراکتر مدنظر در رشته ما وجود دارد، برای مثال:

```
txt = 'I love python'
txt.count('o')
```

به ما خروجی ۲ را برمی‌گرداند زیرا در رشته ما ۲ بار حرف o وجود دارد.

به نظرتان متد replace چه می‌کند؟ این دستور را امتحان کنید:

```
txt = 'I love python'
txt.replace('o', '-')
```

دانشجو: چه جالب؛ تمامی حروف o را با علامت - جایگزین کرد و نوشت:

```
'I l-ve pyth-n'
```

استاد: و به نظرتان این دستور چه می‌کند؟

```
txt = 'HeTTTTlllo worldTTTT'
txt.replace('T', '')
```

دانشجو: یعنی هر جا کاراکتر T رو دید با هیچ چیز جایگزین کنه؟ بذارید امتحان کنم... چه جالب! کار کرد و تمامی کاراکترهای T رو حذف کرد:

```
'Hello world'
```

استاد: حالا چک کن ببین که آیا مقدار txt تغییر کرده؟

دانشجو: نه استاد مقدارش همون قبلیه... پس اگه بخوام مقدار txt هم تغییر کنه

باید نتیجه دستور رو در txt ذخیره می‌کردم یعنی:

```
txt = 'HeTTTTlllo worldTTTT'
txt = txt.replace('T', '')
print(txt)
```

الآن درست شد!

استاد: درود بر تو! متدی به نام strip هم داریم که جالب است و کاراکترهای فاصله

و خالی اول و آخر کلمه را حذف می‌کند، مثلاً:


```
txt = '      Hello world      \n\n'
txt = txt.strip()
print(txt)
```

و به عنوان خروجی می بینید که تمامی کاراکترهای فاصله و خط جدید ($\backslash n$) و کاراکتر Tab ($\backslash t$) را حذف کرده است:

```
Hello world
```

دانشجو: متد جالبی بود؛ کاربرد اون چیه استاد؟

استاد: خیلی از مواقع است که کاراکترهای فاصله قبل و بعد از کلمات خود داریم که مزاحم هستند. به خصوص در کارهای آنالیز داده و دیتاماینینگ زیاد این مورد را می بینیم و به این روش می توانیم داده های خود را تمیزتر کنیم.

فکر می کنید متد find چه می کند؟

```
txt.find
```

دانشجو: وقتی نوشتم و پرانتز رو باز کردم برام به صورت help این رو نوشت (شکل ۱۲):

```
(sub: str, start: SupportsIndex | None
= ..., end: SupportsIndex | None = ...,
/) -> int
```

```
S.find(sub[, start[, end]]) -> int
```

Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are

شکل ۱۲ - توضیحات help برای متد find

استاد: بیا باهم این help رو بررسی کنیم (البته یادمان است که اگر متوجه نشدیم کافی‌ست که در گوگل جستجو کنیم). اولاً این دستور آیا نیاز به ورودی دارد؟ می‌بینیم که بله، یک ورودی دارد؛ sub که از نوع رشته یا همان str است؛ اما بقیه ورودی‌های تابع، یک علامت مساوی بعد از خود دارند یعنی وارد کردن آن‌ها اجباری نیستند و اگر ما آن‌ها را وارد نکنیم، خود پایتون به شکل پیش‌فرض به آن‌ها مقدار می‌دهد.

در ادامه توضیح داده که این متد چه چیزی را به‌عنوان خروجی می‌دهد (اصطلاحاً return می‌کند). نوشته، موقعیت اولین باری که آن کاراکتر مدنظر ما ظاهر شده را نشان می‌دهد.

دانشجو: استاد مجدد می‌فهمم ولی متوجه نمی‌شم! می‌شه با یه مثال پیش بریم؟

استاد: چرا که نه. مثلاً فرض کن رشته ما، متغیری به اسم txt است:

```
txt = 'Hello pYthon'
```

حال اگر بنویسیم:

```
txt.find('H')
```

چه خروجی‌ای می‌دهد؟ به ما خروجی عدد ۰ داد. چرا؟ چون کجا در رشته مدنظر ما حرف H وجود دارد؟

دانشجو: اولین موقعیت... اما چرا خروجی داد ۰؟ و خروجی ندا ۱؟

استاد: این موضوع را همیشه در خاطر داشته باشید که در پایتون شمارش از صفر شروع می‌شود. موقعیت اول در پایتون با عدد ۰، موقعیت دوم با اندیس ۱، موقعیت سوم با اندیس ۲ و ... مشخص می‌شود. برای این که یادتان نرود، سیستم شماره‌گذاری پایتون شبیه سیستم شماره طبقات در آسانسور است! اولین طبقه را در آسانسور نمی‌گوییم اولین طبقه؛ می‌گوییم همکف یا همان طبقه ۰ و به همین ترتیب.

دانشجو: آهان الآن گرفتم چرا ۰ رو خروجی داد. پس اگر بنویسم:

```
txt = 'Hello pYthon'  
txt.find('e')
```

منطقاً باید بنویسه ۱، چون در دومین موقعیت حرف e رو داریم و یا اگر بنویسیم:

```
txt.find('o')
```

باید بنویسه... نمی‌دونم باید چی بنویسه چون دو تا حرف o در رشته‌مون داریم. بذارید امتحان کنم؛ نوشت ۴...، پس انگار اولین حرف o را در نظر گرفته...

استاد: دقیقاً همان‌طور که در برنامه help دیدیم. حال اگر کاراکتری که در رشته ما وجود ندارد را بنویسیم، برای مثال:

```
txt.find('z')
```

چه خروجی‌ای دریافت می‌کنیم؟

دانشجو: نوشت ۱-... چه جالب؛ پس اگه اون کاراکتر وجود نداشت عدد ۱- رو به ما بر می‌گردونه...

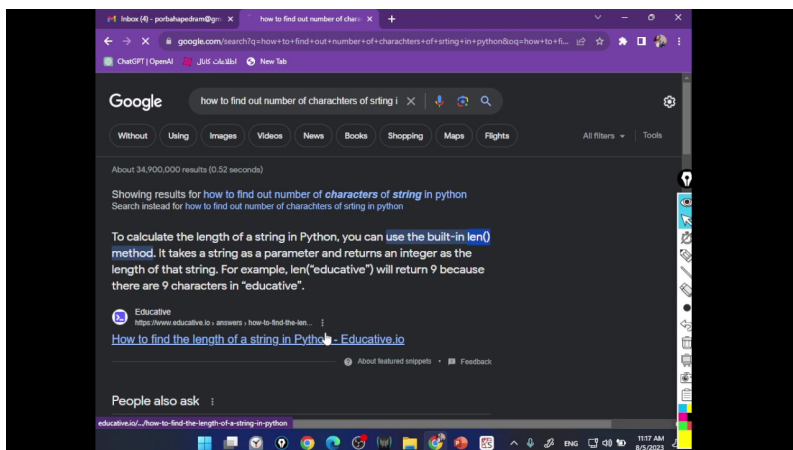
استاد: دقیقاً! در خط آخر متد help هم نوشته بود که در صورتی که آن کاراکتر پیدا نشود مقدار ۱- را خروجی می‌دهد.

فکر می‌کنید چطور می‌توان طول یک رشته را در پایتون متوجه شد؟ به عبارتی بفهمیم رشته ما چند کاراکتر دارد؟

دانشجو: فکر کنم می‌دونم منظور تون چیه! در گوگل جستجو می‌کنیم!

استاد: درود بر شما! برای مثال من در گوگل جستجو کردم (شکل ۱۳):

how to find out the number of characters of a string in python



شکل ۱۳- جستجو در گوگل

مثل اینکه یک دستور built-in در پایتون وجود دارد به نام len. دستور built-in به چه معنا بود؟ یعنی اینکه در خود پایتون وجود دارد و نیازی به نصب کتابخانه خاصی ندارد، اما اگر نیاز به نصب هم داشت، یاد گرفتیم که چطور به سرعت این کار را انجام دهیم. من برای این که با مثال این دستور را یاد بگیرم روی لینک زدم تا صفحه باز شود (شکل ۱۴):

```

IndexError                                Traceback (most recent call last)
Cell In[15], line 3
      1 seq = 'ATCAGC'
      2 for i in range(0, 7):
----> 3     print(seq[100])

IndexError: string index out of range
    
```

شکل ۱۴- باز کردن یکی از لینک‌های حاصل از جستجو

و می‌بینیم مثالی که زده:

```
len('educative')
```

که خروجی این دستور شده ۹، زیرا این رشته ۹ کاراکتر دارد یا در مثال ما:

```
txt = 'Hello pYthon'
```

```
len(txt)
```

که نوشت ۱۲، زیرا این رشته ۱۲ کاراکتر دارد (خود فاصله نیز یک کاراکتر محسوب می‌شود). راستی تا یادم نرفته است به شما بگویم که بعضی مواقع که مشکل یا خطای خود را در گوگل جستجو می‌کنیم، سایتی به اسم `stackoverflow` در نتایج نشان داده می‌شود. این سایت، سایت بسیار خوبی است و به این شکل است که افراد در آن سؤال می‌پرسند و این قدر این سایت جمعیت بزرگی دارد که به سرعت به آن‌ها پاسخ داده می‌شود. البته معمولاً نیازی نیست که خودمان در آن سؤال بپرسیم و معمولاً کسی قبلاً سؤال ما را پرسیده است و می‌توانیم در بین پاسخ‌هایی که به آن سؤال داده شده ببینیم کدام امتیاز بیشتری آورده و همان جواب را مطالعه کنیم. اگر سؤالی بود که جدید بود و نیاز به کمک داشتید، می‌توانید در این سایت بپرسید و به سرعت کسی وجود دارد که به آن جواب دهد. حتی توصیه می‌کنم خود شما هم هرازگاهی به این سایت سر بزنید و اگر می‌توانید، به سؤالی پاسخ دهید. این مورد باعث می‌شود امتیاز شما در این سایت بالاتر برود و خودش یک رزومه برای شماست. من خودم به‌شخصه در این سایت هم سؤال پرسیده‌ام و هم اگر توانسته باشم، به سؤال‌هایی پاسخ داده‌ام. سایت لذت‌بخشی است. هرازگاهی به آن سر بزنید.

و به‌عنوان آخرین دستور، چک کنید دستور زیر چه خروجی‌ای می‌دهد:

```
txt = 'Hello pYthon'
```

```
if 'H' in txt:
```

```
print('your text has H charachter')
```

دانشجو: شرط اجرا شد! چه باحال، یعنی می‌تونیم چک کنیم که کاراکتری در رشته ما وجود داره یا نه. استاد حتماً باید یه حرف باشه یا می‌تونیم یه کلمه رو هم چک کنیم؟

استاد: نظرت چیست؟ خودت امتحان می‌کنی؟

دانشجو: راست می‌گید! حواسم نبود! مثلاً این رو بذارین اجرا کنم:

```
txt = 'Hello pYthon'
if 'Hello' in txt:
    print('your text has Hello')
```

چه باحال جواب داد!

استاد: دقیقاً! و اگر بخواهیم این شرط را منفی کنیم، یعنی بگوییم اگر کلمه Hello وجود نداشت چگونه؟

دانشجو: یادم نیست... باید منفی‌اش کنیم...

استاد: چه کلمه‌ای شرط را برعکس می‌کرد؟

دانشجو: یادمه and و or داشتیم که ربطی به منفی کردن نداشتن و ... آهان کلمه کلیدی not یعنی:

```
txt = 'Hello pYthon'
if not 'Salam' in txt:
    print('your text has not Salam!')
```

آهان، چه باحال! چون کلمه Salam در متن ما وجود نداشت، شرط اجرا شد.

استاد: درود بر تو! در پایتون می‌توان کلمه not را به این شکل هم در دستور in نوشت:

```
txt = 'Hello pYthon'
if 'Salam' not in txt:
    print('your text has not Salam!')
```

که به این شکل نوشتن آن مرسوم تر است.

یک مثال علوم پزشکی تر!

برویم یک مثال زیستی تر با رشته‌ها بزنیم. احتمالاً با توالی‌های DNA آشنا باشید. توالی‌هایی که اطلاعات وراثتی ما در آن‌ها ذخیره شده و به ما به ارث رسیده شده‌اند. همان‌طور که احتمالاً بدانید توالی‌های DNA معمولاً از ۴ نوع حرف ساخته شده‌اند: C، T، A و G؛ برای مثال یک توالی DNA می‌تواند این باشد:

ATACGCAAAATTTTCGGG...

یک رشته DNA می‌تواند حتی شامل هزاران حرف (یا به اصطلاح درست‌تر، هزاران نوکلئوتید باشد).

اگر موافق باشید یک رشته DNA را در پایتون نوشته و در یک متغیر (مثلاً متغیر seq مخفف sequence) ذخیره کنیم:

```
seq = 'ATACGCAAAATTTTCGGAAC'
```

به نظر تان چطور می‌توانیم بفهمیم رشته DNA ما چند نوکلئوتید (یا به عبارت خودمانی چند حرف) دارد؟

دانشجو: ... فکر کنم همون دستوری که در اینترنت جستجو کردیم رو یاد گرفتیم... دستور len:

```
print(len(seq))
```

استاد: دقیقاً! همان‌طور که احتمالاً بدانید، رشته DNA در بدن طی فرآیند رونویسی

به رشته RNA تبدیل می‌شود. رشته RNA تفاوتی که با DNA دارد این است که به جای حرف T، دارای حرف U است. چگونه می‌توانیم به سرعت بفهمیم رشته معادل RNA رشته DNA مدنظر ما چیست؟

دانشجو: ... یعنی انگار باید هر حرف T را با حرف U جایگزین کنیم؛ آهان! متد `replace`:

```
seq.replace('T', 'U')
```

ولی چون می‌خواهم مقدارش رو جایی ذخیره کنم، بذارین یه متغیر تعریف کنم به اسم مثلاً `rna_seq`:

```
seq = 'ATACGCAAATTTTCGGAAC'
rna = seq.replace('T', 'U')
print(rna)
```

عجب چیز باحالی شد! نوشت:

```
AUACGCAAAUUUCGGAAC
```

اگه می‌خواستیم خودم دستی تک به تک حروف T رو با U عوض کنم کلی طول می‌کشید!

استاد: بسیار عالی! برنامه‌نویسی می‌تواند کارهای تکراری ما را بسیار تسریع کند! حالا یک سؤال؛ چطور بشماریم که چند عدد نوکلئوتید C در کلمه ما وجود دارد؟

دانشجو: خیلی راحت! با متد `count`:

```
seq = 'ATACGCAAATTTTCGGAAC'
print(seq.count('C'))
```

استاد: درود بر تو!

دسترسی به موارد موردنظر در رشته!

بیا امتحان کنیم که خروجی دستور زیر چیست؟

```
seq = 'ATACGCAAATTTTCGGAAC'  
seq[0]
```

دانشجو: برای من نوشت:

'A'

استاد: یادتان است که دیدیم شمارش در پایتون از صفر شروع می شود؛ یعنی الان کاراکتر صفرم یا به عبارتی همان کاراکتر اول را به ما برگرداند. حال اگر بنویسیم:

```
seq[1]
```

دانشجو: نوشت:

'T'

یعنی همان کاراکتر دوم را برگرداند.

حالا اگر بنویسیم:

```
seq[100]
```

دانشجو: خطا داد استاد! نوشت (شکل ۱۵):

```
-----  
IndexError                                Traceback (most recent call last)  
Cell In[15], line 3  
    1 seq = 'ATCAGC'  
    2 for i in range(0, 7):  
----> 3     print(seq[100])  
  
IndexError: string index out of range
```

شکل ۱۵ - خطای نمایش داده شده

استاد: دقیقاً! چون رشته ما عضو ۱۰۰ ام ندارد! و به همین دلیل به ما خطای index

out of range داد؛ چون اندیسی که دادیم (یعنی اندیس ۱۰۰) از تعداد کاراکترهای رشته ما فراتر است.

حالا اگر بنویسیم:

```
seq[0:2]
```

چطور؟

دانشجو: چه جالب نوشت:

‘AT’

به عبارتی کاراکتر صفر و یک رو برگردوند؛ اما مگه ما ننوشتیم از صفر تا دو؟

استاد: یادت است که در حلقه‌های دستور range دیدیم که پایتون تا یکی قبل از چیزی که نوشتیم می‌رود؟ به همین دلیل فقط ۰ و ۱ را برگرداند. مثلاً اگر بنویسیم:

```
seq[1:3]
```

کاراکتر ۱ و ۲ را برمی‌گرداند، یعنی:

‘AT’

یک نکته کنکوری! معمولاً در پایتون اگر بخواهیم بنویسیم:

```
seq[0:3]
```

آن صفر اول را حذف می‌کنند و مستقیماً می‌نویسند:

```
seq[:3]
```

حالا اگر بنویسیم چطور؟ به عبارتی کاراکتر ۱- کدام کاراکتر می‌شود؟

```
seq[-1]
```

دانشجو: حرف C رو برگردوند... یعنی کاراکتر آخر؟

استاد: دقیقاً! و کاراکتر ۲- چطور؟

seq[-2]

دانشجو: چه جالب؛ کاراکتر یکی مانده به آخر!

حلقه‌های رشته‌ها! یک مثال ژنتیک!

فرض کنید رشته DNA زیر را داریم که شامل ۶ نوکلئوتید است:

seq = 'ATCAGC'

می‌خواهیم بدون استفاده از متد count بفهمیم که چند کاراکتر C در این رشته وجود دارد. چگونه می‌توانیم این کار را انجام دهیم؟

دانشجو: سخت شد استاد! نمی‌دونم!

استاد: هر وقت برایت سخت شد، ببین خودمان به‌عنوان یک انسان چه مراحل برای انجام این کار طی می‌کنیم.

کیسه یادت است؟ یک کیسه خالی در نظر می‌گیریم. از حرف اول شروع می‌کنیم؛ اگر این حرف برابر با C بود، یک عدد به کیسه اضافه می‌کنیم و در نهایت در داخل کیسه چه داریم؟

دانشجو: تعداد حرف‌های C در رشته‌مون که شمردیم.

استاد: خب چطور می‌توانیم به تک‌تک حروف رشته‌مان دسترسی داشته باشیم؟...

دانشجو: نمی‌دونم استاد!!!

استاد: یادت نرود که هر برنامه پیچیده، زمانی برنامه ساده‌ای بوده که کار می‌کرده است. بیاییم ساده‌تر نگاه کنیم. چگونه می‌توانیم به حرف اول رشته خود دسترسی پیدا کنیم؟

دانشجو: ... با دستور زیر؟

```
seq[0]
```

استاد: و چگونه می‌توانیم به حرف دوم دسترسی پیدا کنیم؟

دانشجو: قاعدتاً:

```
seq[1]
```

استاد: و چگونه به عضو سوم؟

دانشجو: با دستور:

```
seq[2]
```

استاد: آیا می‌توانی یک حلقه بنویسی که هربار، حلقه همین کار را برای ما از ۰ تا عدد ۶ انجام دهد؟

دانشجو: آهان فکر کنم گرفتم چی شد! یعنی:

```
seq[i]
```

و ما می‌خواهیم از ۰ بریم تا ۶، یعنی:

```
seq = 'ATCAGC'
for i in range(0, 7):
    print(seq[i])
```

الآن اگه برنامه رو اجرا کنم، باید قاعدتاً هر حرف رو جداگانه برام روی صفحه چاپ کنه؛ اما استاد ارور گرفتیم! نوشت:

```
index out of range
```

A
T
C
A
G
C

```
-----
IndexError                                Traceback (most recent call last)
Cell In[17], line 3
      1 seq = 'ATCAGC'
      2 for i in range(0, 7):
----> 3     print(seq[i])

IndexError: string index out of range
```

شکل ۱۶ - خطای نشان داده شده

استاد: در پیام خطا می بینیم که در دستور:

```
print(seq[i])
```

دچار خطا شده ایم. نظرت چیست که هر بار مقدار i را هم در حلقه پرینت کنیم تا ببینیم قبل از این که خطا بگیریم مقدار i چقدر است؟ مثلاً:

```
seq = 'ATCAGC'
for i in range(0, 7):
    print('i= ', i)
    print(seq[i])
```

دانشجو: این کار رو که کردم نوشت:

```
i= 0
A
i= 1
T
i= 2
C
```

```
i= 3
```

```
A
```

```
i= 4
```

```
G
```

```
i= 5
```

```
C
```

```
i= 6
```

و بعد از این، خطا داده.

استاد: مقدار `i` در هنگامی که خطا رخ داده چقدر بوده؟

دانشجو: طبق چیزی که در خروجی می‌بینم `i` برابر با ۶ بوده:

```
i= 6
```

استاد: و در کدام دستور با خطای `index out of range` مواجه شدیم؟

دانشجو: در این خط:

```
print(seq[i])
```

استاد: و چون `i` برابر با ۶ بوده یا انگار در دستور زیر در حلقه دچار خطای `out of`

`range` شدیم:

```
print(seq[6])
```

رشته ما کلاً چند حرف داشت؟

دانشجو: رشته ما این بود:

```
seq = 'ATCAGC'
```

و در کل ۶ حرف داشت.

استاد: و شمارش در پایتون از چه عددی شروع می‌شود؟

دانشجو: از عدد ۰ یعنی عضو ششم حداکثر ... عضو پنجم می‌شه! و منم نوشتیم عضو ششم رو بده!! چقدر من نادونم!!! چطور حواسم نبود!! ۰، ۱ و ۲ و ۳ و ۴ و ۵ ولی من نوشته بودم از ۰ تا ۷ یعنی خود ۶ رو هم آورده بودم!!!

استاد: با خودت مهربان باش! همین مهارت رفع خطا به صورت مرحله به مرحله، از مهارت کدنویسی مهم تر است. حال به جای سرزنش کردن خودت، به من بگو چطور می‌توانیم کد را اصلاح کنیم؟

دانشجو: ممنون استاد! کافیه به جای:

```
range(0, 7)
```

بنویسیم:

```
range(0, 6)
```

به عبارت دیگه کد کامل ما به این شکل می‌شه:

```
seq = 'ATCAGC'
for i in range(0, 6):
    print('i= ', i)
    print(seq[i])
```

استاد: درود بر تو! حال نظرت چیست که یک کیسه خالی تعریف کنیم و تعداد کاراکتر C را بشماریم؟

دانشجو: بله حتماً استاد! یه متغیر به اسم count تعریف می‌کنم و چون در ابتدا کیسه خالیه، مقدار رو صفر در نظر می‌گیرم:

```
count = 0
```

استاد: و چگونه در هر بار حلقه چک کنیم که کاراکتر ما برابر با C است یا نه؟

دانشجو: قاعدتاً از if استفاده می‌کنیم، یعنی:

```
if seq[i] == 'C':
    count += 1
```

که هربار کاراکتر ما برابر با حرف C بود، یکی به تعداد count اضافه کنه؛ و در نهایت هم که حلقه ما تموم شد، نتیجه را پرینت کنه (دیدین استاد این بار یادم بود 😊):

```
seq = 'ATCAGC'
```

```
count = 0
for i in range(0, 6):
    if seq[i] == 'C':
        count += 1
print(f'The count of C in your DNA is {count}')
```

استاد: بسیار عالی! فقط یک جای برنامه کمی به دل من نمی چسبید؛

دانشجو: کجا استاد؟

استاد: این قسمت

```
for i in range(0, 6):
```

ما خودمان شمردیم که توالی ما (seq) ۶ کاراکتر دارد. آیا به جای ۶ می‌توان کاری کرد که خود کامپیوتر تعداد کاراکتر را بشمارد؟

دانشجو: یادمه دستور زیر، تعداد حروف رشته ما (seq) رو به ما نشون می‌داد:

```
len(seq)
```

استاد: پس نظرت چیست به جای عدد ۶، همین را جایگزین کنیم، یعنی:

```
for i in range(0, len(seq)):
```

و هم‌چنین یادمان است که اگر در دستور range نقطه شروع را مشخص نمی‌کردیم، به‌صورت پیش‌فرض پایتون همان صفر را در نظر می‌گرفت؛ پس می‌توانیم بنویسیم:


```
for i in range(len(seq)):
```

و به عبارتی کل برنامه ما شد:

```
seq = 'ATCAGC'
count = 0
for i in range(len(seq)):
    if seq[i] == 'C':
        count += 1
print(f'The count of C in your DNA is {count}')
```

دانشجو: چه باحال! در این صورت اگر حتی توالی seq رو عوض کنیم، باز هم برنامه ما کار می‌کنه و با خطا روبرو نمی‌شه.

استاد: دقیقاً! این ساختار

```
range(len(seq))
```

را خیلی جاه‌در برنامه‌های پایتون می‌بینیم.

حالا یک سؤال! ما نوشتیم:

```
seq = 'ATCAGC'
for i in range(len(seq)):
    print(seq[i])
```

و در این حلقه i هر بار یک عدد بود و به همین دلیل یکی از حروف رشته ما می‌شد. اگر مستقیماً می‌نوشتیم:

```
seq = 'ATCAGC'
for i in seq:
    print(i)
```

هر بار i در حلقه چه می‌شود؟

دانشجو: امتحان کردم؛ `i` هر بار یکی از حروف رشته ما می‌شد!

استاد: درود بر تو! به این شکل هم می‌توان از حلقه برای رشته‌ها استفاده کرد و هر بار `i` به جای یک عدد، یکی از حروف رشته ما می‌شود؛ به همین دلیل بهتر است به جای `i` از نام متغیر دیگری استفاده کنیم، مثلاً `letter` که نمایانگر حرف باشد:

```
seq = 'ATCAGC'
for letter in seq:
    print(letter)
```

دانشجو: استاد بهتر نیست به جای روش قبل از همین روش استفاده کنیم؟

استاد: من هم موافقم چون این روش خوانایی بیشتری دارد؛ اما زمان‌هایی هم وجود دارد که ما علاوه بر حرف، به اندیس هم نیاز داریم. با یک مثال بهتر موضوع جا می‌افتد. فرض کنید ما دو رشته DNA به اسم `seq1` و `seq2` داریم:

```
seq1 = 'ATCA'
seq2 = 'AGCA'
```

همان‌طور که می‌بینید، این دو رشته هم‌اندازه (هر دو چهار حرفی بوده) و عین هم هستند و فقط در کاراکتر دوم باهم تفاوت دارند؛ به عبارتی از چهار حرف، در سه حرف شبیه هم هستند و به عبارت دیگر ۷۵٪ شباهت دارند. ما می‌خواهیم برنامه‌ای بنویسیم که شباهت دو رشته هم‌اندازه را برای ما به درصد حساب کند. چه کنیم؟

دانشجو: استاد دیگه این بار خودم میگم! اول می‌بینیم که خودمون به عنوان یه انسان، چطور مرحله به مرحله این موضوع رو انجام می‌دیم. بقیه رو نمی‌دونم اما من به‌شخصه از حرف اول رشته اول شروع می‌کنم و با حرف اول رشته دیگه مقایسه می‌کنم. اگه یکسان بود... آهان کیسه یادم افتاد...؛ از ابتدا یه کیسه شباهت‌ها کنار می‌ذارم که مقدار اون رو در ابتدا صفر در نظر می‌گیرم و هر بار که شباهت وجود داشت،

یه سنگ به این کیسه اضافه می‌کنم و در نهایت تعداد شباهت و تعداد کل حرف یک رشته رو که دارم، درصد رو حساب می‌کنم.

استاد: بسیار عالی! پس در مرحله اول نیاز داریم که به حروف رشته اول دسترسی داشته باشیم:

```
count = 0
for i in range(len(seq1)):
    seq1[i]
```

و اگر این حرف با حرف در رشته دوم برابر بود، یکی به کیسه count اضافه کنیم، یعنی:

```
count = 0
for i in range(len(seq1)):
    if seq1[i] == seq2[i]:
        count += 1
```

و در نهایت مقدار count را تقسیم بر تعداد حروف کنیم (دیدیم می‌توانیم با دستور len به تعداد حروف دسترسی پیدا کنیم) تا درصد شباهت به دست بیاید:

```
print('Similarity is', count/len(seq1))
```

البته بهتر است نتیجه را در ۱۰۰ ضرب کنیم که نتیجه را به درصد داشته باشیم؛ یعنی دستور بالا به این شکل دربیاید:

```
print('Similarity is', count/len(seq1)*100)
```

به عبارتی کل برنامه ما شد:

```
seq1 = 'ATCA'
seq2 = 'AGCA'
```

```
count = 0
```

```
for i in range(len(seq1)):
    if seq1[i] == seq2[i]:
        count += 1
```

```
print('Similarity is', count/len(seq1)*100)
```

دانشجو: استاد برای این که من متوجه بشم که درست فهمیدم، می‌شه یه بار برنامه رو باهم ردگیری کنیم؟

استاد: چرا که نه! خودت را جای کامپیوتر بگذار و خط به خط جلو برو.

دانشجو: بله حتماً. بعد از این خطوط:

```
seq1 = 'ATCA'
seq2 = 'AGCA'
count = 0
```

کامپیوتر به خط اول، یعنی شروع حلقه می‌رسه:

```
for i in range(len(seq1)):
```

و می‌دونیم که دستور `len(seq1)` همون تعداد حروف `seq1` رو حساب می‌کنه که می‌دونیم ۴ حرفه؛ پس انگار دستور بالا معادل زیره:

```
for i in range(4):
```

و در درس قبل یاد گرفتیم (بالاخره ☺) که یه متغیر به اسم `i` تعریف شده و چون مقدار شروع در `range` مشخص نشده، از صفر شروع می‌شه، یعنی مقدار `i` برابر با ۰ می‌شه. حال به درون حلقه وارد می‌شیم:

```
if seq1[i] == seq2[i]:
```

چون `i` برابر با صفره انگار دستور بالا معادل زیره:

```
if seq1[0] == seq2[0]:
```

`seq1[0]` برابر با عضو صفرم (می‌دونیم عضو صفرم در پایتون همون عضو اوله) در `seq1` است که برابر با حرف 'A' است و `seq2[0]` هم عضو صفرم در `seq2` است که می‌شه حرف 'A' و انگار دستور بالا معادل زیره:

```
if 'A'=='A':
```

و این شرط برقراره؛ پس وارد دستور داخل شرط می‌شیم:

```
count += 1
```

پس یکی به مقدار `count` اضافه می‌شه و مقدار `count` که قبلاً ۰ بود، برابر با ۱ می‌شه. حالا که به خط آخر داخل حلقه رسیدیم کامپیوتر به خط اول حلقه برمی‌گرده:

```
for i in range(len(seq1)):
```

و اکنون `i` برابر با ۱ می‌شه و تمامی مراحل قبل تکرار می‌شن؛ منتها این‌بار چون `i` برابر با ۱ است، `seq1[1]` و `seq2[1]` یعنی عضو دوم رشته‌ها بررسی می‌شن و همین‌طور تا انتهای رشته پیش می‌ریم.

درنهایت هم که حلقه تموم شد، به این خط می‌رسیم و چون هم تعداد شباهت‌ها رو داریم، تعداد حرف رشته رو هم که می‌توانیم با دستور `len` حساب کنیم:

```
print('Similarity is', count/len(seq1)*100)
```

چون از ۴ حرف، ۳ حرف یکسان بوده، پس مقدار `count` برابر با ۳ می‌شه و درنتیجه روی صفحه برای مخاطب چاپ می‌شه:

```
Similarity is 0.75
```

حال کردین استاد چقدر خفن تحلیل کردم! 😊

استاد: ممنون که پیشنهاد ردگیری برنامه را دادی تا بهتر موضوع جابی‌افتد.

حال یک سؤال! در این مثال از همان روش اول حلقه برای دسترسی به کاراکترهای موجود در یک رشته استفاده کردیم. حال اگر می‌آمدیم و از روش دوم حلقه برای رشته استفاده می‌کردیم، یعنی مستقیماً می‌نوشتیم:

```
for letter in seq1:
```

در این صورت تنها در هر بار حلقه به یکی از کاراکترها دسترسی داشتیم و سخت‌تر می‌شد که بتوانیم کاراکتر نظیر در رشته دوم را پیدا و مقایسه کنیم. به همین دلیل بهتر بود از همان روش نازیبای `range(len(seq1))` استفاده کنیم یعنی:

```
for i in range(len(seq1)):
    seq1[i]
```

دانشجو: عجب... انگار نمی‌شه هم اندیس و هم کاراکتر رو مستقیماً در هر بار حلقه دریافت کرد...!

استاد: یک راه دیگر وجود دارد که اگر تمایل داشتید به عنوان اطلاعات بیشتر بدانید، می‌توانید استفاده کنید. دستوری به نام `enumerate` در پایتون وجود دارد که می‌توان این جور کدها را زیباتر نوشت. `enumerate` در لغت به معنای شمردن است. کافی است بنویسیم:

```
enumerate(seq)
```

و حلقه خود را به این شکل بنویسیم:

```
for i, letter in enumerate(seq):
```

و به این شکل در هر بار حلقه متغیر `i` شماره اندیس بوده و `letter` کاراکتر است، مثلاً

```
seq = 'ATCA'
```

```
for i, letter in enumerate(seq):
    print(i, letter)
```

خروجی برنامه بالا به شکل زیر است:

```
0 A
1 T
2 C
3 A
```

و جالبی ماجرا اینجاست که این دستور یک ورودی اختیاری هم دارد که اگر آن را وارد کنید، شمارش را به جای صفر از آن عدد شروع می کند؛ مثلاً عدد ۱:

```
seq = 'ATCA'
for i, letter in enumerate(seq, 1):
    print(i, letter)
```

```
1 A
2 T
3 C
4 A
```

دانشجو: چه جالب! استاد می شه یه جمع بندی از این سه روش حلقه زدن روی کاراکتر رو انجام بدید.

استاد: بله حتماً! برای این که روی یه رشته حلقه بزنینم، در هربار حلقه سه روش داشتیم. روش اول:

```
seq = 'ATCA'
for i in range(len(seq)):
    print(i, seq[i])
```

خروجی کد:

که در این روش به شماره اندیس دسترسی داشتیم، در نتیجه می توانستیم به کاراکتر هم دسترسی داشته باشیم، اما مشکل آن این بود که کد زیبا نبود و از نظر

بصری پیچیده به نظر می‌آمد.

روش دوم: زمان‌هایی که با اندیس کاری نداریم و تنها به کاراکترها نیاز داریم، بهتر است مستقیماً بنویسیم:

```
seq = 'ATCA'
for letter in seq:
    print(letter)
```

خروجی کد:

```
A
T
C
A
```

ولی دیگر به اندیس دسترسی نداریم (یا حداقل باید کمی بیشتر برنامه بنویسیم تا به اندیس هم دسترسی پیدا کنیم).

روش سوم: که جزو بیشتر بدانید بود، هم از نظر بصری زیبا بود و هم کمک می‌کرد هم‌زمان هم به کاراکتر و هم به اندیس دسترسی داشته باشیم:

```
seq = 'ATCA'
for i, letter in enumerate(seq):
    print(i, letter)
```

خروجی کد:

```
0 A
1 T
2 C
3 A
```

دانشجو: البته روش سوم چندان هم زیبا نبود ولی قبول دارم از روش اول زیباتر بود!
استاد: از دست تو!

تمرین‌های جلسه چهارم

در تمرین هر درس ممکن است که یک نکته جدید هم آموزش داده شود، پس حتماً توصیه می‌کنم آن‌ها را ببینید و انجامشان دهید. تمرین‌ها و به خصوص خطاهایی که دریافت می‌کنید و آزمون و خطاها (و همچنین سرچ‌هایی که در صورت نیاز انجام می‌دهید)، بسیار بسیار به روند یادگیری برنامه نویسی شما کمک می‌کند. پاسخ نمونه برای تمرین‌ها نیز پس از آن‌ها قرار داده شده است.

تمرین‌ها

۱- برنامه‌ای بنویسید که یک توالی DNA را از مخاطب گرفته و درصد G و C را برای مخاطب به دست آورد (برای مثال توالی ATCTG مجموعاً ۲ تا از ۵ حرف G یا C دارد به عبارتی ۴۰ درصد GC content دارد).

۲- با سرچ کردن یا آزمون و خطا ببینید متد `ljust` و `rjust` برای رشته‌ها چه می‌کند. 😊

۳- یک برنامه بنویسید که جنسیت فرد و فشارخون او را پرسیده و در صورتی که فشارخون او بالای ۱۶ بود، اگر آقا بود آزمایش ۱ و اگر خانم بود آزمایش ۲ را توصیه کند و در صورتی که فشارخون بالای ۱۶ نبود، اگر آقا بود آزمایش ۳ و اگر خانم بود آزمایش ۴ را برای پیشگیری بیشتر توصیه کند.

نکته: برنامه را طوری بنویسید که اگر کاربر جنسیت را به اشتباه با حروف بزرگ یا کوچک یا ... وارد کرد، درست کار کند. برای مثال اگر `Male` یا `male` یا `maLe` یا ... وارد کرد، برنامه کماکان درست عمل کند.

۴- چگونه می‌توان در پایتون به مدت چند ثانیه مکث کرد؟ با جستجو کردن در اینترنت، دستور مربوطه را پیدا کنید و از آن یک مثال بنزید!

پاسخ تمرین‌های جلسهٔ چهارم

اکیداً توصیه می‌کنم که اول خودتان تمرین‌ها را حل کنید و سپس به پاسخ‌ها مراجعه کنید. در ضمن این پاسخ‌ها تنها یکی از حالت‌های ممکن نوشتن برنامه هستند و شما ممکن است به شیوه‌ای دیگر برنامه را نوشته باشید و کاملاً هم صحیح باشد. همچنین برای مشاهدهٔ توضیحات شفاهی مربوط به این تمرین‌ها، ابتدا ویدیوی جلسهٔ بعد را ببینید (لینک ویدیو در صفحهٔ اول کتاب گذاشته شده است).

« پاسخ تمرین اول:

```
seq = input('Please enter your DNA sequence:')
seq = seq.upper()
```

```
gc_count = 0
for nu in seq:
    if nu=='C' or nu=='G':
        gc_count += 1
```

```
gc_content = gc_count/len(seq)*100
print(f'your CG content is {gc_content} percent')
```

پرسش تکمیلی:

هدف از این خط در برنامهٔ بالا چیست؟

```
seq = seq.upper()
```

پاسخ:

ممکن است مخاطب حروف رشتهٔ DNA را با حروف کوچک وارد کند. با این خط

رشته ما هرچه که باشد، به حروف بزرگ تبدیل می‌شود و به همین دلیل در قسمت شرط‌ها برنامه ما درست کار می‌کند.

نکته تکمیلی:

راه زیباتر در پایتون به جای نوشتن شرط:

```
if nu=='C' or nu=='G'
```

به این صورت است:

```
if nu in ['C', 'G']:
```

یعنی می‌توان برنامه را به این شکل نوشت:

```
seq = input('Please enter your DNA sequence:')
```

```
seq = seq.upper()
```

```
gc_count = 0
```

```
for nu in seq:
```

```
    if nu=='C' or nu=='G':
```

```
        gc_count += 1
```

```
gc_content = gc_count/len(seq)*100
```

```
print(f'your CG content is {gc_content} percent')
```

یک مثال دیگر برای نوشتن شرط به این شکل: مثلاً می‌خواهیم برنامه‌ای بنویسیم که حروف رشته DNA داده شده را بررسی کند و اگر غیر از A و T و C و G بود، بنویسد که این نوکلئوتید ناشناخته است. می‌توانیم در اینجا از همان `not in` که در طی درس یاد گرفتیم استفاده کنیم یعنی:

```
seq = 'ATCXGXGT'
```

```

for nu in seq:
    if nu not in ['A', 'T', 'C', 'G']:
        print(f'Error! {nu} is not defined!')
        break
print('The program has finihed')

```

خروجی کد:

```

Error! The nucleotide X is not defined!
The program has finihed

```

دانشجو: استاد break چیه دیگه؟

استاد: break یعنی حلقه را بشکن و برو خطوط بعد از حلقه را اجرا کن! مثلاً در این مثال، اگر با یک کاراکتر غیر از A و T و C و G مواجه شویم، پیغام خطا نمایش داده شده و حلقه شکسته می‌شود و ادامه خطوط اجرا می‌شوند.

دانشجو: استاد در جایی دیدم در حلقه نوشته بود continue؛ اون به چه معناست؟

استاد: break به معنی این بود که کلاً حلقه را بی‌خیال شو و از حلقه بیرون بیا؛ اما continue به این معناست که این دور حلقه را بی‌خیال شو و برگرد به ابتدای حلقه. مثلاً در این برنامه که break داشت:

```

seq = 'ATCXGXGT'
for nu in seq:
    if nu not in ['A', 'T', 'C', 'G']:
        print(f'Error! {nu} is not defined!')
        break
print('The program has finihed')

```

دیدیم که به محض این که به کاراکتر تعریف نشده X رسیدیم، حلقه شکسته شد و

خطوط بعد از حلقه اجرا شد. به همین دلیل به کاراکتر X بعدی نرسیدیم:
Error! The nucleotide X is not defined!
The program has finihed

اما اگر همین برنامه را با continue بنویسیم یعنی:

```
seq = 'ATCXGXGT'  
for nu in seq:  
    if nu not in ['A', 'T', 'C', 'G']:  
        print(f'Error! The nucleotide {nu}')        continue  
print('The program has finihed')
```

هر وقت به کاراکتر تعریف نشده (مثلاً X) و دستور continue برسیم، به این معناست که حلقه بی خیال این دور شده و برگردد و سراغ کاراکتر بعدی برود. این دستور برای جلوگیری از خطاها بسیار مفید است. برای مثال شاید شما بخواهید یک محاسبه روی این کاراکتر انجام دهید و می خواهید اگر این کاراکتر تعریف شده نباشد، این محاسبه روی آن انجام نشود تا برنامه دچار خطا شده و متوقف شود؛ بلکه کامپیوتر این دور حلقه را بی خیال شده و به ابتدای حلقه برگردد و به سراغ کاراکتر بعدی برود.

دانشجو: عجب چیزی بود! الآن خیلی بهتر متوجه شدم؛ البته به نظر من هر چی جلوتر
بریم، در عمل بهتر درکش کنم.

استاد: بله حتماً!

نکته تکمیلی:

همین برنامه را می توان به جای استفاده از حلقه For با استفاده از متد count نوشت

یعنی:

```
seq = input('Please enter your DNA sequence:')
seq = seq.upper()

c_count = seq.upper().count('C')
g_count = seq.upper().count('G')

gc_content = (c_count+g_count)/len(seq)*100
print(f'your CG content is {gc_content} percent')
```

البته هدف ما این بوده است که در این تمرین، با کاربرد حلقه بر روی رشته‌ها مسلط‌تر شویم و گرنه به این روش هم می‌توانستیم برنامه را بنویسیم.

« پاسخ تمرین دوم:

اگر جستجو یا آزمون و خطا کنیم، می‌بینیم که این دو دستور کار جالبی انجام می‌دهند. این مثال را ببینید:

```
word = 'Hello'

right = word.rjust(10, '*')
left = word.ljust(10, '*')

print(right)
print(left)
```

خروجی کد:

```
*****Hello
Hello*****
```

« پاسخ تمرین سوم:

```
gender = input('Please input your gender:\n')
gender = gender.lower()

bp = input('Please input your blood pressure in cmHg:\n')
bp = float(bp)

if bp>16:
    if gender=='male':
        print('Test 1 is suggested')
    elif gender=='female':
        print('Test 2 is suggested')
else:
    if gender=='male':
        print('Test 3 is suggested')
    elif gender=='female':
        print('Test 4 is suggested')
```

توجه داشته باشید که `else` مربوط به شرط `if bp>16` است. در پایتون در بررسی شروط حتماً به دندانۀ متن (میزان جلو رفتن خطوط) دقت کنید. این مورد در روزهای اول باعث خطاهای زیادی شده اما به مرور زمان عادت می شود و اشتباهات خیلی کمتر خواهند شد (از جمله خودم هم روزهای اول به این شکل حواسم به این جلو رفتن ها نبود، اما به مرور عادت کردم 😊)

« پاسخ تمرین چهارم:

با هر ادبیاتی که بلد بودیم (یا حتی می‌توانیم از گوگل ترجمه استفاده کنیم و ببینیم مکث کردن به انگلیسی چه می‌شود)، من جستجو کردم:

How to pause in python for seconds?

یکی از دستورهای که برایم پیدا شد، دستور sleep از کتابخانه time است. همان‌طور که در درس آموختیم، می‌توانیم به دو شکل از دستورات کتابخانه‌ها استفاده کنیم:

```
import time
print('Hello')
time.sleep(3)
print('Pedram!')
```

و راه دیگر این‌که فقط همان دستور مدنظر، یعنی sleep را import کنیم:

```
from time import sleep
print('Hello')
sleep(3)
print('Pedram!')
```

و وقتی برنامه را اجرا می‌کنیم، کامپیوتر ۳ ثانیه مکث می‌کند و سپس کلمه Pedram را چاپ می‌کند.

• فصل پنجم •

لذت کار با داده‌های بزرگ‌تر با استفاده از لیست‌ها در پایتون!

لیست چیست و به چه دردی می‌خورد؟

سلام و درود دوستان! حالتان چطور است؟ تا الآن با متغیرها آشنا شدیم، دستورات شرطی را دیدیم و متوجه شدیم که با حلقه‌ها چگونه کارهای تکراری را انجام دهیم؛ با مفهوم جذاب و کاربردی کتابخانه‌ها و همچنین متدهای رشته‌ها آشنا شدیم و هر چه جلوتر برویم بیشتر با آن‌ها آشنا خواهیم شد. اگر موافق باشید اسپایدر را باز کنیم و برویم سراغ مبحث این جلسه!

دانشجو: سلام استاد! بریم!

برای این جلسه، فرض کنید که آزمایشی را روی چندین موش انجام داده‌اید و امتیاز این آزمون‌ها را می‌خواهید یادداشت کنید. چطور می‌توانیم این امتیازها را ذخیره کنیم؟ اگر بنویسیم:

`score = 10`

برای نمونه بعدی چه کنیم؟ اگر آن را هم در `score` ذخیره کنیم یعنی:

`score = 12`

که این عدد جدید جایگزین قبلی می شود، در حالی که من به هر دو عدد نیاز داشتم!
پس چه کنیم؟

دانشجو: استاد به نظرم بشه عدد جدید رو در متغیر دیگری مثلاً score2 ذخیره کرد:

```
score2 = 10
```

```
score2 = 12
```

استاد: این روش جواب می دهد، اما اگر ۱۰ تا امتیاز داشتیم باید چه کنیم؟ با این روش باید بنویسیم:

```
score1 = 11
```

```
score2 = 12
```

```
score3 = 13
```

```
score4 = 11
```

```
score5 = 17
```

```
score6 = 19
```

```
score7 = 21
```

```
score8 = 14
```

```
score9 = 10.5
```

```
score10 = 11
```

دانشجو: نه این طوری که خیلی وقت گیر می شه!

استاد: دقیقاً! و فرض کنید می خواهیم یک نمونه بین score4 و score5 اضافه کنیم. آن وقت باید همه نام ها را تغییر دهیم! از طرفی اگر بخواهیم مثلاً میانگین این اعداد را حساب کنیم، به راحتی نمی توانیم!

در این مواقع که با تعداد داده های بیشتر کار داریم، به سراغ ساختار داده ها می رویم! متغیرهایی که بیشتر از یک داده را می توان در آن ها ذخیره کرد. در این درس به سراغ

یکی از مهم‌ترین ساختار داده‌ها در پایتون می‌رویم. ساختار داده‌ی عزیزی به نام لیست! برای این کار کافی است یک لیست تعریف کنیم، مثلاً چون می‌خواهیم امتیازها را در آن ذخیره کنیم به نظر نام `scores` مناسب باشد:

```
scores = [10, 12, 14, 13, 17]
```

همان‌طور که می‌بینید، برای تعریف یک لیست کافی است دو علامت کروشه بگذاریم و با کاما اجزای آن را مشخص کنیم.

تبریک می‌گوییم! شما الان یک لیست دارید که ۵ عضو دارد. حالا چگونه می‌توانم به اولین عضو این لیست دسترسی داشته باشم؟ کافی است بنویسم:

```
scores[0]
```

دانشجو: چه جالب! دقیقاً مثل رشته‌ها!

استاد: دقیقاً! موافقی یک لیست تعریف کنی و برنامه را اجرا کنی؟

دانشجو: بله حتماً استاد! برای این که راحت بتوانم آزمون و خطا کنم، در همون قسمت کنسول یعنی همون قسمت پایین سمت راست اسپایدر، یه لیست به اسم `scores` شبیه خودتون تعریف کردم و برنامه رو اجرا کردم:

```
scores = [10, 12, 14, 13, 17]
```

```
scores[0]
```

جالبه که برام نوشت:

```
10
```

که همان عضو اوله!

استاد: حال اگر بنویسم، چه چیزی نمایش داده می‌شود؟

```
scores[0:2]
```

دانشجو: برام عضو اول و دوم را برگردوند (یا به قول پایتون عضو صفرم و اول). یعنی:
[10, 12]

الآن این خودش یک لیسته استاد درسته؟ چون داخل کروشه اس.

استاد: دقیقاً! یک لیست از عضو اول و دوم! یادت است با کدام دستور می توانستیم
نوع متغیر را ببینیم؟

دانشجو: نوک زبونه!...! الآن بهتون می گم...! ای بابا یادم رفته! ولی اشکال نداره، الآن
در گوگل جستجو می کنم... آهان با دستور type:

```
type(scores)
```

که برام نوشت:

```
list
```

استاد: درود بر تو! و حاصل دستور زیر چه چیزی است؟

```
type(scores[0:2])
```

دانشجو: چه جالب این هم شد لیست، چون دیدیم که خروجی دستور:

```
scores[0:2]
```

خودش یک لیسته یعنی:

```
[10, 12]
```

استاد: بسیار عالی! و اگر بخواهم به عضو آخر لیست دسترسی داشته باشم چطور؟
دقیقاً مثل رشته ها کافی است بنویسیم:

```
scores[-1]
```

و اگر بخواهیم به عضو یکی مانده به آخر دسترسی پیدا کنیم چطور؟

دانشجو: وقتی -۱ می شه آخرین عضو، احتمالاً -۲ می شه یکی مونده به آخری، یعنی:

`scores[-2]`

بذارین امتحان کنم! آره درست گفتم!

استاد: بسیار عالی! حالا اگر بخواهیم در لیستی که داشتیم، اولین عضو به جای ۱۰، ۳۰ باشد. باید چه کار کنیم؟

دانشجو: با دستور زیر می‌توانستیم به عضو دسترسی پیدا کنیم:

`scores[0]`

پس احتمالاً بشود با دستور زیر مقدار آن را تغییر داد:

`scores[0] = 30`

بذارین الان خود لیست `scores` را چک کنم:

`scores`

چه جالب عضو اول لیستم تغییر کرد و شد:

`[30, 12, 14, 13, 17]`

استاد: واقعاً لذت می‌برم که با آزمون و خطا سعی می‌کنی که راهکار را پیدا کنی. حال چگونه می‌توانیم کاری کنیم که به عضو دوم این لیست، ۱۰۰ واحد افزوده شود؟

دانشجو: با دستور زیر می‌توانستیم به مقدار عضو دوم دسترسی پیدا کنیم (حواسم هست استاد که در پایتون شمارش از ۰ شروع می‌شه 😊)

`scores[1]`

و قاعدتاً با دستور زیر می‌توان ۱۰۰ تا به این عضو اضافه کرد:

`scores[1] + 100`

استاد: الان خود لیست `scores` را چک کن.

دانشجو: الان چک می‌کنم:

scores

استاد مقدار عضو دوم هیچ فرقی نکرده!!

استاد: به نظرت دلیلش این نیست که ما به پایتون نگفتیم که عضو دوم این لیست را برابر این مقدار بگذارد؟ یعنی:

```
scores[1] = scores[1] + 100
```

دانشجو: آهان درسته! دقیقاً شبیه چیزی که در جلسات قبل متغیرهای تک عددی داشتیم! الآن مقدار scores رو چک کردم دیدم که ۱۰۰ واحد به عضو دوم اضافه شده یعنی:

```
[10, 112, 14, 13, 17]
```

استاد: درود بر تو! به عنوان یک نکته تکمیلی می توانیم این دستور یعنی:

```
scores[1] = scores[1] + 100
```

را همانند جلسات قبل به صورت زیر بنویسیم:

```
scores[1] += 100
```

افزایش کارایی بامتدهای لیست ها!

همانند رشته ها، لیست ها هم متد دارند. برای این که بتوانیم آزمون و خطا کنیم تا این متدها را بهتر ببینیم، بریم در قسمت کنسول اسپایدر، یک لیست تعریف کنیم. راستی کنسول کجا بود؟

دانشجو: استاد باور کنین یاد گرفتیم! پایین سمت راست اسپایدر! که می شد دستورات رو تک خطی اجرا کرد.

استاد: درود بر تو! حال ناراحت نباش و به جای آن یک لیست تعریف کن؛ مثلاً

numbers:

```
numbers = [10, 12, 14]
```

حالا شما که این قدر خوب یادت است، با کدام علامت می‌توانستیم به متدهای یک‌چیز دسترسی داشته باشیم؟

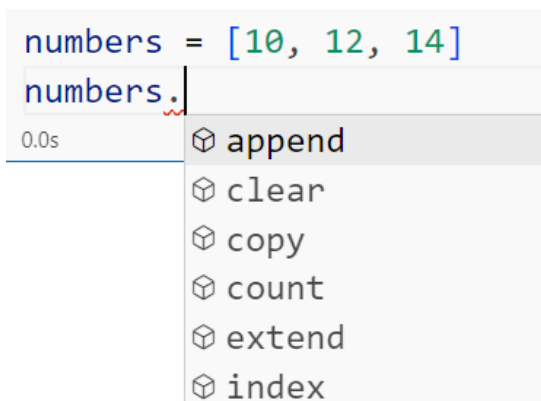
دانشجو:... این رو یادم رفته... نوک زبونمه...

استاد: علامممتتت...؟؟

دانشجو: آهان نقطه!

`numbers.`

استاد: دقیقاً! که در خیلی از محیط‌های کدنویسی پایتون، بعد از نوشتن نقطه و زدن کلید Tab روی کیبورد (البته در بعضی از محیط‌های کدنویسی کلید `Cntrl+Space` همین کار را می‌کند)، خود پایتون راهنمایی کرده و متدها را به ما نشان می‌دهد (شکل ۱):



شکل ۱- معمولاً خود محیط کدنویسی پایتون بعد از زدن نقطه، به ما راهنمایی می‌کند.

حالا برویم به سراغ اولین متد! متد `append`. مثل همیشه با یک مثال، دستور زیر را امتحان کن و ببین چه کاری با لیست ما می کند:

```
numbers.append(5)
```

دانشجو: امتحان کردم، چیزی ننوشت! بذارین در همون کنسول بنویسم:

```
numbers
```

ببینیم الان محتوای `numbers` چیه! آهان! عدد ۵ که نوشتیم رو به آخر لیست اضافه کرد:

```
[10, 12, 14, 5]
```

یعنی اگه مثلاً بنویسم:

```
numbers.append(8)
```

ببینم مقدار لیست من الان چقدره:

```
print(numbers)
```

چه باحال عدد ۸ هم به لیستم اضافه شد:

```
[10, 12, 14, 5, 8]
```

استاد: حالا به نظرت متد `insert` چه می کند؟ نکته جالب بسیاری از محیط های کدنویسی پایتون این است که همین که پرانتز را بعد از دستور تایپ می کنیم، `help` آن دستور برای ما به عنوان راهنمایی ظاهر می شود (شکل ۲):

```
>>> numbers.insert(0, 1)
(index: SupportsIndex, object: int, /)
-> None

numbers.insert()
```

MagicPyt

شکل ۲- معمولاً در محیط های کدنویسی پایتون، همراه با تایپ کردن دستور، راهنمایی آن دستور نیز ظاهر می شود.

و البته یک راه دیگر هم داشتیم که بتوانیم help یک دستور را ببینیم! آن‌هم استفاده از خود دستور help است:

```
help(numbers.insert)
```

که برای ما نوشت:

Help on built-in function insert:

```
insert(index, object, /) method of builtins.list instance  
Insert object before index.
```

و درنهایت هم اگر با هیچ کدام جواب نگرفتیم، راه همیشگی، جستجو در گوگل! خب ببینیم این help دارد به ما چه می‌گوید. اول آن‌که این دستور چند ورودی می‌گیرد؟

دانشجو: نوشته:

```
insert(index, object, /)
```

استاد: دقیقاً! یعنی این تابع دو ورودی نیاز دارد (یا اصطلاحاً این تابع، دو آرگومان نیاز دارد). حال، آیا باید حتماً این دو آرگومان وارد شوند یا اگر وارد نشوند، پایتون می‌تواند ایراد نگیرد و به‌صورت پیش‌فرض در نظر بگیرد؟

دانشجو: استاد، توی دستورات جلسه قبل، در help آن‌ها یک علامت مساوی بعد از بعضی از این آرگومان‌ها نوشته شده بود (مثلاً، 'end=' که یادمه گفتید به معنی اینه که اگه ما وارد نکنیم، پایتون خودش به شکل پیش‌فرض مقدار اون رو در نظر می‌گیره. اینجا نه آرگومان اول (index) و نه آرگومان دوم (object) علامت مساوی رو بعد از خودشون ندارند.

استاد: دقیقاً! و این دستور درنهایت چه می‌کند؟ در خط آخر help نوشته شده:

Insert object before index.

و همان طور که خودت گفتی این دستور index و object ما را به عنوان ورودی می گیرد؛ پس همان طوری که نوشته یک چیز را به اندیس مورد نظر ما اضافه می کند.

دانشجو: استاد کاملاً می فهمم اما متوجه نمی شم!! یه مثال پیلز!

استاد: صد در صد! همان لیست قبلی خودمان را در نظر بگیریم:

```
numbers = [10, 12, 14]
```

این دستور را اجرا کن و نتیجه را به من بگو!

```
numbers.insert(0, 56)
```

دانشجو: مثل دستور قبل چیزی به من نشون نداد. بذارین مقدار جدید لیست رو

ببینم:

```
numbers
```

و نتیجه:

```
[56, 10, 12, 14]
```

انگار عدد ۵۶ رو ... به جایگاه اول لیست تزریق کرده!! آهان!!! پس اولین آرگومان، درواقع اندیسی بود که می خواستیم اون چیزی که می خوایم اضافه بشه.

استاد: دقیقاً! و حالا برنامه زیر چه می کند؟

```
numbers = [10, 12, 14]
```

```
numbers.insert(1, 56)
```

```
print(numbers)
```

دانشجو: اون جوری که طبق راهنمای دستور insert یاد گرفتم، اولی اندیس و دومی چیزیه که می خوایم اضافه کنیم. پس قاعدتاً این دستور عدد ۵۶ رو جای عضو

... عضو دوم تزریق می‌کنه (چون در پایتون اندیس ۱ یعنی عضو دوم) پس خروجی باید بشه:

```
[10, 56, 12, 14]
```

استاد: دقیقاً! عدد ۵۶ را قرار داد و بقیه را یکی به جلو هل داد! حال دستور زیر چطور؟

```
numbers.pop(1)
```

دانشجو: چه جالب! چک کردم عضو دوم (همون اندیس ۱ از نظر پایتون) رو پاک کرد! استاد: یا این متد چطور؟

```
scores = [1, 1, 1, 0, 0, 2, 2, 3, 1]
```

```
scores.count(1)
```

دانشجو: نوشت ۴! از اسمش به نظرم اومد و تعداد عضو ۱ رو در لیست ما شمرد!

استاد: دقیقاً! و متد `index`:

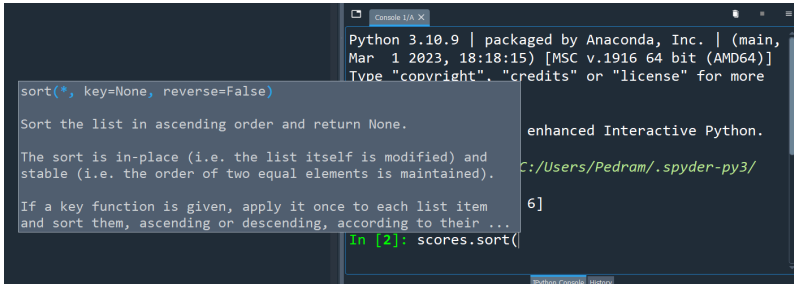
```
scores = [1, 1, 1, 0, 0, 2, 2, 3, 1]
```

```
scores.index(0)
```

جایی که اولین بار عضو مدنظر ما در لیست ظاهر شده را به ما می‌گوید، که اینجا عدد صفر برای اولین بار در اندیس سوم ظاهر شده است.

حال می‌خواهم متد `sort` را شما به من درس بدهید! به نظر تان این متد چه کار می‌کند؟

دانشجو: من وقتی در کنسول نوشتم `scores.sort()` برای من این توضیحات اومد (شکل ۳):



```
sort(*, key=None, reverse=False)

Sort the list in ascending order and return None.

The sort is in-place (i.e. the list itself is modified) and
stable (i.e. the order of two equal elements is maintained).

If a key function is given, apply it once to each list item
and sort them, ascending or descending, according to their ...

enhanced Interactive Python.
C:/Users/Pedram/.spyder-py3/
6]

In [2]: scores.sort()
```

شکل ۳- توضیحات پایتون برای متد sort

استاد: نوشته شده که این تابع، چه کاری انجام می دهد؟

دانشجو: نوشته شده لیست رو sort می کنه؛ به عبارتی مرتب می کنه.

استاد: و نوشته شده return None یعنی چیزی برنمی گرداند و در خط بعد نوشته شده:

The sort is in place (i.e the list itself is modified)

پس یعنی لیست ما تغییر می کند اما چیزی را برنمی گرداند.

دانشجو: استاد کماکان می فهمم ولی متوجه نمی شم!!!

استاد: حق داری! نظرت چیست که یک مثال بزنی؟

دانشجو: بله حتماً مثلاً من لیست زیر رو دارم:

```
scores = [4, 6, 5, 1, 2]
```

اگر بنویسم:

```
scores.sort()
```

و منطقاً چیزی لازم نیست در پرانتز بنویسیم زیرا هر آرگومانی که در توضیحات

دستور نوشته شده، بعد از آن علامت = گذاشته یعنی اگر من چیزی ندارم، خود پایتون به صورت پیش فرض چیزی قرار می‌ده. خب دستور رو در کنسول اجرا کردم... چیزی ننوشت (فکر کنم برای همین نوشته بود چیزی بر نمی‌گردونه)... بذارین لیست رو ببینم:

scores

آهان! الآن لیستم از کوچک به بزرگ مرتب شد!!

[1, 2, 4, 5, 6]

استاد: حالا اگر بخواهیم این مرتب کردن از بزرگ به کوچک باشد چطور؟ آیا در توضیحات دستور (شکل ۳) چیزی می‌بینی؟

دانشجو: یه آرگومان در تابع نوشته شده به نام reverse=False.

استاد: نظرت چیست که ما آن را برابر با True بگذاریم؟

scores.sort(reverse=True)

دانشجو: آهان! حالا فهمیدم! الآن امتحان کردم:

scores

دیدم محتوای لیست من از بزرگ به کوچک مرتب شده!

[6, 5, 4, 2, 1]

پس این آرگومان‌های اختیاری بعضی جاها به در بخور هستن!

استاد: صدالبته! راستی، در محیط اسپایدر اگر در جلوی دستور نوشته شده است که کلیک کنید و بعد کلید `Ctrl + i` را روی کیبورد فشار بدهید، راهنمای تابع برای آن دستور معمولاً ظاهر می‌شود.

برویم در رابطه با لیست‌ها یک نکته دیگر را یاد بگیریم. به نظر شما خروجی دستور

زیر چیست؟

5 in scores

دانشجو: نوشت True! چه جالب این دستور رو در رابطه با رشته‌ها هم یادمه که می‌تونستیم استفاده کنیم.

استاد: دقیقاً! و اگر خواستیم برعکس، نبود یک عضو در یک لیست را چک کنیم می‌توانیم همان‌طور که در رشته‌ها دیدیم:

20 not in scores

که چون عدد ۲۰ در لیست ما حضور نداشت، بنابراین True برگردانده شد.

دانشجو: چی شد استاد؟

استاد: قبول داری چون عضو ۲۰ در لیست ما حضور ندارد خروجی زیر می‌شود
False:

20 in scores

دانشجو: آهان! و چون با not منفی کردیم پس شده True. استاد در این متدها یه متد دیدم به نام extend. اون چیکار می‌کنه و فرقی با append چیه؟
استاد: این دو تا دستور را امتحان کن:

```
scores.append(100)
```

و این دستور را:

```
scores.append([99, 100])
```

دانشجو: بذارین امتحان کنم. بعد از دستور اول، لیست ما شد:

```
[4, 6, 5, 1, 2, 100]
```

حالا دستور دوم رو امتحان کنم... لیست ما شد:

[4, 6, 5, 1, 2, 99, 100]

استاد: همون‌طور که خودت هم دیدی `append` یک عضو را به لیست ما اضافه می‌کند، اما `extend` محتوای یک لیست دیگر را به انتهای لیست ما اضافه می‌کند.

برویم سراغ یک متد مهم دیگر! متد `copy`! فرض کنید لیستی مهم داریم که می‌خواهیم یک نسخه پشتیبان از آن داشته باشیم که اگر تغییرش دادیم، نسخه پشتیبان ما تغییر نکند؛ مثلاً لیستی مهم از فشارخون بیماران ما به نام `patients`:

```
patients = [12, 16, 10, 8]
```

اگر بخواهیم یک کپی از این لیست داشته باشیم چه کنیم؟

دانشجو: ... به نظرم یک لیست دیگه تعریف کنیم مثلاً به نام `backup`:

```
backup = patients
```

بذارین داخل `backup` رو الان چک کنیم:

```
backup
```

بله دقیقاً همون چیزی شد که می‌خواهیم؛ یعنی یک لیست جدید به نام `backup` تشکیل شد:

```
[12, 16, 10, 8]
```

استاد: حالا بیا یه لیست اول را تغییر بدهیم؛ مثلاً یک عضو به آن اضافه کنیم:

```
patients.append(20)
```

حال محتوای `patients` را که چک کنیم، می‌بینیم که عضو ۲۰ به آن اضافه شده است:

```
[12, 16, 10, 8, 20]
```

حال محتوای `backup` را چک کن.

دانشجو: استاد سؤالایی می‌پرسید!!! اصلاً `backup` رو ساختیم که وقتی اون یکی

لیست رو تغییر دادیم این تغییر نکنه! حالا به خاطر شما چشم! اینم برای این محتوای لیست backup رو ببینیم:

```
print(backup)
[12, 16, 10, 8, 20]
```

عه!!! اینم که تغییر کرد!!! آخه چرا؟؟

استاد: در پایتون خیلی حواستان باشد که وقتی از علامت مساوی استفاده می کنیم اصلاً یک کپی جدید ایجاد نمی شود؛ به عبارتی هردوی این لیست ها یک لیست هستند و به یک جا در حافظه اشاره می کنند. به عنوان اطلاعات بیشتر، دستوری داریم به نام id که می توانیم شماره حافظه ای که آن متغیر در پایتون دارد به آن اشاره می کند را ببینیم:

```
id(patients)
id(backup)
```

و می بینیم که در کمال تعجب id این دو لیست یکی هستند!! یعنی در واقع این دو لیست، یک لیست هستند:

```
2944856828928
2944856828928
```

دانشجو: پس استاد چیکار کنیم!!

استاد: که اگر بخواهیم واقعاً یک لیست جدید کپی از لیست قبلی تشکیل شود، از متد copy استفاده می کنیم:

```
backup = patients.copy()
```

در این صورت اگر مثلاً به لیست اول، عددی را اضافه کنیم:

```
patients.append(20)
```

می‌بینیم که محتوای لیست دیگر تغییری نمی‌کند:

```
print(backup)
```

خروجی کد:

```
[12, 16, 10, 8]
```

اگر شما با دستور `id`، شماره حافظه‌ای که این دو متغیر در پایتون به آن اشاره می‌کنند را ببینید:

```
print(id(patients))
```

```
print(id(backup))
```

مشاهده می‌کنید که دو مقدار متفاوت هستند؛ به عبارتی این دو لیست از همدیگر مستقل بوده و دو لیست مجزا هستند:

```
2944862541888
```

```
2944856355776
```

دانشجو: عجب!!!

استاد: من خودم اوایل کار این موضوع را فراموش می‌کردم و حواسم نبود که از متد کپی استفاده کنم و با کلی فکر کردن متوجه شدم علتی که برنامه من درست کار نمی‌کرده همین بوده است!!! 😊

یک متد دیگر را نیز باهم ببینیم! به نظرتان متد `clear` چه می‌کند؟

```
patients.clear()
```

دانشجو: بذارین مقدار `patients` رو ببینم:

```
patients
```

چه جالب یه لیست خالی شد! انگار کل محتوای لیست رو پاک کرد:

```
[]
```

پس یه لیست در پایتون می‌تونه خالی باشه!

استاد: دقیقاً! شما حتی می‌تونید لیست خالی هم در پایتون ایجاد کنید؛ مثلاً:

```
results = []
```

که با این کار یک لیست خالی به نام results ایجاد می‌شود که می‌تونیم بعداً در طی برنامه، آن را هر جور که می‌خواهیم پر کنیم!

چطور می‌تونیم ببینیم که لیست ما چند عضو دارد؟

دانشجو: سرچ کنم استاد؟

استاد: می‌توانی در گوگل جستجو کنی اما همین الان هم دستورش را بلد هستیم. اگر یک رشته داشتیم، چطور می‌توانستیم تعداد کاراکترهایش را بشماریم؟

دانشجو: آهان... فکر کنم دستور len بود که مخفف کلمه length بود یعنی اگر بنویسیم:

```
len(patients)
```

خروجی کد میشه... نوشت ۴ تا! این لیست هم ۴ عضو داشت!

استاد: بسیار عالی! حالا به نظرت دستور زیر چه می‌کند؟

```
sum(patients)
```

دانشجو: تا جایی که قبلاً یادمه به معنای جمع بود... پس احتمالاً باید جمع اجزای این لیست را حساب کنه! ... درسته همین کار رو هم کرد.

استاد: حالا به نظرت خروجی این دستور چیست؟

```
sum(patients)/len(patients)
```

دانشجو: جمع اجزای لیست ما تقسیم بر تعداد؟

استاد: که حاصلش چه می‌شود؟

دانشجو: همین می‌شه دیگه!

استاد: نه منظورم این است که اگر مجموع را تقسیم بر تعداد کل کنیم، به حاصل می‌گفتیم میا...؟!

دانشجو: میانگین! آهان گرفتم! پس مثلاً می‌تونیم حاصل رو در یه متغیر به اسم mean بریزیم و بعداً از اون استفاده کنیم!

```
patients = [12, 16, 10, 8]
mean = sum(patients)/len(patients)
print(mean)
```

استاد بسیار عالی!

یک مثال از قبل با لیست‌ها!

استاد: قبل از این که مثال‌های کاربردی‌تر را باهم تمرین کنیم، با این مثال یکی از مهم‌ترین کاربرد لیست‌ها را بهتر جا بیندازیم. در درس حلقه‌ها، یادتان است برنامه‌ای نوشتیم که تمامی اعداد بخش‌پذیر بر ۳ کوچک‌تر از عدد ۱۰۰۰ را به ما نمایش می‌داد؟

دانشجو: آره استاد! با حلقه نوشتیم و در آخر اگه باقیمانده هر عدد بر ۳ صفر بود، اون عدد رو نمایش می‌دادیم. بذارین بنویسیمش که ببینین چقدر من خفنم!

```
for num in range(1, 1000):
    if num%3==0:
        print(num)
```

که خیلی قشنگ تمامی اعداد مضرب ۳ کوچک‌تر از عدد را نشان می‌داد.

استاد: بسیار عالی! حالا اگر بخواهی این اعداد را به من مجدداً نشان دهی باید چه کار کنی؟ این اعداد کجا ذخیره شده‌اند؟

دانشجو: ... راستش... فقط پرینت شدن... جایی ذخیره نشدن... آگه بخوام دوباره بهتون نشونشون بدم باید دوباره برنامه رو اجرا کنم!

استاد: دقیقاً! ما در بسیاری از مواقع می‌خواهیم نتیجه را جایی نگه داریم و از آن استفاده کنیم و ...

یک راه ساده برای این کار، استفاده از یک لیست است. یک لیست خالی که هر موقع عددی مضرب ۳ را پیدا کردیم به لیستمان اضافه کنیم. به نظرت نام لیست را چه بگذاریم؟

دانشجو: mezrabs چگونه استاد؟ همون مضرب خودمون!

استاد: از دست تو! باشه همین را می‌گذاریم:

```
mezrabs = []
```

این یک لیست خالی به نام mezrabs ایجاد می‌کند. حال هر زمان به عددی بخش‌پذیر بر ۳ رسیدیم، کافی است آن عدد را به لیستمان اضافه کنیم. یادت است که چگونه می‌توانستیم یک چیز را به لیستمان اضافه کنیم؟

دانشجو: ... dneppa! آره با متد append می‌شد!

استاد: بسیار عالی یعنی به جای دستور:

```
print(num)
```

که الان نیازی به آن نداریم، کافی است بنویسیم:

```
mezrabs.append(num)
```

پس برنامه ما می‌شود:

```
mezrabs = []  
for num in range(1, 1000):  
    if num%3==0:  
        mezrabs.append(num)
```

الآن لیستی به نام mezrabs داریم که می‌توانیم با آن کارهای بسیار زیادی انجام دهیم که در ادامه خواهیم داد، اما برای این که خالی از عریضه نباشد، می‌توانیم در نهایت این لیست را چاپ کنیم یعنی:

```
print(mezrabs)
```

راستی چطور می‌توانیم ببینیم چند عدد درون این لیست وجود دارد و به کاربر نشان دهیم؟

دانشجو: دستور ... اسمش رو یادم رفته استاد...

استاد: یادت نرود که می‌توانی جستجو کنی!

دانشجو: الآن سرچ می‌کنم؛ how to find out numbers of the list in python...
آهان دستور len یادم اومد!

```
len(numbers)
```

یعنی برنامه ما می‌شود:

```
mezrabs = []  
for num in range(1, 1000):  
    if num%3==0:  
        mezrabs.append(num)  
  
print(mezrabs)
```


`len(mezrabs)`

بذارین امتحان کنم؛ استاد، لیست مضرب‌ها رو نشون داد ولی تعداد رو نشون نداد، چرا؟

استاد: حواست باشد که الآن در محیط کنسول نیستیم که دستورات را خط به خط می‌نوشتیم و اجرا می‌کردیم؛ الآن در حال نوشتن تمام برنامه هستیم و هر جا که می‌خواهیم چیزی نمایش داده شود یا اصطلاحاً برای مخاطب روی صفحه چاپ شود باید از دستور ...

دانشجو: پرینت! از دستور پرینت استفاده کنیم! آهان من یادم رفته بود یعنی به‌جای:

`len(mezrabs)`

باید بنویسم:

`print(len(mezrabs))`

الآن درست شد! در آخر نوشت:

333

استاد: نظرت چیست که به مخاطب هم راهنمایی کنیم و همین‌جوری یک عدد را ننویسیم؛ مثلاً بنویسیم:

`print('The count of numbers is:', len(mezrabs))`

که خروجی می‌شود:

The count of numbers is: 333

دانشجو: ... خب آره استاد این‌طوری یه کمی بهتره 😊

استاد:...

دانشجو: 😊 😊

استاد:

دانشجو: 😊😊😊

.....

----- بعد از مدتی طولانی!!! -----

استاد: که یک کمی ... خب!

دانشجو: دیگه استاد! برنامه‌نویسی سخته؛ اون رو سخت‌تر نکنیم! 😊

استاد: از دست تو!!!

مثال بیماری‌ها!

بریم و با یک مثال کاربردی‌تر، لیست‌ها را بهتر به کار بگیریم! فرض کنید لیستی از شهرها داریم، مثلاً:

```
cities = ['Shiraz', 'Tehran', 'Esfahan', 'Bojnourd']
```

دانشجو: مگه میشه در لیست‌ها، رشته هم ذخیره کرد؟ من فکر می‌کردم فقط میشه عدد داخلشون ذخیره کرد!

استاد: بله؛ چرا که نه! لیست‌ها در اصطلاح ساختار داده‌های هتروژن هستند و به عبارتی حتی می‌توانند ترکیبی از عدد و رشته و ... داشته باشند، مثلاً لیست زیر:

```
examples = ['Pedram', 4, 'Ali', 3, 6]
```

و یا حتی اعضای آن‌ها می‌توانند خودشان یک لیست باشند!

```
examples = ['Pedram', 4, 'Ali', [1, 2, 3]]
```

که می‌بینید عضو آخر آن خودش یک لیست است! با این لیست‌های تودرتو بعداً

بیشتر کار خواهیم کرد.

دانشجو: استاد مثلاً همون لیست cities که گفتید:

```
cities = ['Shiraz', 'Tehran', 'Esfahan', 'Bojnourd']
```

اگر من روی این لیست حلقه بزنم مثلاً:

```
for i in cities:  
    print(i)
```

در هر بار حلقه به یکی از این اعضا، یعنی به یه شهر دسترسی پیدا می کنم، درسته؟

استاد: دقیقاً! به همین دلیل بهتر است که نام بهتری برای متغیر حلقه انتخاب کنیم، مثلاً city:

```
for city in cities:  
    print(city)
```

دانشجو: آهان! بعد استاد تمام متدهایی که تا الان یاد گرفتیم، اینجا هم می تونیم پیاده کنیم؟

استاد: صدالبته! با یک لیست سروکار داریم؛ پس مثلاً خروجی زیر چه می شود؟
`cities[0]`

دانشجو: * یعنی عضو اول لیست؛ پس می شه:

```
'Shiraz'
```

استاد: دقیقاً! یا مثلاً:

```
cities.append('Fooladshahr')
```

دانشجو: چه جالب؛ شهر فولادشهر هم به انتهای لیست اضافه شد.

استاد: یادش به خیر؛ فولادشهر... شهری که کودکی و نوجوانی مان در آنجا سپری

شد... و یا مثلاً:

```
cities.count('Shiraz')
```

دانشجو: نوشت ۱. چون تنها یه رشته به نام شیراز در شهرهامون داشتیم.

استاد: درود بر تو! خب برگردیم به سؤال اصلی مون. فرض کنید یه لیست داریم که شهرها در اون نوشته شده:

```
cities = ['Shiraz', 'Esfahan', 'Tehran']
```

لیست دیگری تعریف کنیم به نام populations و به ترتیب، جمعیت هر کدام از این شهرها را بر اساس میلیون نفر بنویسیم (اعداد فرضی هستند):

```
populations = [2, 2.5, 12]
```

و لیست دیگری هم به نام areas که مساحت این شهرها را به ترتیب نوشتیم:

```
areas = [2, 3, 8]
```

یعنی به عبارتی کل برنامه ما:

```
cities = ['Shiraz', 'Esfahan', 'Tehran']
```

```
populations = [2, 2.5, 12, 1.5]
```

```
areas = [2, 3, 8]
```

حالا اگر بخواهیم تمام اطلاعات مربوط به هر شهر به ترتیب برای ما نمایش داده شوند، چه کنیم؟

دانشجو: نمی‌دونم استاد!

استاد: بیایید کار را ساده‌تر کنیم. اگر بخواهیم فقط نام شهرها نمایش داده شود، چه کنیم؟

دانشجو: تا اونجایی که یادمه و تا الان عمل می‌کردیم، با حلقه For روی لیست

موردنظرمان حرکت می کردیم، یعنی:

```
for city in cities:  
    print(city)
```

استاد: درسته؛ اما در اینجا ما به اندیس هم در هر لحظه نیاز داریم تا بدانیم که الان عضو چندم لیست هستیم تا مثلاً اگر عضو اول هستیم، برویم و عضو اول را در لیست جمعیت به عنوان جمعیت آن شهر بیاوریم.

دانشجو: آهان درسته استاد! خب برای این که بخوایم طوری از حلقه for استفاده کنیم که به اندیس هم دسترسی داشته باشیم، همون طور که یاد گرفتیم می تونیم این طور عمل کنیم:

```
for i in range(len(cities)):  
    print(i)
```

چون هر سه لیست ما طولشان هم اندازه است، به نظرم فرقی نداشت که از کدام لیست استفاده کنم و برای همین از cities استفاده کردم. الان در خروجی، ما به اندیس ها دسترسی داریم، یعنی:

```
0  
1  
2
```

استاد: بسیار عالی! حال اگر بنویسیم:

```
print(cities[i])
```

چه اتفاقی می افتد؟

دانشجو: الان برنامه ما شد:

```
for i in range(len(cities)):  
    print(cities[i])
```

یادمه که هر موقع می‌خواستیم برنامه رو بهتر بفهمیم، اون رو خط به خط می‌خوندیم و خودمون رو جای کامپیوتر می‌داشتیم. وقتی کامپیوتر به این خط می‌رسه:

```
for i in range(len(cities)):
```

یه متغیر به نام `i` تعریف کرده و مقدار اون رو برابر با شروع، یعنی `۰` می‌ذاره. حالا این خط:

```
print(cities[i])
```

چون `i` برابر با صفره، انگار این خط برنامه معادل خط زیره:

```
print(cities[0])
```

یعنی عضو صفرم لیست ما به عبارتی `'Shiraz'` رو چاپ می‌کنه و سپس دوباره به ابتدای حلقه برمی‌گردیم و این بار `i` برابر با `۱` می‌شه و همین روال تا انتهای لیست تکرار می‌شه و تمامی شهرها چاپ می‌شن، یعنی:

```
Shiraz
Esfahan
Tehran
```

استاد: بسیار عالی! حال اگر بنویسیم:

```
print(populations[i])
```

دانشجو: منطقاً جمعیت هر شهر هم در حلقه چاپ می‌شه.

استاد: پس اگر موافق باشی، کنار هم بنویسیم که کنار هم چاپ شوند:

```
for i in range(len(cities)):
    print(cities[i], populations[i])
```

که خروجی آن می‌شود:

```
Shiraz 2
Esfahan 2.5
```

Tehran 12

هر شهر در کنار جمعیت خودش.

دانشجو: استاد اگر داده‌های جدولی و اکسل و ... داشتیم چطور؟

استاد: ما مفصل با داده‌های جدولی در فصل‌های بعدی کار داریم و با کتابخانه‌ای به نام پانداز آشنا می‌شویم که خیلی راحت می‌توانیم فایل بخوانیم و به داده‌های خودمان دسترسی داشته باشیم.

دانشجو: خب استاد ببخشید، ولی سؤالم اینه که چرا مستقیم نمی‌ریم پانداز رو یاد بگیریم؟

استاد: درست است که در بسیاری از مواقع از کتابخانه پانداز استفاده می‌کنیم، اما بسیاری از کارها وجود دارد که اگر با اصول و مقدمات آن تا حدودی آشنا نباشیم، نمی‌توانیم خودمان در پایتون انجام بدهیم و در بعضی مواقع مجبوریم خودمان ساختار داده خودمان را بسازیم (که برای خود من بسیار پیش آمده). نگران نباشید، جوری یاد می‌گیریم که زیاد از حد درگیر مقدمات شویم و نه این‌که کورکورانه بدون هیچ مقدمه خاصی به سراغ پانداز برویم.

خب، پس تا اینجا دیدیم که می‌توانیم به این شکل، به نام هر شهر، جمعیت آن و مساحت آن دسترسی داشته باشیم و آن را به کاربر نمایش دهیم:

```
cities = ['Shiraz', 'Esfahan', 'Tehran']
```

```
populations = [2, 2.5, 12, 1.5]
```

```
areas = [2, 3, 8]
```

```
for i in range(len(cities)):
```

```
    print(cities[i], populations[i], areas[i])
```

چطور می‌توانم برنامه‌ای بنویسیم که تنها شهرهایی که جمعیت بیشتر از ۲ میلیون دارند را به ما نشان دهد؟

دانشجو: به نظرم باید از شرط‌ها استفاده کنیم... اما چطورش رو نمی‌دونم!

استاد: همین‌طور به زبان محاوره با خودت بگو، چه شرطی نیاز داریم؟

دانشجو: ... شرط جمعیت شهر > بزرگ‌تر از ۲.

استاد: همین را می‌توانیم به حلقه اضافه کنیم؟

آهان گرفتیم؛ ما که جمعیت هر شهر رو در هر حلقه داریم به دست می‌آریم. پس کافیه از اون در شرط استفاده کنیم، یعنی:

```
if populations[i]>2:
```

که وقتی این شرط برقرار بود، حالا جمعیت رو به ما نشان بده:

```
for i in range(len(cities)):
```

```
    if populations[i]>2:
```

```
        print(cities[i], populations[i], areas[i])
```

چه باحال بود! خوشم اومد! استاد یه سؤال! من یادمه در جلسات قبل خیلی از `range(len(cities))` خوشمون نمی‌آمد و از دستور `enumerate` استفاده کردیم. اینجا هم می‌تونیم از اون استفاده کنیم؟

استاد: سؤال خوبیه! به‌عنوان بیشتر بدانید، بله چرا که نه! می‌توانیم زیبایی کدمان را افزایش دهیم:

```
for i, city in enumerate(cities):
```

```
    if populations[i]>2:
```

```
        print(city, populations[i], areas[i])
```


اما باز هم مجبوریم از اندیس‌ها استفاده کنیم. در این گونه مواقع که می‌خواهیم هم‌زمان روی دو یا تعداد بیشتر از لیست‌ها حرکت کنیم، تابعی به نام zip وجود دارد که اگر دوست داشتید، می‌توانید از آن استفاده کنید:

```
zip(cities, populations, areas)
```

که برای استفاده از آن در حلقه، به راحتی کافی است بنویسیم:

```
for city, population, area in zip(cities, populations, areas):
```

```
    if population > 2:
        print(city, population, area)
```

دانشجو: چقدر این دستور zip کار رو راحت تر کرد.

استاد: دقیقاً! با هر روش که راحت تر هستید، همان را انجام دهید. روش i منطقی تر است و به همین دلیل برای شروع از آن استفاده کردیم، اما استفاده از دستور zip خواناتر است.

حالا اگر بخواهیم ببینیم شهرهای بالای ۲ میلیون نفر، مجموعاً چه جمعیتی دارند چه کنیم؟

دانشجو: به نظرم اصل کار همون کاری بود که کردیم، یعنی دسترسی به شهرهایی با جمعیت ۲ میلیون! می‌تونیم یه متغیر (یا همون کیسه) خالی به اسم total ایجاد کنیم و هر وقت به شهری که جمعیت بیشتر از ۲ میلیون داشت رسیدیم، جمعیت اون رو به total اضافه کنیم، یعنی:

```
cities = ['Shiraz', 'Esfahan', 'Tehran']
populations = [2, 2.5, 12, 1.5]
areas = [2, 3, 8]
```

```
total = 0
for i in range(len(cities)):
    if populations[i]>2:
        total += populations[i]

print(total)
```

استاد: اگر بخواهیم به شیوه خواناتر که الآن یاد گرفتیم، یعنی دستور zip کار کنیم، کارمان حتی راحت‌تر هم می‌شود:

```
cities = ['Shiraz', 'Esfahan', 'Tehran']
populations = [2, 2.5, 12, 1.5]
areas = [2, 3, 8]

total = 0
for city, population, area in zip(cities, populations,
areas):
    if population>2:
        total += population

print(total)
```

دانشجو: آره دقیقاً استاد!

استاد: تاکنون، ما اطلاعات شهرها را در سه لیست جداگانه داشتیم:

```
cities = ['Shiraz', 'Esfahan', 'Tehran']
populations = [2, 2.5, 12, 1.5]
areas = [2, 3, 8]
```

آیا می‌توانیم به جای سه لیست جدا، تنها یک لیست داشته باشیم؟ البته! به شکل لیستی از لیست‌ها!

دانشجو: استاد باز می‌فهمم ولی متوجه نمی‌شم!

استاد: الآن در عمل می‌بینیم و بهتر متوجه می‌شویم! به این لیست نگاه کنید:

```
[ 'Esfahan', 2.5, 3] data = [[ 'Shiraz', 2, 2],
```

عضو اول این لیست ما چیست؟

دانشجو: عضو اول ما 'Shiraz' است؟

استاد: نه! عضو اول لیست ما این است!

```
[ 'Shiraz', 2, 2]
```

و عضو دوم ما این است:

```
[ 'Esfahan', 2.5, 3]
```

دانشجو: چه جالب! یعنی عضوهای ما هر کدام به لیست هستند!

استاد: دقیقاً! همان‌طور که احتمالاً بدانید، پایتون ایرادی نمی‌گیرد که در تعریف یک

لیست، هر عضو را در یک خط جدید تعریف کنیم، به عبارتی مثلاً می‌توان لیستمان

را به این شکل هم نوشت که خواناتر باشد:

```
data = [  
    [ 'Shiraz', 2, 2],  
    [ 'Esfahan', 2.5, 3],  
    [ 'Tehran', 12, 8]  
]
```

دانشجو: مگه می‌شه! مگه داریم! یعنی می‌شد مثلاً لیست زیر رو:

```
cities = [ 'Shiraz', 'Esfahan', 'Tehran']
```

به این شکل هم نوشت؟

```
cities = ['Shiraz',
          'Esfahan',
          'Tehran']
```

بذارین امتحان کنم... چه باحال؛ انگاری شد!

استاد: در بسیاری از مواقع برای خوانایی بیشتر، مخصوصاً وقتی هر کدام از اعضای لیستمان طولانی هستند، از این روش استفاده می‌شود تا خوانایی بیشتری داشته باشد. به موضوع بحثمان برگردیم... پس دیدیم می‌توانیم سه لیست جداگانه‌ای داشتیم یعنی:

```
cities = ['Shiraz', 'Esfahan', 'Tehran']
populations = [2, 2.5, 12, 1.5]
areas = [2, 3, 8]
```

آن را به صورت یک لیستی از لیست‌ها بنویسیم، مثلاً:

```
data = [
    ['Shiraz', 2, 2],
    ['Esfahan', 2.5, 3],
    ['Tehran', 12, 8]
]
```

دانشجو: استاد حالا چطوری ازش استفاده کنیم؟

استاد: به نظرتان خروجی دستور زیر چیست؟

```
print(data[0])
```

دانشجو: تا جایی که می‌دونم یعنی عضو اول لیست رو به ما نشان بده. آهان! دیگه اشتباه قبلی رو نمی‌کنم! عضو اول لیست ما، خودش یه لیسته، یعنی:

```
['Shiraz', 2, 2]
```

چه جالب، امتحان کردم، همین شد!

استاد: و به همین ترتیب خروجی دستور:

```
data[1]
```

می‌شود دومین عضو که خودش یک لیست است:

```
['Esfahan', 2.5, 3]
```

حال به نظرت، خروجی دستور زیر چیست؟

```
data[0][0]
```

دانشجو: جالب شد؛ اولین باریه که همچنین چیزی می‌بینم! بذارین همون طور که تا الان همیشه تأکید داشتین مسئله رو ساده کنم! خروجی دستور زیر:

```
data[0]
```

می‌شه عضو اول لیست که خودش یه لیسته:

```
['Shiraz', 2, 2]
```

حالا ما گفتیم:

```
data[0][0]
```

یعنی عضو اول لیست نهایی! الان هم که در پایتون چک کردم، شد 'Shiraz' پس درست متوجه شدم! هورا به خودم!!

استاد: نحوه تحلیل کردن تو را خیلی دوست داشتم! دقیقاً همین طور است! ما به این شکل می‌تونیم به تک‌تک المان‌های موجود در لیستمان دسترسی داشته باشیم! حالا سؤال دیگر، به نظرت خروجی دستور زیر چیست؟

```
for i in data:  
    print(i)
```

دانشجو: یادمه هر وقت مستقیماً روی خود لیست حلقه می‌زدیم و از range استفاده نمی‌کردیم، هربار به یکی از اعضای لیست دسترسی پیدا می‌کردیم. پس منطقاً باید در بار اول حلقه، i برابر با:

```
['Shiraz', 2, 2]
```

شده و وقتی به خط زیر می‌رسیم:

```
print(i)
```

همین پرینت بشه و وقتی که به ابتدای حلقه برمی‌گردیم، i برابر با عضو دوم لیست بشه، یعنی:

```
['Esfahan', 2.5, 3]
```

و همین‌طور تا آخر!

استاد: برای همین بهتر است که به جای اسم متغیر i از یک اسم واضح‌تر مثلاً line (یا هر نام دیگری که شما تمایل داشتید) استفاده کنیم که برنامه خوانا تر شود، مثلاً:

```
for line in data:  
    print(line)
```

حالا ببینیم که چطور می‌توانیم اطلاعات شهر را به دست بیاوریم؟ دیدیم که هربار متغیر line یکی از اعضا یعنی یکی از لیست‌ها می‌شود، شهر عضو چندم این لیست است؟

دانشجو: عضو اول هر لیست مربوط به شهر، عضو دوم مربوط به جمعیت و عضو سوم مربوط به مساحت.

استاد: پس می‌توانیم بنویسیم:

```
city = line[0]  
population = line[1]
```

```
area = line[1]
```

و به عبارتی برنامه ما می شه:

```
for line in data:
    city = line[0]
    population = line[1]
    area = line[2]
    print(city, population, area)
```

خروجی برنامه:

```
Shiraz 2 2
Esfahan 2.5 2.5
Tehran 12 12
```

در پایتون می توانیم این برنامه رو به شکل جالبتری هم بنویسیم و به جای این حالت:

```
city = line[0]
population = line[1]
area = line[2]
```

می توانیم بنویسیم:

```
city, population, area = line
```

و خود پایتون به صورت نظیر به نظیر این متغیرها را تعریف و مقداردهی می کند.

دانشجو: چقدر پایتون امکانات باحالی داره!

استاد: به این کار در پایتون unpacking می گویند که کار را بسیار راحت تر می کند.
همان طور که می بینید کل برنامه ما در تعداد خط بسیار کمتری نوشته شد:

```
for line in data:
    city, population, area = line
    print(city, population, area)
```

حال حتی پایتون کار را راحت‌تر هم کرده است! شما لازم نیست یک متغیر line در حلقه تعریف کنید و سپس آن را unpack کنید، می‌توانید مستقیماً در حلقه، همین سه متغیر را بنویسید، یعنی:

```
for city, population, area in data:
```

و ببینید با این روش آخر کل برنامه ما چقدر کوچک‌تر شد!

```
for city, population, area in data:
```

```
    print(city, population, area)
```

دانشجو: چه جالب! استاد می‌شه یه بار باهم برنامه رو خط به خط ردگیری کنیم تا بهتر یاد بگیرم؟

استاد: بله حتماً! پایتون وقتی به این خط می‌رسد:

```
for city, population, area in data:
```

اولین عضو ما در لیست data چه بود؟

دانشجو: این بود:

```
[ 'Shiraz', 2, 2 ]
```

استاد: پایتون سه متغیر تعریف می‌کند به نام city, population و area و مقدار city می‌شود برابر با 'Shiraz'، مقدار population برابر با ۲ و مقدار area برابر با ۲ که ما می‌توانیم در ادامه از این اطلاعات استفاده کنیم. مثلاً اگر بخواهیم برنامه‌ای بنویسیم که مجموع جمعیت تمام شهرها را به ما بدهد، باید چه کنیم؟

دانشجو: می‌توانیم یک متغیر به اسم total با مقدار اولیه ۰ تعریف کنیم و به هر شهر که رسیدیم، جمعیت اون رو به این متغیر اضافه کنیم، یعنی:

```
data = [
    [ 'Shiraz', 2, 2 ],
```



```
[ 'Esfahan', 2.5, 3],  
[ 'Tehran', 12, 8]  
]  
total = 0  
for line in data:  
    city, population, area = line  
    total += population  
  
print(f"The sum of the populations was {total}")
```

استاد: بسیار عالی! اگر بخواهیم در خود حلقه متغیرها را unpack کنیم می‌توانیم مستقیماً بنویسیم:

```
total = 0  
for city, population, area in data:  
    total += population  
  
print(f"The sum of the populations was {total}")
```

که دیگر نیازی به متغیر line نداشته باشیم.

دانشجو: آهان! حالا استاد یه خط کمتر و بیشتر که تفاوتی نداره!

استاد: از دست تو! درست است اما بعضی مواقع حتی یک خط کمتر هم کمک می‌کند که برنامه خواناتر شده و بعداً راحت‌تر بتوان آن را فهمید و در صورت نیاز تغییر داد یا چیزی به آن اضافه کرد.

استخراج اطلاعات از یک متن (مثال شهر و جمعیت)

فرض کنید که اطلاعات شهر مادر قالب یک رشته متنی باشند یک لیست، برای مثال:

```
txt = 'Shiraz 2 1.5'
```

و توجه داریم که این یک رشته است نه یک لیست! یعنی کلاً یک متن یکپارچه است.

در پایتون، رشته‌ها، متدی دارند به نام `split`. این متد چه می‌کند؟ اگر من بنویسم:

```
txt.split()
```

خروجی این دستور چیست؟

دانشجو: بذارین در محیط کنسول امتحان کنم. خب `txt` رو تعریف کردم و حالا دستوری که گفتین...

```
['Shiraz', '2', '1.5']
```

چه جالب شد؛ خودش موارد رو به‌صورت یه لیست درآورد و انگار اجزا رو از هم جدا کرد!

استاد: جالب است که به متد `split` می‌توانیم بگوییم که با چه کاراکتری، رشته‌ما را از هم تفکیک می‌کند، مثلاً:

```
txt.split(' ')
```

که به این معناست که با کاراکتر فاصله، رشته‌ما را از هم جدا کند. ما می‌توانیم مثلاً بگوییم با کاراکتر `tab` رشته‌ما را از هم جدا کن که بعداً در کار با داده‌های جدولی بسیار به کارمان می‌آید:

```
txt.split('\t')
```

ولی اگر هیچ‌چیزی ننویسید، پایتون به‌صورت پیش‌فرض هر کاراکتر فاصله چه “ ” یا `'t'` یا `'n'` وجود داشته باشد را تفکیک می‌کند. نام این آرگومان همان‌طور که در توضیحات این دستور که ظاهر شد احتمالاً دیدید، `sep` است، مخفف کلمه `separator`

یعنی جداکننده. مای توانستیم این دستور را به این شکل نیز استفاده کنیم:

```
txt.split(sep='\\t')
```

اگر تمایل داشتید، می توانید به عنوان یک تمرین دل خواه برای علاقه بیشتر خودتان، برنامه ای بنویسید که کار split را انجام دهد تا درک برنامه نویسی شما بیشتر شود.

خب تا الان یک رشته داشتیم:

```
txt = 'Shiraz 2 1.5'
```

چطور می توانیم شهر، جمعیت و مساحت را در متغیرهای دیگری به اسم city و ... ذخیره کنیم؟

دانشجو: ... تا اینجا خودتون راهنمایی کردین که متدی به نام split داریم که می تونه رشته ما رو به صورت یه لیست تفکیک کنه که در اینجا یه لیست سه تایی داریم؛ پس می تونیم همون کاری که بلد بودیم رو انجام بدیم. حاصل این دستور رو در یه متغیر بریزیم، مثلاً info:

```
info = txt.split()
```

و الان محتوای info یه لیسته. حالا متغیرها رو مقداردهی کنیم:

```
city, population, area = info
```

استاد: بسیار عالی! می توانستیم به صورت مستقیم هم متغیرها را مقداردهی کنیم و دیگر متغیری به نام info نداشته باشیم:

```
city, population, area = txt.split()
```

حال یک سؤال! به نظرت خروجی دستور زیر چیست؟

```
txt.split()[0]
```

دانشجو: یه مقدار ظاهرش پیچیده اس!!!

استاد: دقت کن که مرحله به مرحله چه چیزی داریم...

دانشجو: خب قسمت:

```
txt.split()
```

که می‌شه:

```
['Shiraz', '2', '1.5']
```

آهان!!!! پس [0] به معنی اینه که عضو اول این لیست رو می‌خوایم؛ پس درواقع جواب سؤال شما می‌شه 'Shiraz'!

استاد: دقیقاً! بعضی مواقع شاید ما فقط یکی از اعضا را نیاز داشته باشیم و می‌توانیم مستقیماً برای مثال بنویسیم:

```
city = txt.split()[0]
```

ولی در مثالی که داشتیم، چون اطلاعات بیشتری را می‌خواستیم، همان روش unpacking بهتر بود.

```
txt = 'Shiraz 2 1.5'  
city, population, area = txt.split()  
print(city, population, area)
```

می‌خواهیم لیستی از شهرها، جمعیت‌ها و مساحت‌ها (شبیه لیست data که در دو تمرین قبل دیدیم) را داشته باشیم، اما این لیست را از مخاطب بپرسیم! به این شکل که به مخاطب بگوییم لطفاً شهر، جمعیت و مساحت را با علامت فاصله وارد کن، مثلاً:

```
print('Please enter city, population and area with  
space:')
```

و مخاطب مثلاً وارد کند:

```
Shiraz 2 1.5
```

و وقتی کلید Enter را روی کیبورد فشار داد، کامپیوتر، شهر بعدی را بپرسد و این کار را تا ۳ بار تکرار کند و در نهایت یک متغیر به اسم data داشته باشیم که شبیه قبل، لیستی از لیست‌ها باشد، مثلاً:

```
data = [  
    ['Shiraz', 2, 2],  
    ['Esfahan', 2.5, 3],  
    ['Tehran', 12, 8]  
]
```

دانشجو: ... استاد سؤال خوبی! اما فقط می‌تونم همینو بگم نظر دیگه‌ای ندارم!

استاد: ...

دانشجو: ...

استاد: ... (نفس عمیق...) بیا پسر! اول روی کاغذ، به زبان خودمان و ساده‌شده بنویسیم که می‌خواهیم چه کار کنیم. شعار هرکس یادمان نرود: «هر برنامه پیچیده، زمانی برنامه‌ای ساده بوده که کار می‌کرده است». ما می‌خواهیم سه بار این کار تکرار شود. بیا سه بار تکرار را رها کنیم، چطور می‌توانیم یک ورودی از مخاطب بگیریم و این کار را انجام دهیم؟

دانشجو: استاد سؤال بهتری شد! اما باز هم می‌تونم همینو بگم!

استاد:

دانشجو:

استاد: (نفسی عمیق‌تر می‌کشد...) پسر، باز مسئله را ساده‌تر کن! چطور می‌توانیم از مخاطب یک ورودی بگیریم؟

دانشجو: اینو دیگه بلدم! با دستور `input`:

```
input('Please enter city, population and area with  
space:\n')
```

و نتیجه رو هم بذارین در یه متغیر بریزم؛ چون دارم یه ورودی متنی می‌گیرم،
بذارین اسمش رو `txt` بذارم:

```
txt = input('Please enter city, population and area  
with space:\n')
```

عه! این که شبیه همون مثال قبل شد! ما یه رشته داریم که با فاصله می‌خوایم از هم
تفکیک‌شون کنیم، یعنی کافیه از متد `split` استفاده کنیم:

```
txt.split()
```

و چون گفتیم شهر، جمعیت و مساحت را وارد کنه، پس درنهایت یه لیست سه‌تایی
داریم، یعنی:

```
city, population, area = txt.split()
```

یعنی تا الان برنامه ما به این شکل شد:

```
txt = input('Please enter city, population and area  
with space:\n')  
city, population, area = txt.split()
```

چه خوب پیش رفتما! ولی ... قرار بود چیکار کنیم؟ یادم رفت!

استاد: آفرین خیلی خوب پیش رفتی! قرار بود که درنهایت یک لیست شبیه به این
داشته باشیم:

```
data = [  
    ['Shiraz', 2, 2],  
    ['Esfahan', 2.5, 3],
```

```
[ 'Tehran', 12, 8 ]  
]
```

دانشجو: استااااا سؤال خوبیه اما...

استاد: پسرم! سؤال را ساده کن! ما می خواهیم به لیستمان چیزی اضافه کنیم، پس اول از همه باید یک لیست خالی داشته باشیم...

دانشجو: درسته! لیستی خالی به اسم data:

```
data= []
```

استاد: و ما با چه دستوری می توانستیم چیزی را به لیستمان اضافه کنیم؟

دانشجو: ... آهان متد append!

استاد: بسیار عالی! و حال قرار است چه چیزی را به این لیست اضافه کنیم؟

دانشجو: قراره شهر، جمعیت و مساحت این شهر در قالب یک لیست به این لیست اضافه بشه، یعنی این:

```
data.append(city, population, area)
```

و در نهایت هم محتوای لیست data رو چاپ کنیم که ببینیم به چه نتیجه ای رسیدم:

```
print(data)
```

به عبارتی کل برنامه مون شد:

```
data= []  
txt = input('Please enter city, population and area  
with space:\n')  
city, population, area = txt.split()  
data.append(city, population, area)  
print(data)
```

بذارین اجراش کنم؛ خب از من خواست وارد کنم. من مثلاً وارد می‌کنم:

Shiraz 1 1.5

و اینتر رو زدم... الآن باید درست بشه! ... عه چرا خطا داد؟!!

```
-----  
-----  
TypeError                                Traceback (most recent call last)  
Cell In[12], line 4
```

```
    2 txt = input('Please enter city, population  
    and area with space:\n')  
    3 city, population, area = txt.split()  
----> 4 data.append(city, population, area)
```

TypeError: list.append() takes exactly one argument (3 given)

استاد: کجا خطا داده است؟ چه خطایی؟

دانشجو: در این خط:

```
data.append(city, population, area)
```

و خطایی که آخر کار نوشته:

TypeError: list.append() takes exactly one argument (3 given)

من به درد برنامه‌نویسی نمی‌خورم! 😞 ...

استاد: (یک نفس عمیق...) همه برنامه‌نویس‌ها حداقل به چند خطا در طول یک برنامه برخورد می‌کنند... از جمله خود من...؛ خیلی هم دردناک است اما با یک نفس عمیق و فکر کردن و آزمون و خطا، علت خطا را پیدا می‌کنیم و در نهایت اگر نتوانستیم،

جستجو می کنیم یا از هوش مصنوعی مثل چت جی پی تی کمک می گیریم.

خب برگردیم سر موضوع اصلی. قرار بود که ما در قالب یک لیست، اطلاعات را اضافه کنیم، درست است؟

دانشجو: درسته استاد.

استاد: پس درواقع می خواهیم این را به data اضافه کنیم؛ صحیح؟

```
[city, population, area]
```

دانشجو: آهان درسته! من باید می نوشتم:

```
data.append([city, population, area])
```

یعنی برنامه من شد:

```
data= []  
txt = input('Please enter city, population and area  
with space:\n')  
city, population, area = txt.split()  
data.append([city, population, area])  
print(data)
```

بذارین برنامه رو امتحان کنم... باورم نمیشه درست شد!! (اشک شوق!!) و خروجی

نهایی شد:

```
data= []  
txt = input('Please enter city, population and area  
with space:\n')  
city, population, area = txt.split()  
data.append([city, population, area])  
print(data)
```

استاد: درود بر تو! همه ما با این حس رودررو شده ایم! حواست باشد و بدان که تنها

نیستی!

در نهایت اگر می‌خواستیم این کار را سه بار انجام دهیم و به عبارتی سه بار از مخاطب اطلاعات بگیریم، چطور می‌توانیم کاری کنیم کامپیوتر یک کار را به تعداد مشخص تکرار کند؟

دانشجو: ... حلقهٔ for؟

استاد: دقیقاً! پس می‌توانیم کل برنامه را در یک حلقه قرار دهیم.

دانشجو: آهان درسته یعنی:

```
for i in range(3):  
    data= []  
    txt = input('Please enter city, population and  
    area with space:\n')  
    city, population, area = txt.split()  
    data.append([city, population, area])
```

```
print(data)
```

عجب چیزی نوشتم! بذارین تستش کنیم. خب اولین شهر رو پرسید وارد کردم، چه جالب!!! رفت دومین شهر! اون رو هم وارد کردم و سومین شهر... بریم که نتیجهٔ شاهکار رو ببینیم... عه! چرا فقط آخرین شهر داخل لیستمه و خبری از دو شهر دیگه نیست؟!!! 😞

```
[[ 'Tehran', '15', '10' ]]
```

استاد: بیا خودمان را به جای کامپیوتر بگذاریم و برنامه را خط به خط در ذهنمان اجرا کنیم تا علت خطا را پیدا کنیم.

دانشجو: بریم استاد! کامپیوتر اول به این خط می‌رسه:

```
for i in range(3):
```

و به متغیر به اسم `i` تعریف می‌کنه و مقدار اون رو `۰` می‌ذاره (البته این متغیر رو فقط گذاشتیم که کامپیوتر دستورات بعد رو `۳` بار تکرار کنه وگرنه استفاده خاصی از `i` نداریم).

سپس در خط بعد به لیست خالی تعریف می‌شه:

```
data= []
```

در خط بعد اطلاعات از مخاطب در قالب به رشته دریافت می‌شه:

```
txt = input('Please enter city, population and area with space:\n')
```

و در خط بعد به صورت لیست، تفکیک می‌شه و متغیرها مقداردهی می‌شن:

```
city, population, area = txt.split()
```

و در خط بعد به لیست ما، این لیست `append` می‌شه:

```
data.append([city, population, area])
```

نمی‌فهمم کجای این برنامه مشکل داره!

استاد: ادامه بده!

دانشجو: چشم استاد! کامپیوتر به ابتدای حلقه برمی‌گرده:

```
for i in range(3):
```

و این بار `i` برابر با `۱` می‌شه. سپس در خط بعد:

```
data= []
```

یک لیست خالی تعریف می‌شه به اسم `data`...

یک لیست خالی تعریف می‌شه؟! یعنی اطلاعات قبلی از بین می‌ره!!! ای‌وای! یعنی

در هر بار حلقه داده لیست data از ابتدا تعریف می‌شه و برابر با [] می‌شه!

استاد: درود بر تو! پس باید چه کار کنیم؟

دانشجو: ... آهان من باید لیست خالی data رو قبل از شروع حلقه تعریف می‌کردم
نه در هر بار حلقه! یعنی:

```
data= []
for i in range(3):
    txt = input('Please enter city, population and
    area with space:\n')
    city, population, area = txt.split()
    data.append([city, population, area])

print(data)
```

بذارین امتحانش کنم! خب شهر اول، شهر دوم و شهر سوم... درست کار کرد!!!! ایول!
[['Shiraz', '2', '1.5'], ['Esfahan', '2', '2'], ['Tehran',
'15', '10']]

داره از برنامه‌نویسی خوشم می‌آد! (٩٠)

استاد: درود بر تو؛ دقیقاً همین است! لذتی که پس از دیباگ کردن کد و کشف
باگ (مخصوصاً خطاهای منطقی که ما خطای واضحی از برنامه دریافت نمی‌کنیم ولی
برنامه درست کار نمی‌کند) وصف‌ناپذیر است!

فقط یک نکته بسیار مهم! چرا به نظرت در نتیجه نهایی data، اعداد در علامت °
قرار گرفته‌اند؟ مثلاً '2' یا '1.5'؟

دانشجو: ... یعنی از جنس رشته هستن نه عدد اعشاری؟

استاد: دقیقاً همان‌طور که در جلسهٔ اول دیدیم، در `input`، ورودی به‌صورت یک رشته دریافت می‌شود، درحالی‌که ما اینجا می‌خواهیم یک عدد داشته باشیم که بتوانیم بعداً اگر خواستیم روی آن‌ها محاسبات انجام دهیم. برای این کار می‌توانیم چه کنیم؟ مثلاً می‌توانیم در خط بعد از:

```
city, population, area = txt.split()
```

متغیرهای مدنظر را به نوع متغیری که می‌خواهیم تغییر دهیم. جمعیت و مساحت متغیرهای کمی هستند که می‌توانند اعداد اعشاری هم داشته باشند؛ پس به نظر می‌آید متغیر `float` برای آن‌ها مناسب باشد:

```
population = float(population)
area = float(area)
```

و به عبارتی کل برنامه ما شد:

```
data= []
for i in range(3):
    txt = input('Please enter city, population and
    area with space:\n')
    city, population, area = txt.split()
    population = float(population)
    area = float(area)
    data.append([city, population, area])

print(data)
```

و اگر برنامه را امتحان کنیم، می‌بینیم این بار، دیگر اعداد به‌صورت رشته نیستند:

```
[['Shiraz', 2.0, 1.5], ['Esfahan', 2.0, 2.0], ['Tehran',
15.0, 10.0]]
```

دانشجو: نکتهٔ بسیار ظریفی بود!

استاد: راه دیگر این است که موقع `append` کردن به لیست، `float` شده این متغیرها را به لیست نهایی خود اضافه می‌کردیم:

```
data.append([city, float(population), float(area)])
```

که در این صورت برنامه ما کوتاه‌تر می‌شود:

```
data= []
for i in range(3):
    txt = input('Please enter city, population and
area with space:\n')
    city, population, area = txt.split()
    data.append([city, float(population), float(area)])

print(data)
```

دانشجو: نکته‌ای بسیار ظریف‌تر بود!

استاد: چون می‌بینم که به نکات ظریف علاقه داری، یک نکته ظریف دیگر هم بگویم! در اینجا همان‌طور که گفتی از `i` استفاده‌ای نکردیم و فقط قصدمان تکرار برنامه بود. درست است؟

دانشجو: درسته استاد!

استاد: در این‌گونه موارد بهتر است اسم متغیر حلقه خود را چیزی بی‌معنی مانند `_` بگذاریم تا مخاطبی که کد ما را می‌خواند بداند که هدف ما فقط تکرار برنامه بوده و قرار نبوده از این متغیر استفاده‌ای شود، یعنی:

```
data= []
for _ in range(3):
    txt = input('Please enter city, population and
area with space:\n')
    city, population, area = txt.split()
```

```
float(area)])    data.append([city, float(population),

print(data)
```

دانشجو: استااااا دیدی خیال !!!

استاد: ... اگر به فکر نفرت بعد از خود که قرار است کد ما را بخوانند و از آن استفاده کرده یا ارتقا دهند نیستیم، حداقل به فکر خودمان باشیم که ممکن است بعداً به سراغ کد خود بیاییم و اگر تمیزتر آن را نوشته باشیم، وقت و انرژی کمتری نیاز است تا متوجه منظور کد خودمان شویم! چیزی که من اوایل برنامه‌نویسی رعایت نمی‌کردم و بعداً متوجه شدم! مثل معروفی در دنیای برنامه‌نویسی وجود دارد که می‌گویند: «وقتی یک برنامه‌نویس، کدی را می‌نویسد، فقط دو نفر می‌دانند که این کد چه کار می‌کند؛ خودش و خدا ... اما پس از یک هفته، فقط خدا می‌داند که آن کد چه کار می‌کند! با تمیزتر نوشتن کد، درواقع به خودمان در آینده کمک می‌کنیم!

دانشجو: شیر فهم شدم استاد!

استاد: برویم سراغ مبحث شیرین تمرین‌ها!

دانشجو: نه نه نه نه نه!!!!

استاد: 😊😊😊

تمرین‌های جلسه پنجم

در تمرین هر درس ممکن است که یک نکته جدید هم آموزش داده شود، پس حتماً توصیه می‌کنم آن‌ها را ببینید و انجامشان دهید. تمرین‌ها و به خصوص خطاهایی که دریافت می‌کنید و آزمون و خطاها (و همچنین سرچ‌هایی که در صورت نیاز انجام می‌دهید)، بسیار بسیار به روند یادگیری برنامه‌نویسی شما کمک می‌کند. پاسخ نمونه برای تمرین‌ها نیز پس از آن‌ها قرار داده شده است.

تمرین‌ها

۱- برنامه‌ای بنویسید که ۲ بار از مخاطب ورودی بگیرد به این شکل food و نتیجه را به شکل یک لیست نمایش دهد. مثال:

Apple 23

Orange 100

نتیجه:

```
[[ 'Apple', 23], [ 'Orange', 300]]
```

۲- در ادامه برنامه قبل، برنامه‌ای بنویسید که مجموع کالری‌های تمام این غذاها را حساب کند.

۳- برنامه‌ای بنویسید که هزار بار تاس بریزد و در یک لیست ذخیره کند. سپس بگوید که چند بار عدد چهار آمده است. (برای تولید عدد تصادفی در پایتون، نیاز به یک سرچ کوچک دارید 😊)

۴- تمرین همراه با سرچ: چگونه می‌توان یک رشته را به لیست تبدیل کرد؟ و چگونه می‌توان یک لیست را با سرهم کردن اعضای آن به یک رشته تبدیل کرد؟ برای مثال:

```
'ATCTG' -> [ 'A', 'T', 'C', 'T', 'G' ] -> 'ATCTG'
```

به نظر تان این کار چه مزیت‌ها و امکاناتی را می‌تواند فراهم کند؟

پاسخ تمرین‌های جلسه پنجم

اکیداً توصیه می‌کنم که اول خودتان تمرین‌ها را حل کرده و سپس به پاسخ‌ها مراجعه نمایید. در ضمن این پاسخ‌ها تنها یکی از حالت‌های ممکن نوشتن برنامه هستند و شما ممکن است به شیوه‌ای دیگر برنامه را نوشته باشید و کاملاً هم صحیح باشد. همچنین برای مشاهده توضیحات شفاهی مربوط به این تمرین‌ها، ابتدای ویدئوی جلسه بعد را ببینید (لینک ویدئو در صفحه اول کتاب گذاشته شده است).

« پاسخ تمرین اول:

```
data = []
for i in range(2):
    txt = input('please enter food and calories
               with space:')
    food, calorie = txt.split(' ')
    data.append([food, float(calorie)])
print(data)
```

« پاسخ تمرین دوم:

در ادامه برنامه قبل، می‌توان نوشت:

```
total = 0
for food, calorie in data:
    total += calorie
print(f'The total calories was {total}')
```

اگر در مرحله قبل، کالری را به عدد تبدیل نکرده باشیم، در این مرحله که می‌خواهیم کالری‌ها را جمع کنیم، با مشکل روبرو می‌شویم.

« پاسخ تمرین سوم:

با جستجو کردن در گوگل متوجه می‌شویم کتابخانه‌ای built-in (بدون نیاز به

نصب) در پایتون وجود دارد. یکی از دستورات آن، randint است که عددی تصادفی در محدوده مد نظر ما (مثلاً بین ۱ و ۶) می‌دهد:

```
from random import randint
rolls=[]
for _ in range (10000):
    roll = randint(1,6)
    rolls.append(roll)

count_of_six = rolls.count(6)
print(f'The six number has appered {count_of_six}
times')
```

جالبی این تمرین این است که هرچه تعداد تاس انداختن را بیشتر کنیم، تعداد ۱ها با تعداد ۲ها و ... بسیار نزدیک‌تر می‌شود! این همان مفهوم احتمال است و چقدر زیبا بود که در کلاس ریاضی، مفهوم احتمال را به این شکل به دیگران بیاموزیم. به این شکل که هزاران بار یک سکه را بیندازیم (الآن با پایتون راه آن را بلد هستیم) و ببینیم که چند بار ۰ و چند بار ۱ می‌آید.

سؤال تکمیلی:

کتابخانه‌ی random توابع دیگری هم دارد، مانند choice. با سرچ کردن ببینید آن دستور چه می‌کند؟

« پاسخ تمرین چهارم:

با جستجو کردن در اینترنت متوجه می‌شویم که با دستور list می‌توانیم یک رشته را به لیست تبدیل کنیم:

```
seq = 'ATCTG'
seq_list = list(seq)
```

```
print(seq_list)
```

خروجی:

```
['A', 'T', 'C', 'T', 'G']
```

و با دستور نازیبای join می توان یک لیست را به رشته تبدیل کرد:

```
new_seq = ''.join(seq_list)
```

```
print(new_seq)
```

خروجی:

```
ATCTG
```

حال این کار چه سودی دارد؟ رشته‌ها در پایتون قابل تغییر نیستند، برای مثال نمی‌توانیم عضو دوم را تغییر دهیم یا یک کاراکتر به آن اضافه کنیم، اما لیست‌ها با متدهای مفید قابلیت هرگونه تغییری را به ما می‌دهند. برای مثال ما می‌توانیم یک رشته DNA را از مخاطب گرفته و آن را به لیست تبدیل کنیم:

```
seq = 'ATCTASTARSTAT'
```

```
seq_list = list(seq)
```

و آنگاه در این DNA جهش ایجاد کنیم؛ نوکلئوتیدی را تغییر دهیم یا از آن نوکلئوتید حذف یا اضافه کنیم و ببینیم در پروتئین حاصل از ترجمه این رشته DNA چه تغییری ایجاد می‌شود. برنامه ترجمه پروتئین را در فصل بعد خواهیم نوشت.



اگر در مکانی هستی که همه جا تاریک است،
نشان دهنده این است
که تو حتی یک شمع نیز روشن نکرده‌ای...
عین صاد

• فصل ششم •

لذت دسترسی آنی به اطلاعات مورد نظر با دیکشنری!

دیکشنری چیست و به چه دردی می خورد؟

سلام و درود دوستان! امیدوارم تا الآن از پایتون لذت برده باشید! ما تا حال حاضر با متغیرهایی از نوع `int`، `float` و `str` آشنا شدیم و دیدیم که این متغیرها تنها می توانند یک مقدار را در خود جای دهند. در ادامه با ساختار داده‌ای به نام لیست آشنا شدیم که می توانست از هر جنس داده‌ای در خود جای دهد. امروز با مبحث جذاب دیکشنری آشنا خواهیم شد.

برای این که ببینیم دیکشنری چیست و به چه دردی می خورد، مروری بر مثالی شبیه هفته پیش می کنیم و سپس یک راه جدید را برای انجام آن یاد خواهیم گرفت. حالا فرض کنیم به جای اطلاعات شهرها، اطلاعات بیماران را به ترتیب داریم: نام بیمار، فشارخون و قندخون:

```
data= [
    ['Ali', 13, 200],
    ['Hamed', 12, 130],
    ['Leila', 11, 100]
]
```

حال می‌خواهیم، اطلاعات Hamed را به دست بیاوریم؟ باید چه کنیم؟ باید در این لیست حرکت کنیم. با چه کاری می‌توانستیم روی لیستمان حرکت کنیم؟
دانشجو: با حلقهٔ for استاد!

استاد: بسیار عالی! دیدیم که می‌توانیم به این شکل در لیست حرکت کنیم:
`for row in data:`
و الآن در هر بار حلقه، متغیر ما یعنی row یکی از اعضای لیست می‌شود، مثلاً بار اول می‌شود:

```
[ 'Ali', 13, 200 ]
```

و بار دوم می‌شود:

```
[ 'Hamed', 12, 130 ]
```

و همین‌طور تا آخر! پس می‌توانیم این سه مقدار را در سه متغیر بریزیم که به آن‌ها دسترسی داشته باشیم:

```
name, blood_pressure, blood_suger = row
```

به عبارتی:

```
for row in data:  
    name, blood_pressure, blood_suger = row
```

اما راه راحت‌تری هم داشتیم که نیازی به متغیری مثل row نداشته باشیم. آن راه چه بود؟

دانشجو: یادمه می‌تونستیم مستقیماً در همون حلقه این سه متغیر رو بنویسیم:
`for name, blood_pressure, blood_suger in data:`

استاد: و حالا چگونه چک کنیم که الآن به عضو Hamed رسیده‌ایم یا نه؟

دانشجو: قاعدتاً هر جا شرط نیاز است، نام `if` می‌درخشد! می‌تونیم از `if` استفاده کنیم.

استاد: بسیار عالی! و بگوییم که اگر فشار خون بیشتر از ۱۱.۵ بود، مثلاً نام و اطلاعات فرد را برایمان نمایش بده:

```
if name == 'Hamed':
    print(name, blood_pressure, blood_suger)
```

و به عبارتی کل برنامه ما شد:

```
data= [
    ['Ali', 13, 200],
    ['Hamed', 12, 130],
    ['Leila', 11, 100]
]
```

```
for name, blood_pressure, blood_suger in data:
    if blood_pressure > 11.5:
        print(name, blood_pressure, blood_suger)
```

و چون وقتی به اطلاعات Hamed دسترسی پیدا کردیم، دیگر نیازی نیست که حلقه تا انتها ادامه پیدا کند و وقتمان گرفته شود، می‌توانیم در صورت پیدا کردن Hamed یک `break` بگذاریم تا حلقه قطع شده و ادامه برنامه اجرا شود:

```
for name, blood_pressure, blood_suger in data:
    if name == 'Hamed':
        print(name, blood_pressure, blood_suger)
        break
```

این هم خروجی کد است:

Hamed 12 130

حالا اشکال کار کجاست؟ فرض کنید لیست ما ۱۰۰۰ عضو داشته باشد. باید برای هم چین کار ساده‌ای یعنی خواندن اطلاعات یک فرد، از ابتدا تا انتهای لیست را بررسی کنیم! و تازه چند خط کد مجبور شدیم برای همچنین کار ساده‌ای بنویسیم!

دانشجو: پس چه می‌شه کرد؟

استاد: اینجاست که دیکشنری وارد می‌شود! در دیکشنری معمول که در زندگی استفاده می‌کنیم ما چه چیزی داریم؟ لغت و معنای کلمه:

لغت: معنی

دقیقاً دیکشنری در دنیای پایتون همین‌طور است، منتها ما اینجا key و value داریم:

key: value

برویم و همان مثال قبل را که به صورت لیست تعریف کرده بودیم، به وسیله دیکشنری پیاده کنیم تا دیکشنری را بهتر متوجه شویم. در پایتون برخلاف لیست که با علامت [] تعریف می‌شد، دیکشنری با علامت {} یا همان آکولاد تعریف می‌شود. اسم دیکشنری خود را هم data می‌گذاریم و برای این که فعلاً مثال ما ساده‌تر باشد، تنها فشار خون را می‌نویسیم:

```
data = {'Ali': 13}
```

همان‌طور که می‌بینید keyهای ما اسم افراد و valueهای ما در این مثال، فشار خون هستند. اگر بخواهیم بقیه افراد را هم وارد کنیم، می‌توانیم با علامت کاما، بقیه موارد را نیز وارد کنیم:

```
data = {'Ali': 13, 'Hamed': 12, 'Leila': 11}
```

نکتهٔ جالب ماجرا این است که مانند لیست، اینجا هم می‌توانیم برای خوانایی بیشتر، هر عضو را در یک خط جدید بگذاریم (کلاً در پایتون معمولاً هر جا علامت کاما وجود دارد اگر تمایل داشتیم می‌توانیم برای خوانایی بیشتر با زدن کلید Enter روی صفحه کلید، یک خط جدید ایجاد کنیم):

```
data = { 'Ali': 13,
         'Hamed': 12,
         'Leila': 11 }
```

حالا چطور می‌توانیم برای مثال به اطلاعات حامد دسترسی پیدا کنیم؟ بسیار ساده!

```
data[ 'Hamed' ]
```

آن همه کد که برای لیست نوشتیم، در همین نیم خط کد به همان پاسخ رسیدیم! کافی است نام دیکشنری را نوشته و کلید موردنظر خود را در علامت [] وارد کنیم! برای مثال اگر بخواهیم مقدار فشارخون Ali را روی صفحه چاپ کنیم، کافی است:

```
data[ 'Ali' ]
```

البته همان طور که دیدیم، اگر بخواهیم مقدار را روی صفحه چاپ کنیم، از دستور `print` استفاده می‌کنیم:

```
print(data[ 'Ali' ])
```

دانشجو: چه باحال استاد! یعنی سریعاً به ما می‌گه که فشارخونش چقدره!

استاد: دقیقاً! در اصطلاح علوم کامپیوتر، زمان دسترسی به اطلاعات موردنظر در یک لیست خطی است مثلاً اگر ۱۰۰۰ عضو داشته باشیم، باید در بدترین حالت ۱۰۰۰ بار عملیات انجام دهیم تا به آن عضو برسیم اما دیکشنری‌ها زمان ثابت ۱ یا به عبارتی O(1) دارند؛ یعنی با یک بار عملیات به آن داده می‌رسیم و در نتیجه در این مورد بسیار بهینه هستند! به زبان خودمانی، دیکشنری مثل کسی که قیمت اجناس در مغازه را

حفظ است و اگر بپرسی سریعاً به شما قیمت را می گوید، اما لیست شبیه کسی است که بخواهد کل دفتر مغازه را ورق بزند تا قیمت یک جنس را به شما بگوید.

دانشجو: پس استاد لیست ها رو بندازیم دور و کلاً از دیکشنری ها استفاده کنیم!

استاد: لیست ها و دیکشنری ها هر کدام کاربرد خاص خود را دارند. به عبارتی هر وقت با داده های جدولی سرو کار داشتیم (داده ای داشتیم که می توانستیم در جدول نمایش دهیم) (مانند همین مسئله که داشتیم، اسم و فشار خون)) در صورتی که می خواهیم سریعاً به اطلاعات یک آیتم مورد نظر دسترسی پیدا کنیم، معمولاً دیکشنری ها انتخاب مناسب تری هستند (شکل ۱) و وقتی با لیستی از آیتم ها روبرو هستیم یا تنها می خواهیم اطلاعات را ذخیره کنیم، لیست ها مناسب تر هستند. البته همان طور که در مثال جلسه قبل دیدیم، می توانیم همان کار دیکشنری ها را با لیست ها هم انجام دهیم.

مشارعون ۲۱	
۱۳	Al:
.	.
.	.
.	.

شکل ۱- برای داده های جدولی که نیازی به دسترسی سریع داریم، معمولاً دیکشنری ها انتخاب مناسب تری هستند.

دانشجو: استاد اگر فشار خون یک بیمار رو نداشتیم، می تونیم جلوی اسم بیمار رو

خالی بذاریم؟

استاد: پایتون در این گونه موارد به ما خطا می‌دهد؛ در این صورت می‌توانید به جای مقدار فشارخون، مثلاً None یا هر مقدار دیگری را بنویسید که بعداً خودتان بدانید منظور این است که فشارخون برای این بیمار ثبت نشده است:

```
data = {'Ali': 13,
        'Hamed': None,
        'Leila': 11}
```

خب دوستان! بیاید در محیط کنسول پایتون این موارد را باهم امتحان کنیم که امکانات دیکشنری‌ها را بهتر متوجه شویم. اگر این دستور را بنویسیم به نظر تان چه اتفاقی می‌افتد؟

```
data['Ali'] = 18
```

دانشجو: امتحان کردم، چیزی ننوشت. بذار این محتوای دیکشنری رو بررسی کنم:

```
data
```

چه جالب! مقدار فشارخون علی تغییر کرد!

```
{'Ali': 18, 'Hamed': 12, 'Leila': 11}
```

استاد: دقیقاً! دیکشنری‌ها مانند لیست‌ها، قابل تغییر (اصطلاحاً mutable) هستند و می‌توان مقادیر آن‌ها را پس از ایجاد کردن تغییر داد. حالا این دستور را چک کنید:

```
data['Leila'] = data['Leila'] + 1
```

دانشجو: `data['Leila'] + 1` برابر می‌شه با ۱۲. حالا ما گفتیم که مقدار `data['Leila']` برابر این مقدار قرار بده! چه جالب یعنی با این کار یکی به مقدار فشارخون اضافه کردیم. استاد می‌تونیم همون طوری که تا الان یاد گرفتیم به‌جای دستوری که نوشتید، بنویسیم:

```
data['Leila'] += 1
```

بذارین امتحان کنم... انگار شد!

استاد: دقیقاً! این دو دستور معادل هم هستند و در بسیاری از مواقع می‌توانید از هر کدام که دوست داشتید استفاده کنید، اما اگر جواب نداد یا با خطا روبرو شدید از همان دستور اول که حالت منطقی‌تر است استفاده کنید.

یک بار دیکشنری خودمان را در نظر بگیریم:

```
data = {'Ali': 13,  
        'Hamed': 12,  
        'Leila': 11}
```

به نظرتان اگر بنویسیم:

```
data['Pedram']
```

چه اتفاقی می‌افتد؟

دانشجو: ما در دیکشنری خودمون کلیدی با عنوان Pedram نداریم... نمی‌دونم چی می‌شه! بذارین امتحان کنم... خطا داد استاد! نوشت `!KeyError`

استاد: دقیقاً! چون ما در دیکشنری خود کلیدی به عنوان Pedram نداریم که مقدار آن را به ما نشان دهد. حالا این دستور چطور؟

```
data['Pedram'] = 11
```

دانشجو: این هم باید خطا بده! بذارین امتحان کنم... نه خطا نداد... یعنی چی شده؟ بذارین محتوای دیکشنری رو ببینم:

```
data
```

چه جالب این کلید و مقدار به دیکشنری اضافه شد!

```
{‘Ali’: 13, ‘Hamed’: 12, ‘Leila’: 11, ‘Pedram’: 11}
```

استاد: دقیقاً! با این دستور، اگر این کلید در دیکشنری وجود داشته باشد (مثل دستور `data[‘Ali’] = 18` که کلید `Ali` در دیکشنری وجود داشت) مقدار آن جایگزین می‌شود و اگر کلید وجود نداشته باشد، این کلید و مقدار به دیکشنری اضافه می‌شوند.

متدهای دیکشنری (مثال فشارخون بیماران)

همان‌طور که رشته‌ها و لیست‌ها متد داشتند، دیکشنری‌ها هم متدهای مفیدی دارند. با چه علامتی به متدها دسترسی داشتیم؟

دانشجو: با علامت سؤال؟

استاد: ..!

دانشجو: علامت تعجب!؟

استاد:!

دانشجو: علامت حاکم بزرگ! میتی کومان!!

استاد: از دست تو! با علامت `pop`!!!!!! مت نقطه!!! با نوشتن نقطه می‌توانیم متدهای دیکشنری را ببینیم. برای مثال متد `pop`:

```
data.pop(‘Ali’)
```

همان‌طور که در توضیحات می‌بینید چه می‌کند؟

دانشجو: نوشته یک آیتم رو حذف می‌کنه. بذارین محتوای دیکشنری `data` رو چک کنم:

```
data
```

چه جالب؛ الآن دیگه دیکشنری ما کلیدی به نام Ali رو نداره!
{‘Hamed’: 12, ‘Leila’: 11}

استاد: یا مثلاً متد get. این به نظرتان چه می‌کند؟ مثلاً:
`data.get(‘Hamed’)`

دانشجو: برگردوند ۱۲. استاد این که همون دستور:
`data[‘Hamed’]`

درسته؟ پس به چه دردی می‌خوره؟
استاد: درسته! اما یه نکته ظریف در این متد نهفته است. فرض کنید کلیدی که وجود ندارد، مثلاً Peyman، این دستور را وارد کنیم:
`data[‘Peyman’]`

دانشجو: برنامه خطا داد!
استاد: درسته اما با متد get چطور؟
`data.get(‘Peyman’)`

دانشجو: خطا نداد اما هیچی هم نشون نداد.
استاد: درواقع None برگرداند. می‌توانیم با دستور پرینت این موضوع را چک کنیم:
`print(data.get(‘Peyman’))`

خروجی کد:
None

چرا؟ به توضیحات دستور get دقت کنید؛ همان توضیحاتی که موقع نوشتن به‌عنوان راهنما ظاهر شد یا می‌توانیم خودمان هم دستی بنویسیم:
`help(data.get)`

می‌بینیم که نوشته:

`get(key, default=None, /)` method of `builtins.dict` instance

Return the value for key if key is in the dictionary, else default.

به عبارتی، این تابع، `value` یا همان مقدار متناظر هر کلید را `return` می‌کند، در صورتی که کلید در دیکشنری وجود داشته باشد، در غیر این صورت مقدار پیش‌فرض را برمی‌گرداند. در پرانتز هم می‌بینیم که این متد، یک آرگومان اختیاری به نام `default` دارد که مقدار به‌صورت پیش‌فرض برابر با `None` است. اگر مثلاً بنویسیم:

```
data.get('Peyman', 0)
```

چون این کلید وجود ندارد، به جای خطا دادن و متوقف شدن برنامه، مقدار صفر را برمی‌گرداند. در بعضی مواقع این موضوع بسیار کاربردی می‌شود، مخصوصاً وقتی می‌خواهیم چیزهایی را بشماریم و در صورتی که چیزی وجود نداشت، برنامه خطا ندهد بلکه تعداد آن را ۰ نشان دهد.

به نظر تان متد `clear` چه می‌کند؟

```
data.clear()
```

دانشجو: این جور که از اسمش مشخصه، احتمالاً می‌زنه و دیکشنری رو خالی می‌کنه... بذارین چک کنم...

```
data
```

بله درست فکر کردم:

```
{}
```

استاد: بسیار عالی! و همان‌طور که در لیست‌ها هم داشتیم، اگر می‌خواستیم یک

کپی از لیست یا دیکشنری خود تهیه کنیم که با تغییر یکی، دیگری هم دست نخورد، از متد `copy` استفاده می‌کنیم، مثلاً:

```
data_bak = data.copy()
```

و حواسمان هست که با دستور:

```
data_bak = data
```

درواقع دیکشنری جدیدی ایجاد نمی‌شود، بلکه درواقع این دو دیکشنری، یکی هستند و یا به عبارت تخصصی‌تر به یک مکان در حافظه کامپیوتر اشاره دارند. خلاصه کلام این که اگر می‌خواهید یک کپی از دیکشنری یا لیست تهیه کنید، حتماً از متد `copy` استفاده کنید. من در یکی از برنامه‌هایی که نوشته بودم، به این نکته حواسم نبود و زمان زیادی طول کشید تا بتوانم بفهمم ایراد از همین جاست و هر تغییری که در یکی از این دیکشنری‌ها یا لیست‌ها انجام می‌دادم، در دیگری هم انجام می‌شد، چون از متد `copy` استفاده نکرده بودم.

ما سه متد دیگر داریم. یک‌بار دیکشنری خود که نامش `data` بود را ببینیم:

اولین متد `keys` است:

```
data.keys()
```

که چه می‌کند؟ همان‌طور که مرور کردیم دیکشنری‌ها از `key`ها و `value`ها تشکیل شده‌اند. در نتیجه این دستور به ما `key`های دیکشنری‌مان را برمی‌گرداند، به عبارتی:

```
dict_keys(['Ali', 'Hamed', 'Leila'])
```

اگر بخواهیم این خروجی به لیست تبدیل شود، می‌توانیم از همان دستور معروف `list` استفاده کنیم:

```
list(data.keys())
```

که می‌بینید خروجی آن به شکل یک لیست است:

```
['Ali', 'Hamed', 'Leila']
```

متد `values` چه می‌کند؟ همان‌طور که احتمالاً حدس می‌زنید، `value`های دیکشنری ما را به ما برمی‌گرداند:

```
data.values()
```

خروجی دستور:

```
dict_values([13, 12, 11])
```

دانشجو: قاعداً این رو هم باید بتونیم به شکل لیست تبدیل کنیم. من با اجازهتون امتحان کنم...

```
list(data.values())
```

آره دقیقاً این هم شد یه لیست!

```
[13, 12, 11]
```

استاد: بسیار عالی! حالا سؤال این است: این متدها چه کاربردی دارند؟ دیدیم که دیکشنری به راحتی می‌تواند مثلاً اطلاعات `Ali` را به ما بدهد؛ پس این دستورها به چه دردی می‌خورند؟

فرض کنید می‌خواهیم ببینیم چه کسانی فشارخون بالای ۱۱.۵ دارند و آن‌ها را ببینیم. به عبارتی می‌خواهیم روی کل دیکشنری حرکت کنیم. چطور روی چیزی حرکت کنیم؟

دانشجو: فقط حلقه `for` به ذهنم می‌رسه، اما نمی‌دونم چطور؟

استاد: دقیقاً می‌توانیم از `for` استفاده کنیم. نظرت چیست که روی کلیدها حرکت کنیم، یعنی:

```
for key in data.keys():
```

می‌توانید امتحان کنید و ببینید در هر بار حلقه، متغیر ما یعنی key چه چیزی می‌شود؟

دانشجو: استاد الان امتحان می‌کنم و پرینتش می‌کنم:

```
for key in data.keys():  
    print(key)
```

چه جالب استاد! همون‌طور که انتظار داشتیم، متغیر ما در هر بار حلقه، یکی از کلیدهای دیکشنری ما شد:

```
Ali  
Hamed  
Leila
```

استاد: بسیار عالی! حال که در هر بار حلقه به یکی از کلیدها دسترسی داریم، پس انگار می‌توانیم به مقدار آن هم دسترسی داشته باشیم. چطور می‌توانستیم ببینیم با کلید که اینجا نام بیمار بود، به مقدار آن یعنی فشارخون دسترسی داشته باشیم؟

دانشجو: استاد تا جایی که باهم یاد گرفتیم، با علامت [] یعنی مثلاً:

```
data['Ali']
```

استاد: دقیقاً! و حال متغیر key ما در هر بار حلقه یکی از کلیدها می‌شود، پس می‌توانیم بنویسیم:

```
data[key]
```

که می‌توانیم آن را در یک متغیر، مثلاً blood_pressure ذخیره کنیم و بعد آن را پرینت کنیم یا با آن کار دیگری انجام دهیم:

```
blood_pressure = data[key]
```

به عبارتی کل حلقه ما شد:

```
for key in data.keys():  
    blood_pressure = data[key]  
    print(key, blood_pressure)
```

خروجی برنامه:

```
Ali 13  
Hamed 12  
Leila 11
```

دانشجو: چه خفن استاد!

استاد: جالب تر این که در پایتون اگر همین طور بنویسیم:

```
for i in data:  
    print(i)
```

یعنی مستقیماً روی خود دیکشنری حلقه بزنیم و نه (`data.keys()`) باز هم پایتون به شکل پیش فرض روی کلیدها حلقه می زند:

```
Ali  
Hamed  
Leila
```

دانشجو: پس می تونیم کدی که نوشتیم رو به این شکل هم ساده تر کنیم:

```
for key in data:  
    blood_pressure = data[key]  
    print(key, blood_pressure)
```

استاد: دقیقاً! باز هم پایتون کار را راحت تر کرده است. دیکشنری ها متدی دارند به نام (`items()`). اگر روی آن حلقه بزنیم، هم زمان هم به `key` و هم `value` دسترسی داریم؛ یعنی:

```
for key, value in data.items():  
    print(key, value)
```

خروجی کد:

```
Ali 13  
Hamed 12  
Leila 11
```

دانشجو: این آخری از همه جالب تر بود! استاد یه سؤال! ما از متد `keys()` استفاده کردیم. اگر از `values()` استفاده کنیم و روی اون حلقه بزنیم، هر بار به یکی از مقادیر دسترسی داریم؛ درسته؟ بذارین امتحان کنم:

```
for i in data.values():  
    print(i)
```

آره درست حدس زدم:

```
13  
12  
11
```

استاد: بسیار عالی! حال به سراغ سؤال اصلی مان برویم! چگونه می توانیم افرادی که فشار خون بالاتر از ۱۱.۵ دارند را روی صفحه ببینیم؟

دانشجو: استاد الان بهتون میگم!!! باید روی دیکشنری حرکت کنیم و هر کسی که فشارخونش بالاتر از ۱۱.۵ هست رو چاپ کنیم. برای حلقه زدن روی دیکشنری ۳ تا روش یاد گرفتیم! من آخرینش رو بیشتر خوشم اومد، برای همین همون رو میگم!

```
for key, value in data.items():  
    if value>11.5:  
        print(key, value)
```

خروجی کد:

Ali 13

Hamed 12

که خیلی شیک و مجلسی افراد با فشار خون بالای ۱۱.۵ رو برامون چاپ کرد. استاد ترشی نخورم یه چیزی میشما! 😊

استاد: درود بر تو! فقط نکته این است که بهتر است از اسم‌های متغیر بهتر استفاده کنیم. دیکشنری ما از چه چیزی تشکیل شده؟

دانشجو: کلیدهای اون نام بیمار و مقادیر، فشار خون هستن.

استاد: پس بهتره که به جای key متغیری با نام بهتر مثلاً name و به جای value متغیری با نام واضح‌تر مثلاً blood_pressure در نظر بگیریم:

```
for name, blood_pressure in data.items():  
    if blood_pressure>11.5:  
        print(name, blood_pressure)
```

دانشجو: استااا! گیر می‌دینا!!

استاد: از من به تو نصیحت! کدت را تمیز بنویس!! تا کد تو نیز بعداً به تو روی خوش نشان دهد و راحت‌تر و خواناتر باشد!

حالا که با متد items حلقه زدیم، بیا و یکبار هم با همان روش اول روی دیکشنری حلقه بزنیم، یعنی روی کلیدها:

```
for name in data:  
    blood_pressure = data[name]  
    if blood_pressure>11.5:  
        print(name, blood_pressure)
```

و یادمان هم هست که اگر مستقیم روی خود دیکشنری با حلقه for حرکت کنیم،

پایتون به شکل پیش فرض روی کلیدها حرکت می کند، یعنی برنامه ما معادل برنامه زیر است:

```
for name in data.keys():
    blood_pressure = data[name]
    if blood_pressure>11.5:
        print(name, blood_pressure)
```

دانشجو: ولی اون متد items یه چیز دیگه بود استاد!

استاد: در بسیاری از مواقع همان متد items کار ما را راه می اندازد و از طرفی خوانایی کد را هم بیشتر می کند.

راستی برای اطلاعات بیشترتان، اگر در حلقه روی items() فقط یک متغیر بنویسیم، خروجی کد زیر چیست؟

```
for i in data.items():
    print(i)
```

دانشجو: جالبه استاد... بذارین امتحان کنم؛ ... نوشت:

```
('Ali', 13)
('Hamed', 12)
('Leila', 11)
```

یعنی انگار هر بار i یک چیز دوتاییه... آهان پس برای همین وقتی می نوشتیم:

```
for key, value in data.items()
```

خود پایتون می اومد و key و value رو جایگزین می کرد.

استاد: دقیقاً! حتی اگر باییم با دستور list، خروجی data.items را به لیست تبدیل کنیم:

```
list(data.items())
```

با این خروجی روبرو می‌شویم:

```
[('Ali', 13), ('Hamed', 12), ('Leila', 11)]
```

فعلاً نیاز ما نیست، اما همین‌قدر بدانید که این، لیستی از توپل‌هاست. توپل شبیه همان لیست است منتها به‌جای علامت `[]`، با پرانتز و کاما تعریف می‌شود:

```
t = (1, 2, 3)
```

و اگر با دستور `type`، نوع آن‌ها را چک کنیم:

```
type(t)
```

می‌بینیم که می‌نویسد توپل:

```
tuple
```

و عین لیست‌ها می‌توانیم به اعضای آن دسترسی داشته باشیم:

```
t[0]
```

خروجی:

```
1
```

فقط یک تفاوت بسیار بزرگ با لیست‌ها دارند. توپل‌ها قابل تغییر نیستند (به عبارتی `immutable` هستند) و مثلاً دستور زیر به ما خطا می‌دهد:

```
t[0] = 8
```

دانشجو: پس استاد وقتی که از لیست‌ها محدودترن چه کاربردی دارن؟

استاد: دقیقاً به همین دلیل که قابل تغییر نیستند، یک مزیت بزرگ دارند؛ از نظر حافظه بهینه‌تر هستند و اگر متغیرهای بسیار بزرگی دارید که حافظه زیادی اشغال می‌کنند، تعریف آن‌ها به‌عنوان توپل از لیست‌ها بهینه‌تر است.

خب از بیشتر بدانید بیاییم بیرون و برویم سراغ موضوع بعدی!

دیکشنری‌های پیچیده‌تر (مثال فشارخون و وزن بیماران به صورت هم‌زمان)

یک بار دیگر دیکشنری که داشتیم را ببینیم:

```
data = {'Ali': 13,
        'Hamed': 12,
        'Leila': 11}
```

در اینجا فقط نام بیمار را به عنوان کلید و فشارخون را به عنوان مقدار داریم. اگر خواستیم علاوه بر فشارخون، مثلاً وزن بیمار هم باشد چه کنیم؟ یک بار دیگر به دیکشنری که داشتیم نگاه کنیم. مقدار متناظر با Ali، یک عدد است که فشارخون آن است:

```
'Ali': 13
```

مقدار متناظر Ali خود می‌تواند یک دیکشنری باشد، مثلاً:

```
'Ali': {'blood_pressure': 13, 'weight': 85}
```

و به عبارتی کل دیکشنری ما می‌تواند به این شکل باشد:

```
data = {
    'Ali': {'blood_pressure': 13, 'weight': 85},
    'Hamed': {'blood_pressure': 12, 'weight': 69},
    'Leila': {'blood_pressure': 11, 'weight': 54}
}
```

دانشجو: حالا استاد چطور از این دیکشنری استفاده کنیم؟

استاد: ما در دیکشنری قبلی که داشتیم، اگر می‌نوشتیم:

```
data['Ali']
```

دریافت می‌کردیم:

که همان فشارخون فرد موردنظر بود؛ اما در این دیکشنری جدید که داریم اگر بنویسیم:

```
data['Ali']
```

چه چیزی دریافت می‌کنیم؟

دانشجو: استاد امتحان کردم نوشت:

```
{'blood_pressure': 13, 'weight': 85}
```

استاد: یعنی مقدار data['Ali'] خودش یک دیکشنری است. حال به نظرت خروجی زیر چیست؟

```
data['Ali']['blood_pressure']
```

دانشجو: آهان! فشارخون فرد موردنظر مون می‌شه؛ یعنی:

13

استاد: و اگر بخواهیم به وزن Hamed دسترسی پیدا کنیم چه کنیم؟

دانشجو: همون طور که گفتین مرحله به مرحله پیش میرم... با نوشتن:

```
data['Hamed']
```

به اطلاعات این فرد دسترسی پیدا می‌کنیم:

```
{'blood_pressure': 12, 'weight': 69}
```

پس اگر بنویسم:

```
data['Hamed']['weight']
```

منطقاً باید بتونم به وزن این فرد دسترسی پیدا کنم... آره استاد شد 😊 :

69

استاد: بسیار عالی! پس به صورت خلاصه دیدیم همان طور که می‌توانیم لیستی از

لیست‌ها داشته باشیم، می‌توانیم دیکشنری از دیکشنری‌ها داشته باشیم و یا حتی مثلاً ترکیب این‌ها! مثلاً فرض کنید، لیست نمرات یک فرد می‌تواند به‌عنوان مقدار روبروی نام یک فرد باشد:

```
data = {
    'Ali': [12, 20, 19],
    'Hamed': [13, 18, 12],
    'Leila': [17, 18, 16]
}
```

که اگر الآن بنویسیم:

```
data['Leila']
```

چه دریافت می‌کنیم؟

دانشجو: جالبه، لیست نمرات رو گرفتیم؛

```
[17, 18, 16]
```

استاد: دقیقاً! و حتی می‌توان تمام متدهای لیست‌ها و ... را اجرا کرد، مثلاً:

```
data['Leila'].append(18)
```

دانشجو: چه عجیب استاد! محتوای دیکشنری رو که چک کردم:

```
data
```

دیدم عدد ۱۸ به لیست موردنظر اضافه‌شده:

```
{
    'Ali': [12, 20, 19],
    'Hamed': [13, 18, 12],
    'Leila': [17, 18, 16, 18]
}
```

پس یعنی این لیست‌ها و دیکشنری‌ها همگی قابلیت تغییر رو دارن!

استاد: دقیقاً همین‌طور! الآن فقط برایتان مثال زدم اما در پروژه‌هایی که در ادامه حل می‌کنیم از این امکانات استفاده خواهیم کرد. حالا انشا... وقتی که وارد کتابخانه پانداز شدیم می‌بینیم که چقدر راحت‌تر می‌توانیم داده‌ها را بررسی کنیم و مانند اکسل داده‌ها زیر دستانمان باشد!

یک مثال اپدمیولوژی!

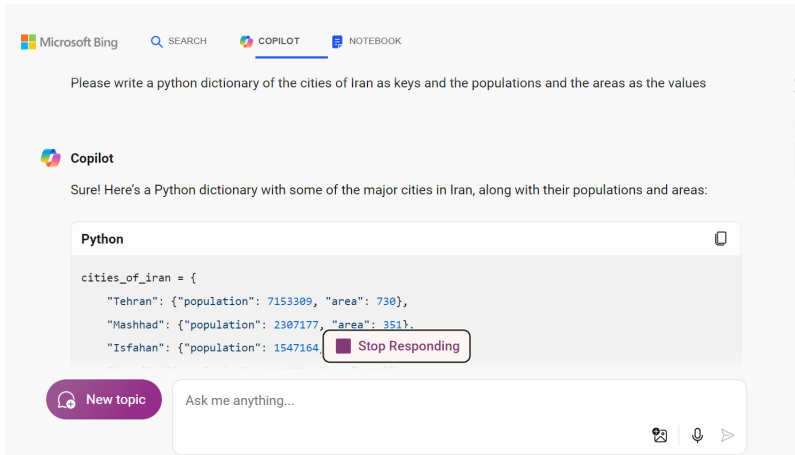
خب برویم یک مثال دیگر بزنیم! برای این کار از هوش مصنوعی چت‌جی‌پی‌تی یا بینگ کمک می‌گیریم. متأسفانه چت‌جی‌پی‌تی تحریم است و نیاز به استفاده از تحریم‌شکن داریم اما بینگ را می‌توان بدون تحریم‌شکن هم باز کرد. در رابطه با استفاده از این چت‌بات‌های هوش مصنوعی، من یک وبینار رایگان ۳ ساعته دارم که توصیه می‌کنم حتماً آن را ببینید و از این ابزار فوق‌العاده در زندگی و کار خود استفاده کنید! کافی است روی این لینک کلیک کرده یا بارکد زیر را اسکن کنید (شکل ۲):



شکل ۲- می‌توانید در گوگل جستجو کنید «کارگاه استفاده از چت‌بات‌های هوش مصنوعی chatGPT و Copilot و gemeni پدram پربها» یا این که بارکد بالا را اسکن کنید.

خب می‌خواهم چه کار کنم؟ ما بعداً یاد خواهیم گرفت که با فایل‌ها کار کنیم یا داده را از یک جدول در وب‌سایت استخراج کنیم، اما فعلاً برای این که یک تمرین واقعی انجام دهیم، من از هوش مصنوعی خواستم که یک دیکشنری پایتون از شهرها

به عنوان کلیدها و جمعیت و مساحت را به عنوان value دیکشنری برای من بنویسد:
Please write a python dictionary of the cities of Iran
as keys and the populations and the areas as the values
(شکل ۳):



شکل ۳- دیکشنری گرفته شده از چت بات هوش مصنوعی (برای مثال من از هوش مصنوعی بینگ استفاده کردم).

و هوش مصنوعی هم این دیکشنری را به من داد که کافی است آن را در کد خودم
کپی کنم 😊 :

```
cities = {  
    "Tehran": {"population": 7153309, "area": 730},  
    "Mashhad": {"population": 2307177, "area": 351},  
    "Isfahan": {"population": 1547164, "area": 551},  
    "Karaj": {"population": 1448075, "area": 162},  
    "Tabriz": {"population": 1424641, "area": 324},  
    "Shiraz": {"population": 1249942, "area": 240},
```

```
"Qom": {"population": 900000, "area": 74},  
"Ahvaz": {"population": 841145, "area": 185},  
"Kermanshah": {"population": 621100, "area": 93},  
"Urmia": {"population": 577307, "area": 81}  
}
```

حالا می‌خواهیم شهرهایی که با مساحت بیشتر از ۵۰۰ هستند را پیدا کنیم. باید چه کنیم؟

دانشجو: استاد پیچیده شد!!

استاد: هر وقت پیچیده شد، چه کار می‌کنیم؟

دانشجو: مسئله رو مرحله به مرحله پیش می‌بریم. الآن من فقط می‌دونم که لازمه روی کل دیکشنری حرکت کنم و می‌دونم با حلقهٔ `for` می‌تونم این کار رو انجام بدم! استاد: عالیه، همین‌طور پیش برو و همین روند به‌مرور از سردرگمی ما را بیرون می‌آورد!

دانشجو: چشم حتماً! اون جواری که یاد گرفتم، اگه مستقیماً از حلقهٔ `for` روی دیکشنری استفاده کنم، روی کلیدهای دیکشنری حرکت می‌کنه؛ پس:

```
for city in cities:  
    print(city)
```

هر بار به نام یکی از شهرها دسترسی پیدا می‌کنم، یعنی:

Tehran
Mashhad
Isfahan
...

و تا انتها...

استاد: حالا برای مثال چطور می‌توانیم به اطلاعات شهر Shiraz دسترسی پیدا کنیم؟

دانشجو: تا جایی که یاد گرفتیم بعد از نام دیکشنری در علامت [] می‌نوشتیم یعنی:

```
cities['Shiraz']
```

که این مثل قبل به ما یک دیکشنری رو می‌ده. بذارین این کد رو جداگونه در محیط کنسول بنویسم. آهان شد:

```
{'population': 1249942, 'area': 240}
```

پس اگر بخوام به مساحت اون دسترسی پیدا کنم، می‌تونم بنویسم:

```
cities['Shiraz']['area']
```

درست شد، نوشت:

```
240
```

استاد: حالا به حلقه خودمان برگردیم:

```
for city in cities:  
    print(city)
```

به من گفتم که city هر بار نام یکی از شهرها می‌شود... چطور می‌توانیم مساحت آن شهر را ببینیم؟

دانشجو: این دستور اطلاعات شیراز رو به ما می‌داد...

```
cities['Shiraz']['area']
```

و الان در متغیر city در هربار حلقه نام شهر رو داریم... پس ... آهان ... می‌تونیم بنویسیم:

```
cities[city]['area']
```

که مثلاً در بار اول حلقه چون city برابر با Tehran است، انگار این دستور برابر این دستوره:

```
cities['Tehran']['area']
```

و بار بعدی چون city می‌شه برابر با شهر بعدی یعنی Mashhad، انگار این دستوره:

```
cities['Mashhad']['area']
```

و تا آخر! پس می‌تونیم برای مثال این مقدار رو در یک متغیر، مثلاً area بریزیم:

```
area = cities[city]['area']
```

و بعد نام شهر و مساحت رو پرینت کنیم، یعنی برنامه ما می‌شه:

```
for city in cities:
    area = cities[city]['area']
    print(city, area)
```

خروجی برنامه:

```
Tehran 730
Mashhad 351
Isfahan 551
Karaj 162
Tabriz 324
...
```

استاد: و حالا چطور می‌تونیم کاری کنیم که فقط شهرهایی که مساحت بیشتر از ۵۰۰ را دارند چاپ شوند؟

دانشجو: می‌تونیم از شرط استفاده کنیم، یعنی:

```
for city in cities:
    area = cities[city]['area']
```



```
if area>500:  
    print(city, area)
```

چه جالب! شهرهایی با جمعیت بالاتر از ۵۰۰ چاپ شد که دو تا شهر بیشتر نبودن:
Tehran 730
Isfahan 551

استاد: ما با لیست‌ها آشنا شدیم! نظرت چیست که نام این شهرها را به یک لیست مثلاً `result` اضافه کنیم؟

دانشجو: بله استاد حتماً! یک لیست خالی به نام `result` اول کار تعریف می‌کنم:
`result = []`

و در حلقه هربار که به یه شهر با مساحت بالاتر از ۵۰۰ رسیدم، به جای چاپ کردن اون، نام شهر و مساحت اون رو به لیست اضافه می‌کنم (شبيه همون کاری که در جلسه قبل کردیم):

```
result.append([city, area])
```

آهان یادم رفت! آخر کار که باید نتیجه رو به کاربر نشون بدم، می‌تونم لیست نهایی رو پرینت کنم؛ پس برنامه ما شد:

```
result = []  
for city in cities:  
    area = cities[city]['area']  
    if area>500:  
        result.append([city, area])  
print(result)
```

خروجی برنامه:

```
[[ 'Tehran', 730], [ 'Isfahan', 551]]
```

استاد: بسیار عالی! دیدی که با خرد کردن مسئله به مسئله‌های کوچک‌تر، خیلی از مواقع برنامه‌های پیچیده را می‌توانیم بنویسیم. من خودم زمان‌هایی بوده است که به‌شدت سردرگم بودم، ولی وقتی از یک جای کوچک شروع کردم و با خودم گفتم فرض کنم که برنامه را می‌خواهم فقط برای یک شهر بنویسم، انگار مسیر روشن‌تر شده و بهتر توانستم کل برنامه را بنویسم.

سؤال آخر! نظرت چیست همین برنامه را با همان متد `items` بنویسیم؟ یادمان هست که با متد `items` در هربار حلقه می‌توانستیم هم به `key` و هم به `value` دسترسی داشته باشیم:

```
for key, value in cities.items():
```

خب اینجا که `key` ما همان نام شهر است پس بهتر است از نام متغیر مناسب مثل `city` استفاده کنیم، اما چون `value` خودش یک دیکشنری است بهتر است همین نام را نگه داریم:

```
for city, value in cities.items():
```

حال نظرت چیست مقدار `city` و `value` را پرینت کنیم تا ببینیم چه چیزی داریم: دانشجو: چشم استاد:

```
for city, value in cities.items():
    print(city, value)
```

همان‌طور که پیش‌بینی می‌کردیم، `value` خودش یک دیکشنری است:

```
Tehran {'population': 7153309, 'area': 730}
Mashhad {'population': 2307177, 'area': 351}
Isfahan {'population': 1547164, 'area': 551}
....
```

استاد: حالا چطور می‌توانیم به مقدار مساحت هر شهر دسترسی داشته باشیم؟ قبول

داریم که مقدار:

```
value['area']
```

همان مساحت شهر است؟ پس می‌توانیم آن را در متغیری به نام مثلاً `area` بریزیم:

```
area = value['area']
```

و برنامه را به این شکل تکمیل کنیم:

```
result = []
for city, value in cities.items():
    area = value['area']
    if area > 500:
        result.append([city, area])
print(result)
```

دانشجو: این جوری هم جالب شد! البته من مورد قبلی رو بیشتر پسندیدم.

استاد: مهم این است که ببینیم در هر پروژه، کدام روش برای ما خواناتر است و از همان روش بهره ببریم.

حالا که جلسه تمام شد، وقت چیست؟

دانشجو: وقت خدا حافظی؟

استاد: بحث شیرین‌ترین‌ها!!!

تمرین‌های جلسه ششم

در تمرین هر درس ممکن است که یک نکته جدید هم آموزش داده شود، پس حتماً توصیه می‌کنم آن‌ها را ببینید و انجامشان دهید. تمرین‌ها و به‌خصوص خطاهایی که دریافت می‌کنید و آزمون و خطاها (و همچنین سرچ‌هایی که در صورت نیاز انجام می‌دهید)، بسیار بسیار به روند یادگیری برنامه‌نویسی شما کمک می‌کند. پاسخ نمونه برای تمرین‌ها نیز پس از آن‌ها قرار داده شده است.

تمرین‌ها

- ۱- دیکشنری‌ها به نظر تان چه مزیت و چه معایبی نسبت به لیست‌ها دارند؟ در چه مواقعی از آن‌ها استفاده می‌کنید؟ (به این سؤال شفاهاً پاسخ دهید 😊)
- ۲- همان‌طور که احتمالاً بدانید، رشته‌های پروتئین از واحدهایی سازنده به نام آمینواسید ساخته شده‌اند که برای سادگی بیشتر، هر آمینواسید را با یک حرف نشان می‌دهند؛ مثلاً رشته AMV از سه آمینواسید آلانین، متیونین و والین تشکیل شده است که هر کدام جرم خود را دارند. جرم کل پروتئین، مجموع جرم آمینواسیدهای سازنده آن است. برنامه‌ای بنویسید که یک رشته پروتئین را دریافت کرده و وزن مولکولی آن را حساب کند برای مثال:

AVAMMV => 710

(راهنمایی: چون هنوز کار با فایل‌ها و جداول را آغاز نکرده‌ایم، می‌توانید از یکی از چت‌بات‌های هوش مصنوعی بخواهید که یک دیکشنری به شما از آمینواسیدها و وزن مولکولی‌های آن بدهد و از آن استفاده کنید).

- ۳- برنامه‌ای بنویسید که در هر بار ورودی، نام فرد و فشارخون را پرسیده و سپس یک دیکشنری ایجاد کند که بتوان از آن اطلاعات لازم را استخراج کرد.
- ۴- برنامه‌ای بنویسید که در هر بار ورودی نام فرد، فشارخون و قندخون را پرسیده و سپس یک دیکشنری ایجاد کند که بتوان از آن اطلاعات لازم را استخراج کرد.
- ۵- یک برنامه بنویسید که از دیکشنری حاصل از سؤال قبل استفاده کرده و برای مثال بیماران با فشارخون بالای ۱۴ را نشان دهد.
- ۶- مشابه تمرین ۲ یا ۳، یک تمرین در رشته یا حرفه خود منتها کاربردی، طراحی و اجرا کنید 😊.

پاسخ تمرین‌های جلسه ششم

اکیداً توصیه می‌کنم که اول خودتان تمرین‌ها را حل کنید و سپس به پاسخ‌ها مراجعه کنید. در ضمن این پاسخ‌ها تنها یکی از حالت‌های ممکن نوشتن برنامه هستند و شما ممکن است به شیوه‌ای دیگر برنامه را نوشته باشید و کاملاً صحیح هم باشد. همچنین برای مشاهده توضیحات شفاهی مربوط به این تمرین‌ها، ابتدای ویدئوی جلسه بعد را ببینید (لینک ویدئو در صفحه اول کتاب گذاشته شده است):

« پاسخ تمرین اول:

همان‌طور که در طول درس دیدیم، اگر نیاز داشتیم که سریعاً به اطلاعات خاصی از یک نفر یا یک چیز دسترسی پیدا کنیم، دیکشنری‌ها بسیار مناسب هستند، اما اگر نیاز داشتیم که سریعاً لیستی از چیزها را داشته باشیم، لیست‌ها انتخاب مناسب‌تری هستند.

« پاسخ تمرین دوم:

ابتدا نیاز به یک دیکشنری از آمینواسید و وزن مولکولی آن‌ها داریم. چون فعلاً هنوز خواندن فایل یا جدول‌ها را با پایتون کار نکرده‌ایم، از یکی از چت‌بات‌های هوش مصنوعی کمک می‌گیریم؛ برای مثال من از هوش مصنوعی بینگ خواستم که یک دیکشنری از نام تک‌حرفی آمینواسیدها به‌عنوان کلید و وزن مولکولی آن‌ها به‌عنوان مقادیر به من بدهد. به زبان فارسی هم می‌توانیم اما من حتی الامکان حتی اگر با گرامر بد هم بنویسم، ترجیح می‌دهم این است که به زبان انگلیسی این کار را انجام دهم زیرا این کار باعث می‌شود روز به روز زبان ما بهتر شود:

Please make a python dictionary of one letter aminoacids as the keys and their Molecular mass as the values.

و هوش مصنوعی هم این دیکشنری را به من داد (جالب است که حتی به‌صورت توضیح بعد از آیتم، نام کامل آن آمینواسید را هم به‌صورت comment آورده. می‌دانیم که در پایتون مواردی که بعد از علامت # باشند، به‌وسیله کامپیوتر خوانده نمی‌شوند و تنها به‌عنوان یک توضیح برای خواننده در نظر گرفته می‌شوند):

```
amino_acid_masses = {
    'A': 89,      # Alanine
    'R': 174,     # Arginine
    'N': 132,     # Asparagine
    'D': 133,     # Aspartic acid
    'C': 121,     # Cysteine
    'E': 147,     # Glutamic acid
    'Q': 146,     # Glutamine
    'G': 75,      # Glycine
    'H': 155,     # Histidine
```

```

'I': 131, # Isoleucine
'L': 131, # Leucine
'K': 146, # Lysine
'M': 149, # Methionine
'F': 165, # Phenylalanine
'P': 115, # Proline
'S': 105, # Serine
'T': 119, # Threonine
'W': 204, # Tryptophan
'Y': 181, # Tyrosine
'V': 117  # Valine
}

```

حال فرض کنیم رشته پروتئین ما مثلاً این رشته است که آن را در متغیر seq ذخیره کردیم:

```
seq = 'AVAMMV'
```

حالا تنها کاری که لازم است انجام دهیم این است که روی این رشته حرکت کنیم و وزن هر آمینواسید را در دیکشنری بخوانیم و به وزن کل اضافه کنیم:

```
seq = 'AVAMMV'
```

```
total=0
```

```
for aa in seq:
```

```
    mw = amino_acid_masses[aa]
```

```
    total += mw
```

```
print(f'The total MW of your seq is {total}')
```

خروجی برنامه:

The total MW of your seq is 710

« پاسخ تمرین سوم:

این تمرین، مشابه تمرین کالری غذا در جلسه قبل است، منتها در جلسه قبل اطلاعات را در نهایت در یک لیست ذخیره می کردیم، اما این بار اطلاعات را به دیکشنری خود اضافه می کنیم.

پس ابتدا یک دیکشنری خالی به نام مثلاً `data` ایجاد می کنیم:

```
data = {}
```

سپس لازم است که اطلاعات فرد را بگیریم؛ مثلاً می توانیم بگوییم نام و فشار خون را با یک فاصله وارد کند:

```
txt = input('Please input the name and bp with space:\n')
```

و سپس آن را با متد `split` به لیست تبدیل کنیم:

```
name, pressure = txt.split()
```

در نهایت این فرد جدید را به دیکشنری اضافه کنیم.

یادمان است که می توانیم با دستور:

```
data[name] = 14
```

این فرد جدید را به دیکشنری اضافه کنیم:

```
data[name] = pressure
```

البته چون می دانیم که متد `split` به ما رشته تحویل می دهد نه عدد، بهتر است که `float` این مقادیر را به دیکشنری اضافه کنیم، یعنی:

```
data[name] = float(pressure)
```

حال که برنامه خود را نوشتیم، کافی است آن را در یک حلقه بگذاریم که سه بار این

برنامه تکرار شود، پس برنامه ما در نهایت شد:

```
data = {}
for _ in range(3):
    txt = input('Please input the name and bp with
                space:\n')
    name, pressure = txt.split()
    data[name] = float(pressure)
print(data)
```

خروجی برنامه:

```
{'Ali': 12.0, 'Pedram': 11.0, 'Reza': 15.0}
```

« پاسخ تمرین چهارم:

این تمرین، مشابه تمرین قبل است اما با این تفاوت که می‌خواهیم هر بیمار، هم‌قدن خون و هم فشارخون را داشته باشد؛ به عبارتی به دنبال همچنین ساختاری هستیم:

```
data = {
    'Ali': {'bp':12, bs:130},
    'Reza': {'bp':14, bs:110}
}
```

پس ابتدا یک دیکشنری خالی مثلاً به نام data ایجاد می‌کنیم:

```
data = {}
```

سپس لازم است که اطلاعات فرد را بگیریم؛ مثلاً می‌توانیم بگوییم نام، فشارخون و قندخون را با یک فاصله وارد کند:

```
txt = input('Please input the name, bp and bs with
            space:\n')
```

و سپس آن را با متد split به لیست تبدیل کنیم:

```
name, pressure, sugae = txt.split()
```

درنهایت این فرد جدید را به دیکشنری اضافه کنیم.

یادمان است که می‌توانیم با دستور:

```
data[name] = 14
```

این فرد جدید را به دیکشنری اضافه کنیم، منتها اینجا مقدار ما خودش یک دیکشنری جدید است، یعنی:

```
data[name] = {'bp':pressure, 'bs':sugar}
```

به عبارتی یک دیکشنری از دیکشنری‌ها داریم. البته چون می‌دانیم که متد `split` به ما رشته تحویل می‌دهد نه عدد، بهتر است که `float` این مقادیر را به دیکشنری اضافه کنیم، یعنی:

```
data[name] = {'bp':float(bp), 'bs':float(bs)}
```

حال که برنامه خود را نوشتیم، کافی است آن را در یک حلقه بگذاریم که سه بار این برنامه تکرار شود، پس برنامه ما درنهایت شد:

```
data = {}
for _ in range(3):
    txt = input('Please input the name, bp and bs
    with space:\n')
    name, pressure, sugar = txt.split()
    data[name] = {'bp':float(pressure), 'bs':float(sugar)}
```

```
print(data)
```

خروجی برنامه:

```
{
```

```
'Ali': {'bp': 12.0, 'bs': 130.0},  
'Pedram': {'bp': 12.0, 'bs': 100.0},  
'Reza': {'bp': 18.0, 'bs': 150.0}  
}
```

« پاسخ تمرین پنجم:

می‌توانیم این برنامه را در ادامه برنامه بعد بنویسیم. الآن یک دیکشنری به نام `data` داریم که تمامی اطلاعات لازم را دارد. کافی است که با حلقه `for` روی آن حرکت کنیم و بیمارانی که فشارخون بالاتر از ۱۴ را نشان دهد:

```
data = {  
    'Ali': {'bp': 12.0, 'bs': 130.0},  
    'Pedram': {'bp': 12.0, 'bs': 100.0},  
    'Reza': {'bp': 18.0, 'bs': 150.0}  
}
```

```
for name, value in data.items():  
    pressure = value['bp']  
    sugar = value['bs']  
    if pressure > 14:  
        print(name, pressure, sugar)
```

خروجی برنامه (تنها Reza فشارخون بالای ۱۴ داشت):

```
Reza 18.0 150.0
```

« پاسخ تمرین ششم:

برای مثال برنامه‌ای که میزان بیشینه دوز مجاز داروها را در قالب یک دیکشنری دارد و از مخاطب، دارو و دوز را می‌پرسد و در صورتی که دوز از حد مجاز بیشتر بود به مخاطب هشدار می‌دهد:

```
max_doses = {
    "Aspirin": 4000,      # mg/day
    "Ibuprofen": 3200,   # mg/day
    "Paracetamol": 4000, # mg/day
    "Amoxicillin": 3000, # mg/day
    "Metformin": 2000,   # mg/day
    "Atorvastatin": 80,  # mg/day
    "Lisinopril": 40,    # mg/day
    "Omeprazole": 40,    # mg/day
    "Simvastatin": 40,   # mg/day
    "Levothyroxine": 200 # mcg/day
}

txt = input('Please input administered drug and its
dose mg/kg with a space:\n')

drug, dose = txt.split()
dose = float(dose)

max_dose = max_doses[drug]

if dose > max_dose:
    print(f'Warning! Max dose is {max_dose} and
your dose {dose} is higher than it!')
else:
    print(f'The dose {dose} is lower than the Max
dose {max_dose}')
```

منتها مشکلی وجود دارد! اگر داروی وارد شده در دیکشنری ما وجود نداشته باشد در خط زیر با خطا روبرو می‌شویم:

```
max_dose = max_doses[drug]
```

می‌توانیم برای حل این مشکل دو کار انجام دهیم:

یکی این که قبل از اجرای این دستور چک کنیم که ببینیم آیا داروی ما در دیکشنری وجود دارد یا نه:

```
if drug in max_doses:
    max_dose = max_doses[drug]
```

راه دوم این است که از متد `get` استفاده کنیم. این متد در صورتی که یک کلید در دیکشنری وجود نداشته باشد، خطا نمی‌دهد، بلکه مقدار `None` را برمی‌گرداند. پس می‌توانیم چک کنیم که اگر مقدار خروجی برابر `None` بود برای ما بنویسید که این دارو در دیکشنری ما وجود ندارد:

```
max_dose = max_doses.get(drug)
if max_dose is None:
    print(f'The drug you entered {drug} is unknown!')
```

و برنامه ما به این شکل دربیاید:

```
max_doses = {
    "Aspirin": 4000,      # mg/day
    "Ibuprofen": 3200,   # mg/day
    "Paracetamol": 4000, # mg/day
    "Amoxicillin": 3000, # mg/day
    "Metformin": 2000,   # mg/day
    "Atorvastatin": 80,  # mg/day
    "Lisinopril": 40,    # mg/day
```

```

    "Omeprazole": 40,      # mg/day
    "Simvastatin": 40,     # mg/day
    "Levothyroxine": 200   # mcg/day
}

txt = input('Please input administered drug and its
dose mg/kg with a space:\n')
drug, dose = txt.split()
dose = float(dose)

max_dose = max_doses.get(drug)

if max_dose is None:
    print(f'The drug you entered {drug} is un-
known!')

else:
    if dose > max_dose:
        print(f'Warning! Max dose is {max_dose}
and your dose {dose} is higher than it!')
    else:
        print(f'The dose {dose} is lower than
the Max dose {max_dose}')

```

البته یک مشکل دیگر هم وجود دارد! ممکن است مخاطب نام دارو را به جای Aspirin، با حرف کوچک یعنی aspirin وارد کند. برای این کار می‌توانیم با متد title که اولین حرف کلمات را بزرگ می‌کند، نام دارو را استاندارد کنیم:

```

drug, dose = txt.split()
drug = drug.title()

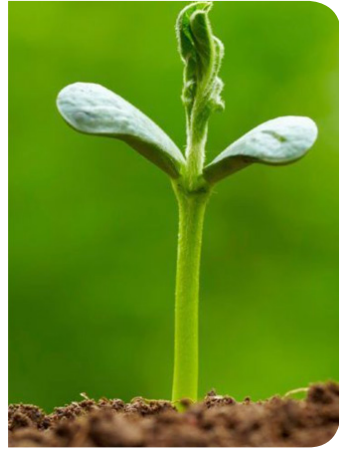
```

کسانی که سر به بالا دارند و حرکتی متعالی را می‌خواهند، مثل سنگ نیستند تا با هل دادن به جایی برسند؛ چرا که وقتی این نیرو تمام شد، سقوط میکنند که سقوط، یک اصل طبیعی برای سنگ‌هاست؛ ولی اگر ریشه‌ها در انسان جان گرفت آنچه برای همه غیرطبیعی است، طبیعی‌ترین خواهد بود که

حرکت برای خلاف جریان طبیعی سنگ‌ها، یک اصل طبیعی برای حرکت گیاهان است.

کتاب حرکت

علی صفایی (عین صاد)



• فصل هفتم •

لذت خروج از مرزهای پایتون با خواندن فایل‌ها!

خواندن و نوشتن فایل‌ها با پایتون برای ما چه کاربردی دارد؟

سلام و درود دوستان! امیدوارم زندگی‌تان پر از معنا باشد. تا این جلسه در مورد ساختار داده‌های لیست و دیکشنری باهم کار کردیم، اما تا الآن یا باید با دست دیکشنری خود را تایپ می‌کردیم یا از دوستی به نام چت‌جی‌پی‌تی (یا هر چت‌بات هوش مصنوعی دیگری) کمک می‌گرفتیم. حالا اگر می‌خواستیم یک دیکشنری متشکل از هزاران بیمار یا هزاران دارو یا ژن یا ... داشته باشیم، چه کنیم؟ با دست تایپ کنیم؟ قطعاً نه!

در اینجا چیزی وجود دارد که می‌تواند کار ما را بسیار راحت‌تر کند: خواندن فایل‌ها! کافی است که فایل اکسل یا متنی یا ... حاوی داده‌های خود را داشته باشیم و با پایتون آن را بخوانیم. از طرفی فایل‌ها باعث می‌شوند، از مرزهای پایتون فراتر بروید. چگونه؟ می‌توانید یک کار را با نرم‌افزار دیگر انجام دهید، سپس آن را به صورت فایل ذخیره کنید و آن فایل را در پایتون باز کرده و ادامه مراحل را در آن انجام دهید و دوباره نتایج حاصل از کار در پایتون را به صورت فایل ذخیره کنید و مثلاً ادامه کار را در نرم‌افزار دیگری پیش ببرید. مثلاً من خودم زیاد از این کار استفاده می‌کنم؛ برای مثال یک

فایل اکسل داشتم که فایل کثیفی بود و پر از کاراکترهای اضافه. من این فایل را با پایتون خواندم و با کتابخانه‌های قدرتمندی که داشت آن را تمیز و دوباره به صورت فایل اکسل ذخیره کردم. مثالی دیگر، همان طور که در فصل اول دیدیم، مثلاً هزاران فایل PDF دارم و می‌خواهم با پایتون تک به تک این فایل‌ها را خوانده و جدول‌های موجود در این فایل‌ها را در یک فایل اکسل وارد کنم. من می‌توانم این فایل‌های PDF را در پایتون بخوانم و خروجی را به صورت یک اکسل ذخیره کنم و در وقت و انرژی خودم صرفه جویی بسیار زیاد کنم (کاری که روزها وقت می‌گرفت را با پایتون در چند ساعت انجام دهم).

برای خواندن یا نوشتن فایل‌ها در پایتون، ما یک روش پایه داریم که می‌توانیم تمامی فایل‌ها را با آن باز کنیم؛ اما توجه داشته باشید که بعداً با کتابخانه‌ای مثل پانداز می‌توانیم به راحتی در یک خط، کد فایل اکسل را بخوانیم و هر کاری که می‌خواهیم با آن انجام دهد.

دانشجو: پس استاد چرا همین اول کار سراغ این کتابخانه پانداز نمی‌ریم؟

استاد: برای این که این روش درست است که شاید چند خط کد بیشتر نیاز داشته باشد؛ اما روش پایه‌ای است که در خیلی از جاها می‌تواند به شما درک بیشتری بدهد. همچنین در بعضی مواقع ممکن است پانداز نتواند به راحتی کاری که شما می‌خواهید را برایتان انجام دهد و یا فایل را به صورتی که شما می‌خواهید برایتان ایجاد کند. در این مواقع استفاده از روش پایه نوشتن فایل بسیار راحت تر است. در کل همان روش پایه هم روش سختی نیست اما این را بدانید که بعداً با کتابخانه پانداز کارمان بسیار بسیار راحت تر می‌شود و در عین حال سعی کنید مواردی که در این جلسه در رابطه با روش پایه یاد می‌گیریم را هم در گوشه ذهنتان داشته و تمرین کنید.

شروع یک مثال کاربردی! آمینواسیدها!

برای این که خواندن و نوشتن فایل‌ها با پایتون را شروع کنیم، برویم به سراغ همان تمرین جلسه قبل که وزن کل یک رشته پروتئینی را از مجموع تک‌تک آمینواسیدهای تشکیل دهنده به دست آوردیم. در آنجا از هوش مصنوعی بینگ کمک گرفتیم و دیکشنری وزن مولکولی آمینواسیدها را ایجاد کردیم؛ اما در اینجا می‌خواهیم این دیکشنری را خودمان ایجاد کنیم.

برای این کار کافیست در اینترنت جستجو کنیم (برای مثال aminoacid tables). اگر یک فایل اکسل پیدا کردیم که چقدر خوب، اگر هم پیدا نکردیم بلکه یک سایت پیدا کردیم که در قالب جدول این اطلاعات را گزارش کرده باشد، باز هم فرقی ندارد. در هر دو حالت می‌توانیم از پایتون کمک بگیریم.

خب بگذارید من در گوگل جستجو کنم... این لینک اول خیلی خوب نیست... آهان این لینک، لینک خوبی بود. عنوان سایت آن یک عنوان ژاپنی است و عنوان صفحه آن، نوشته شده (شکل ۱):

amino acid residues molecular masses



理化学研究所

<http://www2.riken.jp/BiomolChar/Aminoacidmolecul...>

amino acid residues molecular masses

amino acid residues molecular masses*. Residue, 3-letter, 1-letter, Formula, Average mass, Monoisotopic mass. Glycine, Gly, G, C2H3NO, 57.051, 57.02146372.

شکل ۱ - یکی از سایت‌ها که جدول مطلوب ما را دارد.

اگر نتوانستید با جستجو آن را پیدا کنید، کافی است روی این لینک کلیک کنید یا

بارکد زیر را اسکن نمایید (شکل ۲):



شکل ۲- سایت موردنظر

وقتی لینک را باز می کنیم، می بینیم که این جدول را در خود قرار داده است (شکل ۳):



amino acid residues molecular masses*

Residue	3-letter	1-letter	Formula	Average mass	Monoisotopic mass
Glycine	Gly	G	C ₂ H ₃ NO	57.051	57.02146372
Alanine	Ala	A	C ₃ H ₅ NO	71.078	71.03711378
Serine	Ser	S	C ₃ H ₅ NO ₂	87.077	87.03202840
Proline	Pro	P	C ₅ H ₇ NO	97.115	97.05276385
Valine	Val	V	C ₅ H ₉ NO	99.131	99.06841391
Threonine	Thr	T	C ₄ H ₇ NO ₂	101.104	101.04767847
Cysteine	Cys	C	C ₃ H ₅ NOS	103.143	103.00918478
Isoleucine	Ile	I	C ₆ H ₁₁ NO	113.158	113.08406398
Leucine	Leu	L	C ₆ H ₁₁ NO	113.158	113.08406398
Asparagine	Asn	N	C ₄ H ₆ N ₂ O ₂	114.103	114.04292744
Aspartic acid	Asp	D	C ₄ H ₅ NO ₃	115.087	115.02694302
Glutamine	Gln	Q	C ₅ H ₈ N ₂ O ₂	128.129	128.05857751
Lysine	Lys	K	C ₆ H ₁₂ N ₂ O	128.172	128.09496301
Glutamic acid	Glu	E	C ₅ H ₇ NO ₃	129.114	129.04259309
Methionine	Met	M	C ₅ H ₉ NOS	131.196	131.04048491
Histidine	His	H	C ₆ H ₇ N ₃ O	137.139	137.05891186
Phenylalanine	Phe	F	C ₉ H ₉ NO	147.174	147.06841391
Arginine	Arg	R	C ₆ H ₁₂ N ₄ O	156.186	156.10111102
Tyrosine	Tyr	Y	C ₉ H ₉ NO ₂	163.173	163.06332853
Tryptophan	Trp	W	C ₁₁ H ₁₀ N ₂ O	186.210	186.07931295

شکل ۳- جدول موجود در سایت موردنظر

البته هیچ فرقی ندارد از کدام سایت استفاده می کنید اما چون این سایت اطلاعات

آمینواسیدها را به صورت کامل نوشته بود، آن را انتخاب کردیم.

دانشجو: خب استاد این جدول در این سایت رو چطوری می تونیم وارد پایتون کنیم؟

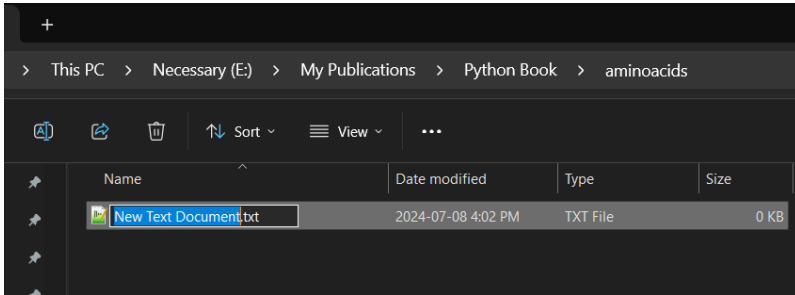
استاد: دو راه برای این کار وجود دارد:

راه اول این است که با موس تمام جدول را کپی کرده و آن را با کلیک راست و زدن دکمهٔ copy، کپی کنیم (شکل ۴):

Residue	3-letter	1-letter	Formula	Average mass	Monoisotopic mass
Glycine	Gly	G			
Alanine	Ala	A			
Serine	Ser	S			
Proline	Pro	P			
Valine	Val	V			
Threonine	Thr	T			
Cysteine	Cys	C			
Isoleucine	Ile	I			
Leucine	Leu	L			
Asparagine	Asn	N			
Aspartic acid	Asp	D			
Glutamine	Gln	Q			
Lysine	Lys	K			
Glutamic acid	Glu	E	$C_5H_7NO_3$	129.114	129.04259309
Methionine	Met	M	C_5H_9NOS	131.196	131.04048491
Histidine	His	H	$C_6H_7N_3O$	137.139	137.05891186
Phenylalanine	Phe	F	C_9H_9NO	147.174	147.06841391
Arginine	Arg	R	$C_6H_{12}N_4O$	156.186	156.10111102
Tyrosine	Tyr	Y	$C_9H_9NO_2$	163.173	163.06332853
Tryptophan	Trp	W	$C_{11}H_{10}N_2O$	186.210	186.07931295

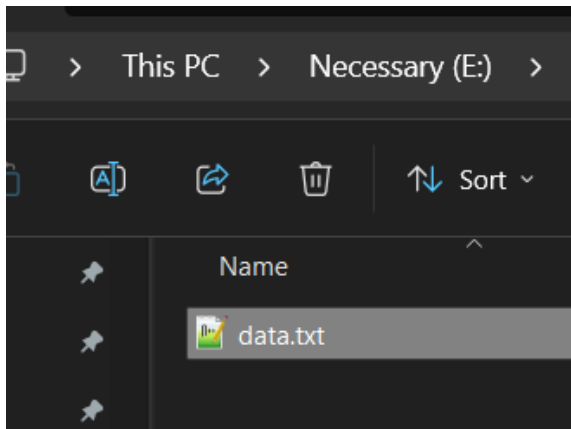
شکل ۴- روش اول: کپی کردن

سپس در پوشه‌ای در ویندوز، کلیک راست کرده و از منوی new text، گزینه new document را بزنیم تا یک فایل جدید متنی ایجاد شود:



شکل ۵- روش اول: ایجاد یک فایل متنی جدید

بهتر است نام این فایل را به نام با معنایی تغییر دهیم؛ مثلاً نام آن را data می‌گذارم (شکل ۶):



شکل ۶- تغییر نام فایل متنی

و درنهایت این فایل را باز کرده و چیزی که کپی کرده بودیم را در اینجا paste کنیم:

Residue	3-letter	1-letter	Formula	Average mass	Monoisotopic mass
Glycine	Gly G	C2H3NO	57.051	57.02146372	
Alanine	Ala A	C3H5NO	71.078	71.03711378	
Serine	Ser S	C3H5NO2	87.077	87.03202840	
Proline	Pro P	C5H7NO	97.115	97.05276395	
Valine	Val V	C5H9NO	99.131	99.06841391	
Threonine	Thr T	C4H7NO2	101.104	101.04767847	
Cysteine	Cys C	C3H5NOS	103.143	103.00918478	
Isoleucine	Ile I	C6H11NO	113.158	113.08406398	
Leucine	Leu L	C6H11NO	113.158	113.08406398	
Asparagine	Asn N	C4H6N2O2	114.103	114.04292744	
Aspartic acid	Asp D	C4H5NO3	115.087	115.02694302	
Glutamine	Gln Q	C5H8N2O2	128.129	128.05857751	
Lysine	Lys K	C6H12N2O	128.172	128.09496301	
Glutamic acid	Glu E	C5H7NO3	129.114	129.04259309	
Methionine	Met M	C5H9NOS	131.196	131.04048491	
Histidine	His H	C6H7N3O	137.139	137.05891186	
Phenylalanine	Phe F	C9H9NO	147.174	147.06841391	
Arginine	Arg R	C6H12N4O	156.186	156.10111102	
Tyrosine	Tyr Y	C9H9NO2	163.173	163.06332853	
Tryptophan	Trp W	C11H10N2O	186.210	186.07931295	

شکل ۷- کپی کردن محتوای سایت در این فایل متنی

و در نهایت با رفتن به منوی File و زدن گزینه Save، فایل را ذخیره کنید (اگر مثل من دوست دارید از کیبورد استفاده کنید، زدن دکمه ترکیبی Cntr1+S روی کیبورد همین کار را می‌کند). در نهایت فایل را ببندید.

الآن فایلی متنی به نام data.txt داریم که تمام اطلاعات مدنظر ما در آن ذخیره شده است. در مراحل بعد، این فایل را با پایتون می‌خوانیم.

دانشجو: استاد گفتید دو روش. روش دوم چیه؟

استاد: آهان خوب شد گفتی! خیلی از مواقع هست که جدول‌هایی که در یک سایت وجود دارد را اگر بخواهیم به این شکل با موس انتخاب و بعد کپی کنیم، به هم می‌ریزند؛ برای مثال سایت این جدول در سایت kegg در قسمت داروهای جدید تأیید شده جهان (new drug approvals) را ببینید (شکل ۸):

Approval dates by FDA, EMA and PMDA

FDA	Date	EMA	PMDA	KEGG	ATC	Active ingredient	Name		
							USA	Europe	Japan
2024/6/10			2024/6/24	D12705		Gumarontrib hydrate			Halyitan
2024/6/6				D11208 A05AX06		Elafiranol	Iqirvo		
				D09630 L01X080		Imetelstat sodium	Rytelo		
			2024/5/17	2024/6/24	D12358 A10AB07	Insulin toctec		Awigil	Awigil
2024/5/16				D12234 (L03FX33)		Tarlatamab	Imdeltra		
2024/4/26				D11510		Mavrisafor	Xolremdi		
2024/4/25				D12908 (B02BD17)		Fidacrogene elaprovoc	Beqvex		
2024/4/23				D12291		Tovorafenib	Ojemda		

شکل ۸- یک جدول دیگر

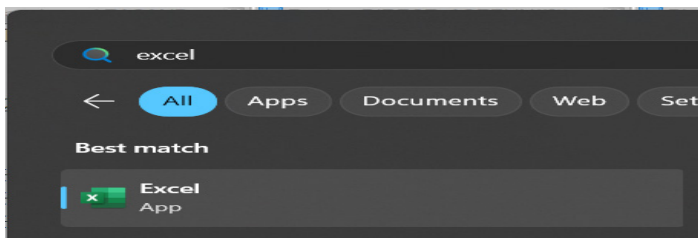
در این جدول در سطر اول، بالای سه ستون اول نوشته شده Date و بالای سه ستون آخر نوشته شده Name. این‌ها باعث می‌شوند که اگر فقط کپی و پیست کنیم، این جدول به هم بریزد.

دانشجو: استاد راهی نیست که مستقیم بتونیم جدول رو وارد کنیم؟

استاد: این همان راهی است که الان می‌خواهیم یاد بگیریم که علاوه بر این دوره، در بسیاری از جاهای دیگر هم به دردتان خواهد خورد. نرم‌افزار Excel یا همان اکسل خودمان، قابلیتی دارد که برای این موضوع بسیار خوب است. می‌تواند لینک اینترنتی را گرفته و جدول موجود در آن را استخراج کند.

دانشجو: عجب چیزیه استاد! حتماً باید کار باهاش خیلی سخت باشه!

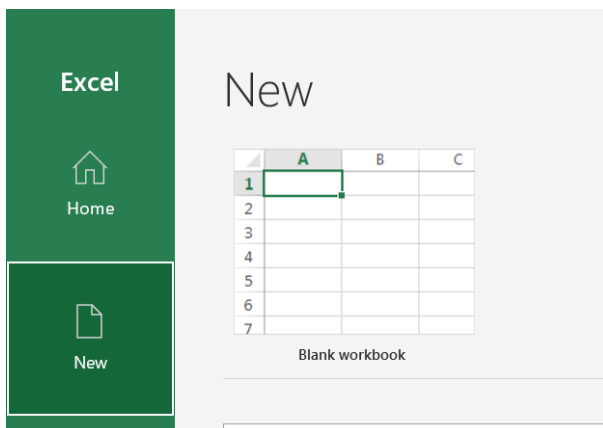
استاد: نه اصلاً! کافی است که نرم‌افزار اکسل را در کامپیوتر خود باز کنید (شکل ۹):



شکل ۹- نرم‌افزار اکسل (من از محیط استارت ویندوز آن را جستجو و باز کردم)

سپس زمانی که اکسل باز شد، روی گزینه New زده و با کلیک روی گزینه Blank

Workbook یک فایل اکسل جدید باز کنید (شکل ۱۰):



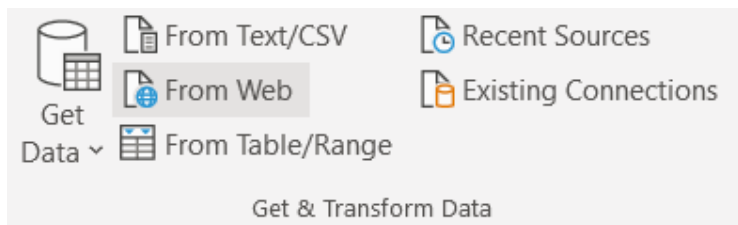
شکل ۱۰- باز کردن یک فایل جدید در اکسل

تا الآن یک فایل اکسل جدید باز کردیم. حال از بالای صفحه روی زبانه Data کلیک می‌کنیم (شکل ۱۱):



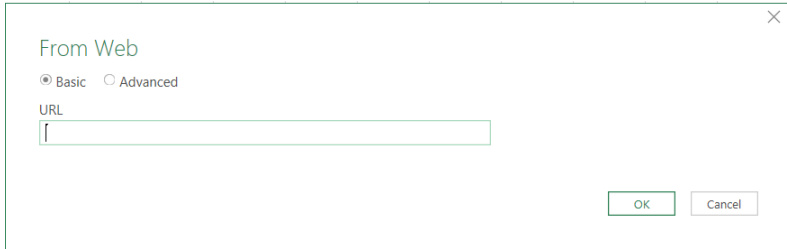
شکل ۱۱- روی زبانه Data کلیک کنید.

در سمت چپ صفحه گزینه‌های جالبی وجود دارد که ما فعلاً با گزینه Get data (شکل ۱۲):



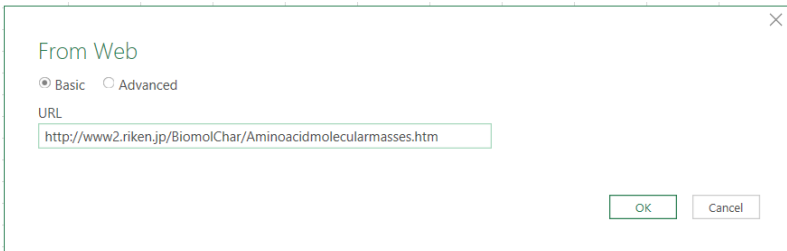
شکل ۱۲- روی From Web کلیک کنید.

یک پنجره باز می‌شود که از شما می‌خواهد آدرس صفحه اینترنتی خود که می‌خواهید جدول آن استخراج شود را وارد کنید (شکل ۱۳):



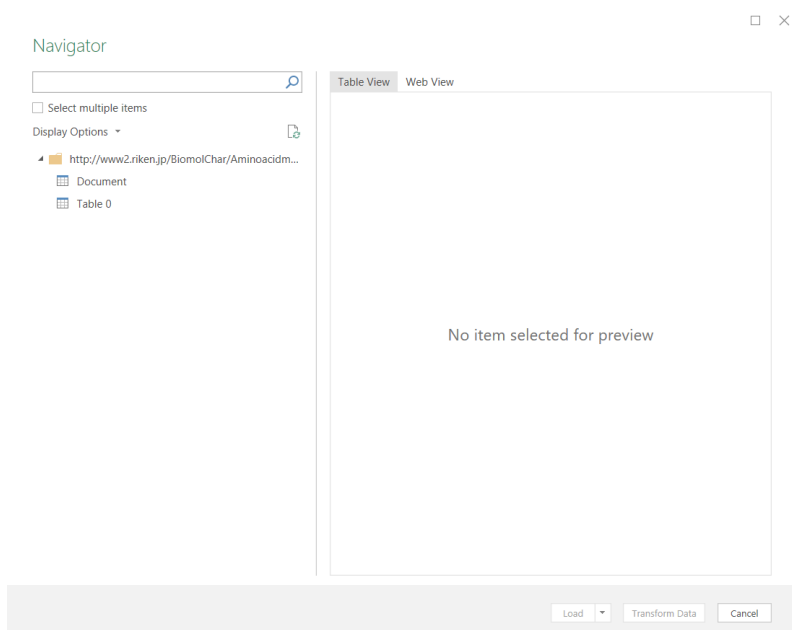
شکل ۱۳- من همان گزینه Basic را دست‌نخورده نگه داشتم.

و لینک صفحه اینترنتی مدنظر خود را در آن کپی کنید که من همان لینک صفحه‌ای که در آن جدول آمینواسیدها وجود داشت را کپی کردم (شکل ۱۴):



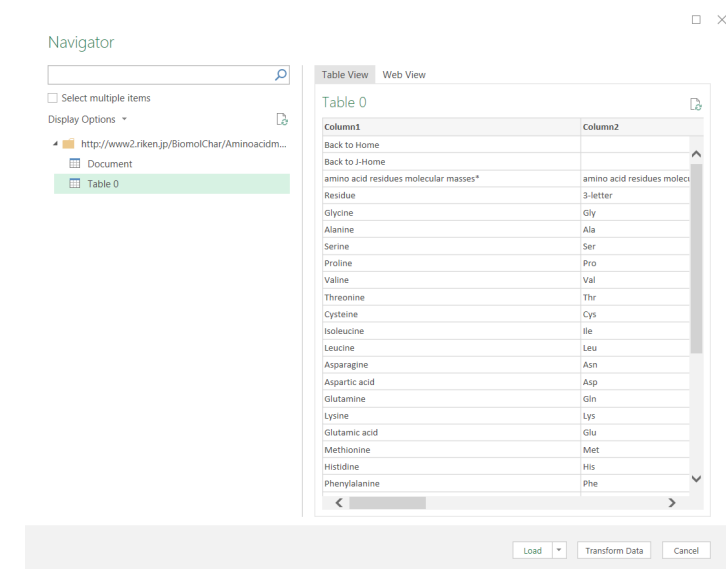
شکل ۱۴- لینک را کپی کنید و روی OK بزنید.

پس از زدن دکمه OK، مقداری زمان طول می‌کشد تا نرم‌افزار اکسل، محتوای لینک ما را خوانده و جداول را استخراج کند. در پنجره بعدی جداول استخراج شده نشان داده شده‌اند (شکل ۱۵):



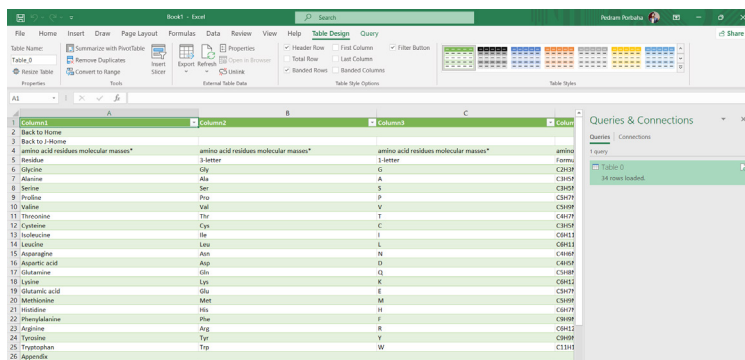
شکل ۱۵- جداول استخراج شده

روی هر کدام کلیک کنید، به شما نشان می‌دهد که هر کدام مربوط به کدام جدول است. برای مثال من روی Table 0 کلیک کردم و دیدم همین مورد، جدول مدنظر من است (درست است که دو سطر اول اضافه استخراج شده‌اند که بعد به راحتی می‌توان این دو سطر اضافه را در اکسل حذف کرد)(شکل ۱۶):



شکل ۱۶- جدول Table 0 را کلیک کرده و دیدیم که محتوای مدنظر ما همین است.

درنهایت کافی است روی گزینه Load پایین صفحه کلیک کنیم و در عرض چند ثانیه، می بینیم که به زیبایی کل جدول ما وارد اکسل شده است:



شکل ۱۷- جدول استخراج شده در نرم افزار اکسل

دانشجو: چه باحال! کیف کردم استاد! عالی بود!... فقط اینکه اون دو ردیف اول که اضافه هستن، توی ذوق می‌زنن! چطوری اونا رو پاک کنیم؟

استاد: خیلی راحت! کافی است موس را روی شماره ردیفی که مدنظر تان است نگه دارید تا شکل موس به صورت یک فلش در بیاید. حالا می‌توان کل ردیف را با یک کلیک انتخاب کرد. اول از همه من آن ردیفی که نام درست ستون‌ها است (ردیف پنجم) را انتخاب می‌کنم (شکل ۱۸):

	A	B	C	D	E	F	G	H
1	Column1	Column2	Column3	Column4	Column5	Column6		
2	Back to Home							
3	Back to J-Home							
4	amino acid residues molecu amino acid residues r amino acid residues mol amino acid residues mole amino acid residues molecular masses*							
5	Residue	3-letter	1-letter	Formula	Average mass	Monoisotopic mass		
6	Glycine	Gly	G	C2H3NO	57.051	57.02146372		
7	Alanine	Ala	A	C3H5NO	71.078	71.03711378		
8	Serine	Ser	S	C3H5NO2	87.077	87.03202840		
9	Proline	Pro	P	C5H7NO	97.115	97.05276385		
10	Valine	Val	V	C5H9NO	99.131	99.06841391		
11	Threonine	Thr	T	C4H7NO2	101.104	101.04767847		
12	Cysteine	Cys	C	C3H5NO2S	103.143	103.00918478		
13	Isoleucine	Ile	I	C6H11NO	113.158	113.08406398		
14	Leucine	Leu	L	C6H11NO	113.158	113.08406398		
15	Asparagine	Asn	N	C4H6N2O2	114.103	114.04292744		
16	Aspartic acid	Asp	D	C4H5NO3	115.087	115.02694302		

شکل ۱۸ - انتخاب ردیفی که نام درست ستون‌ها در آن نوشته شده

و سپس روی آن کلیک راست کرده و روی Cut کلیک می‌کنم:

4								
5	Division	3-letter	1-letter	Formula	Average mass	Monoisotopic mass		
6	Cut	Gly	G	C2H3NO	57.051	57.02146372		
7	Copy	Ala	A	C3H5NO	71.078	71.03711378		
8	Paste Options:	Ser	S	C3H5NO2	87.077	87.03202840		
9		Pro	P	C5H7NO	97.115	97.05276385		
10		Val	V	C5H9NO	99.131	99.06841391		

شکل ۱۹ - Cut کردن نام درست ستون‌ها

و حالا چون می‌خواهم نام ستون‌هایم، همین نام درست باشد نه چیزی که الان هست (Column 1, Column2 و ...)، روی شماره ردیف اول کلیک می‌کنم تا کل ردیف اول انتخاب شود (شکل ۲۰):

1	Column1	Column2	Column3	Column4	Column5	Column6
2	Back to Home					
3	Back to J-Home					
4	amino acid residues	mole	amino acid residues	m	amino acid residues	mole
5	Residue	3-letter	1-letter	Formula	Average mass	Monoisotopic mass
6	Glycine	Gly	G	C2H3NO	57.051	57.02146372
7	Alanine	Ala	A	C3H5NO	71.078	71.03711378
8	Serine	Ser	S	C3H5NO2	87.077	87.03202840
9	Proline	Pro	P	C5H7NO	97.115	97.05276385
10	Valine	Val	V	C5H9NO	99.131	99.06841391
11	Threonine	Thr	T	C4H7NO2	101.104	101.04767847
12	Cysteine	Cys	C	C3H5NO5	103.143	103.00918478

شکل ۲۰- انتخاب ردیف اول

و حالا کلیک راست کرده و گزینه paste را می‌زنم تا نام درست ستون‌ها جایگزین

شود (شکل ۲۱):

Paste Options:



شکل ۲۱- برای پیست کردن روی این دکمه بزنید.

و الان می‌بینید که نام ستون‌ها درست شده است (شکل ۲۲):

	A	B	C	D	E	F
1	Residue	3-letter	1-letter	Formula	Average mass	Monoisotopic mass
2	Back to Home					
3	Back to J-Home					
4	amino acid residues	mole	amino acid residues	m	amino acid residues	mole
5						
6	Glycine	Gly	G	C2H3NO	57.051	57.02146372
7	Alanine	Ala	A	C3H5NO	71.078	71.03711378
8	Serine	Ser	S	C3H5NO2	87.077	87.03202840
9	Proline	Pro	P	C5H7NO	97.115	97.05276385
10	Valine	Val	V	C5H9NO	99.131	99.06841391
11	Threonine	Thr	T	C4H7NO2	101.104	101.04767847
12	Cysteine	Cys	C	C3H5NO5	103.143	103.00918478
13	Isoleucine	Ile	I	C6H11NO	113.158	113.08406398
14	Leucine	Leu	L	C6H11NO	113.158	113.08406398

شکل ۲۲- نام درست ستون‌ها جایگزین شد.

تنها مشکل این است که ردیف شماره ۳ تا ۵ اضافه هستند و بهتر است حذف شوند. برای انتخاب کل این ردیف‌ها دقیقاً مثل مراحل قبل کافی است که موس را روی شماره ردیف آن‌ها نگه داریم تا شکل موس به شکل فلش تغییر پیدا کند و حالا کلیک چپ را نگه داشته و کل این ردیف‌های اضافه را انتخاب می‌کنیم (شکل ۲۳):

	A	B	C	D	E	F	G
1	Residue	3-letter	1-letter	Formula	Average mass	Monoisotopic mass	
2	Back to Home						
3	Back to J-Home						
4							
5							
6	Glycine	Gly	G	C2H3NO	57.051	57.02146372	
7	Alanine	Ala	A	C3H5NO	71.078	71.03711378	
8	Serine	Ser	S	C3H5NO2	87.077	87.03202840	
9	Proline	Pro	P	C5H7NO	97.115	97.05276385	

شکل ۲۳- انتخاب ردیف‌های اضافی

و در نهایت با کلیک راست و زدن گزینه Delete می‌توانیم به راحتی کل این ردیف‌ها را حذف کنیم (شکل ۲۴):

	A	B	C	D	E	F	G
1	Calibri	11	A	A	\$	%	
2	B	I					
3	Back to Home						
4	Cut						
5	Copy						
6	Paste Options:						
7	Paste Special...						
8	Insert						
9	Delete						
10	Clear Contents						
11	Format Cells...						
12							
13							
14							

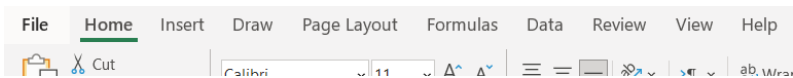
شکل ۲۴- گزینه Delete

همین کار را برای ردیف‌های ۲۲ به بعد هم انجام می‌دهیم و در نهایت می‌بینیم که کل ردیف‌های اضافه ما پاک شدند (شکل ۲۵):

	A	B	C	D	E	F	G
1	Residue	3-letter	1-letter	Formula	Average mass	Monoisotopic mass	
2	Glycine	Gly	G	C2H3NO	57.051	57.02146372	
3	Alanine	Ala	A	C3H5NO	71.078	71.03711378	
4	Serine	Ser	S	C3H5NO2	87.077	87.03202840	
5	Proline	Pro	P	C5H7NO	97.115	97.05276385	
6	Valine	Val	V	C5H9NO	99.131	99.06841391	
7	Threonine	Thr	T	C4H7NO2	101.104	101.04767847	
8	Cysteine	Cys	C	C3H5NO2	103.143	103.00918478	
9	Isoleucine	Ile	I	C6H11NO	113.158	113.08406398	
10	Leucine	Leu	L	C6H11NO	113.158	113.08406398	
11	Asparagine	Asn	N	C4H6N2O2	114.103	114.04292744	
12	Aspartic acid	Asp	D	C4H5N2O3	115.087	115.02694302	
13	Glutamine	Gln	Q	C5H8N2O2	128.129	128.05857751	
14	Lysine	Lys	K	C6H12N2O	128.172	128.09496301	
15	Glutamic acid	Glu	E	C5H7NO3	129.114	129.04259309	
16	Methionine	Met	M	C5H9NO2S	131.196	131.04048491	
17	Histidine	His	H	C6H7N3O	137.139	137.05891186	
18	Phenylalanine	Phe	F	C9H9NO	147.174	147.06841391	
19	Arginine	Arg	R	C6H12N4O	156.186	156.10111102	
20	Tyrosine	Tyr	Y	C9H9NO2	163.173	163.06332853	
21	Tryptophan	Trp	W	C11H10N2O	186.210	186.07931295	
22							
23							
24							

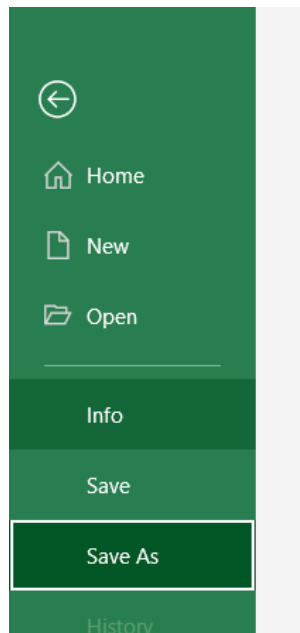
شکل ۲۵- فایل نهایی بدون ردیف اضافی

ما بعداً یاد خواهیم گرفت که همین فایل اکسل را به راحتی در یک خط کد با پایتون بخوانیم، اما در این جلسه چون کار با فایل های متنی را یاد می گیریم لازم است که این فایل را به یک فایل متنی تبدیل کنیم. برای این کار بالای صفحه سمت چپ روی منوی File کلیک کرده (شکل ۲۶):



شکل ۲۶- File

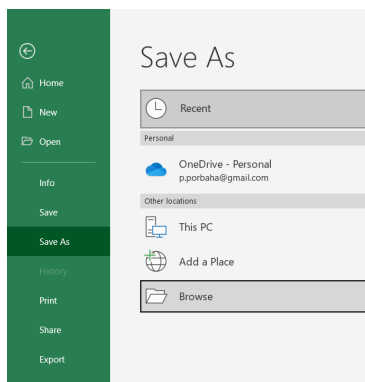
و سپس در پنجره باز شده روی گزینه Save As کلیک می کنیم (شکل ۲۷):



شکل ۲۷- روی گزینه Save As کلیک کنید.

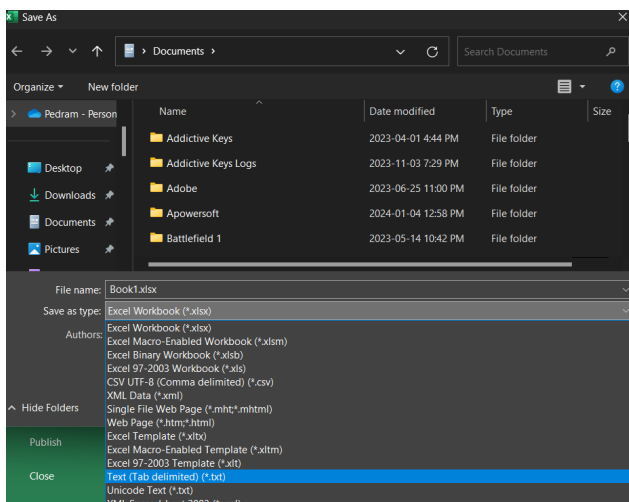
و در صفحه جدید روی گزینه Browse کلیک کنید تا بتوانیم مسیر و نام فایل متنی

که می‌خواهیم ذخیره کنیم را مشخص کنیم (شکل ۲۸):



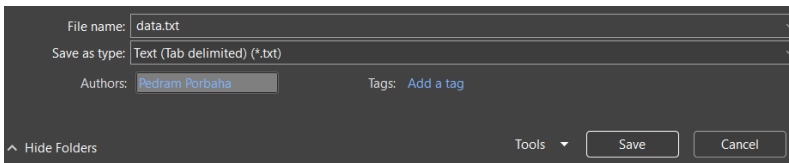
شکل ۲۸- پایین روی گزینه Browse کلیک کنید.

و در پنجره باز شده، حتماً روی قسمت Save as type کلیک کرده و از منوی باز شده گزینه Text (Tab Delimited) را انتخاب کنید (شکل ۲۹).



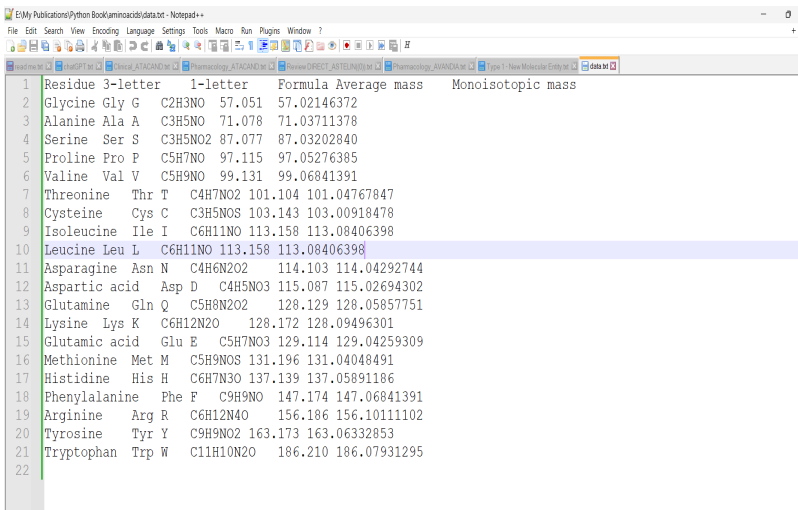
شکل ۲۹- حتماً گزینه Text (Tab delimited) را انتخاب کنید.

می‌توانید نام فایل را به دل‌خواه خود بگذارید؛ برای مثال من نام فایل را data.txt گذاشتم (آخر نام فایل، مربوط به پسوند فایل است که نشان می‌دهد این فایل، یک فایل متنی یا همان text است) و درنهایت روی Save کلیک کنید تا فایل متنی شما ذخیره شود و اگر پیامی هم ظاهر شد روی OK کلیک کنید (شکل ۳۰):



شکل ۳۰- درنهایت روی Save کلیک کنید.

اگر فایل data.txt ایجاد شده در محیط کامپیوتر خود را با برنامهٔ نوتپد باز کنید، می‌بینید که فایلی مشابه همان روش اول دارید (شکل ۳۱):



شکل ۳۱- فایل متنی ایجاد شده نهایی را با دبل کلیک باز کردیم.

و مانند همان روش اول بین عناصر جدول با کاراکتر tab (کاراکتر فاصله اما از فاصله عادی بزرگ‌تر است) جدا شده است.

روش سومی هم وجود دارد که اطلاعات موردنظر را به وسیله خود پایتون بیرون بکشیم و یا حتی سرچ کردن در اینترنت و کنترل موس و کیبورد را به صورت خودکار به پایتون بسپاریم. به این روش وب اسکرپینگ یا وب کراولینگ هم گفته می شود. برای این کار سه کتابخانه محبوب وجود دارد: BeautifulSoup، selenium و scrapy. درست است که کتابخانه دوم یعنی سلنیوم برای این کار خیلی بهینه نیست، اما به شخصه برای من خوش دست تر است و خیلی از مواقع کارم را راه می اندازد.

دانشجو: با این کتابخانه ها هم کار می کنیم؟

استاد: سعی می کنم اگر شد در جلد بعدی کتاب به آن بپردازم ولی اگر تمایل داشتید، اصول آن را به زبان ساده در این لینک توضیح داده ام (یا می توانید در گوگل جستجو کنید: آموزش وب اسکرپینگ با کتابخانه سلنیوم به زبان آدمیزاد! و یا بارکد موجود در شکل ۳۲ را اسکن کنید) که می توانید مرحله به مرحله از آن استفاده کنید. دوره وب کراولینگ در فرادرس نیز در این رابطه دوره خوبی است. در رابطه با scrapy هم اگر تمایل داشتید، دوره ای در فرادرس وجود دارد که می توانید از آن هم در ادامه استفاده کنید.

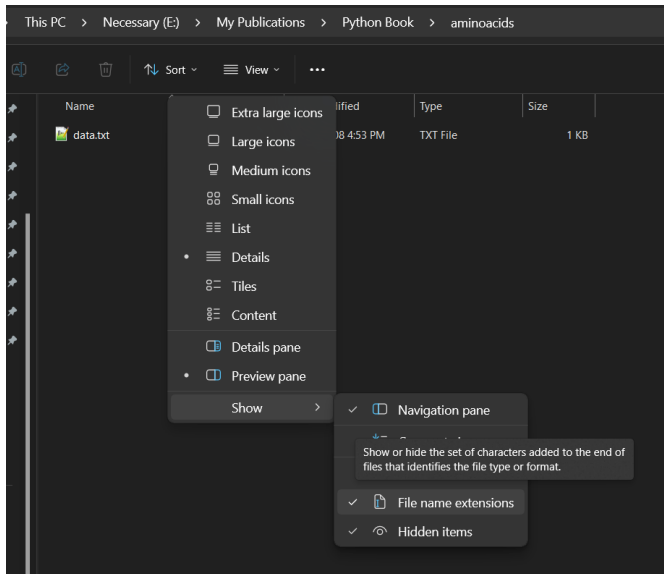


شکل ۳۲- لینک صفحه آموزش وب اسکرپینگ به زبان آدمیزاد!

خب برگردیم به موضوع خودمان! از هر کدام از روش‌ها استفاده کنیم، یک فایل data.txt داریم که نیاز است آن را در پایتون بخوانیم و به شکل یک دیکشنری در بیاوریم؛ اما قبل از آن خیلی سریع مروری بر انواع فایل داشته باشیم.

دانشجو: استاد راستی در سیستم من فقط نام فایل‌ها رو نشون می‌ده و مثل شما پسوند فایل‌ها دیده نمیشه.

استاد: در ویندوز من که ویندوز ۱۱ است، کافی است که در My Computer، در بالا منوی view را پیدا کنید و سپس آخرین گزینه یعنی show را زده و در نهایت file name extension را که به معنای نمایش extension یا همان پسوند فایل‌هاست را تیک بزنید (شکل ۳۳):



شکل ۳۳- روی گزینه view زده و سپس گزینه show و در نهایت روی file name extension بزنید تا فعال شود.

تا جایی که به خاطر دارم، در ویندوز ۱۰ یا ویندوز ۸ وقتی view را می‌زنید، همان‌جا یک تیک با همین عنوان file extension وجود دارد که می‌توانید فعالش کنید. اگر پیدا نکردید هم کافی است که یک سرچ کوچک در گوگل انجام دهید و ببینید که این گزینه در ویندوز شما کجا قرار دارد.

انواع فایل در کامپیوتر و نحوه باز کردن آن‌ها در پایتون

ما در کامپیوتر دو نوع فایل داریم: فایل‌های متنی و فایل‌های باینری. هر فایلی که توانستید آن را با کلیک راست و open with notepad به وسیله نوتپد باز کنید و با کاراکترهای عجیب و غریب روبرو نشدید، آن فایل یک فایل متنی است.

دانشجو: استاد من فکر می‌کردم هر فایلی که آخرش پسوند txt. داره، فایل متنیه. استاد: دقیقاً! من هم قبلاً این اشتباه را می‌کردم. پسوند، فقط چیزی است که به کامپیوتر راهنمایی می‌کند که احتمالاً این فایل باید با چه نرم‌افزاری باز شود، وگرنه هیچ اهمیتی ندارد. مثلاً شما می‌توانید آن فایلی که داشتیم به نام data.txt پسوندش را عوض کنید و آن را با نام خودتان بگذارید! مثلاً data.pedram: درست است که کامپیوتر به شما می‌گوید این فایل ناشناخته است، اما شما کماکان می‌توانید با کلیک راست روی آن و open with notepad آن را باز کنید و می‌بینید هیچ فرقی نکرده است و حتی می‌توان آن را تغییر داد.

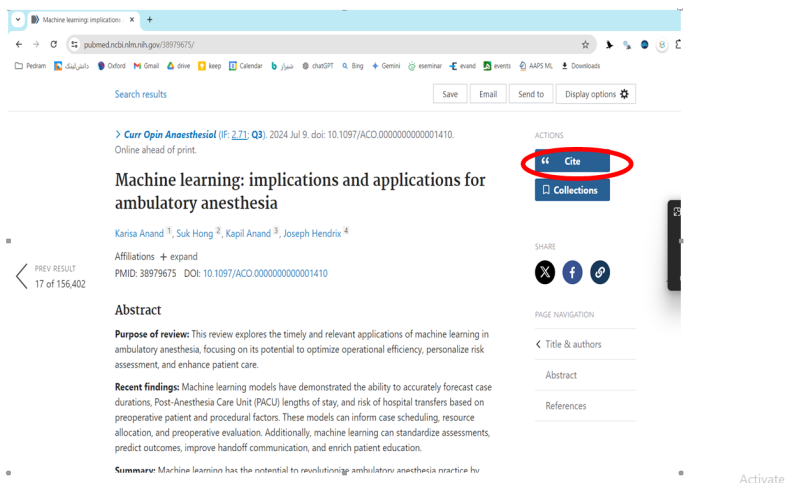
دانشجو: یه مثال از فایل باینری هم می‌زنید استاد؟

استاد: بله حتماً! فایل‌های باینری به صورت ۰ و ۱ در کامپیوتر ذخیره شده‌اند (البته درست‌تر این است که بگوییم در مبنای ۱۶ ذخیره شده‌اند) به همین دلیل اگر آن‌ها را با نوتپد باز کنید، با صحنه عجیبی روبرو می‌شوید. مثلاً یک فایل ورد یا پاورپوینت یا

امکانات بیشتری دارد.

حالا یک سؤال! شما هنگام مقاله‌نویسی، وقتی می‌خواهید citation یک مقاله را از سایت مجله یا پایگاهی مثل PubMed دانلود کنید، یک فایل با فرمت endnote یا nbib ... دانلود می‌شود که بعداً می‌توان آن را با نرم‌افزارهای رفرنس‌نویسی مثلاً endnote باز کرد. مثلاً من یادگیری ماشین را در پایگاه پابمد سرچ کردم و یک مقاله را باز کردم و citation آن را به‌صورت فایل nbib دانلود کردم. آیا این فایل، فایل باینری است یا متنی؟

(شکل ۳۵) بازدن روی گزینهٔ cite و دانلود آن، می‌توان فایل حاوی اطلاعات مثال را دانلود کرد.



دانشجو: قطعاً باینریه استاد! چون باید با نرم‌افزار اندنوت بازش کرد.

1 PMID- 38979675
2 OWN - NLM
3 STAT- Publisher
4 LR - 20240709
5 IS - 1473-6500 (Electronic)
6 IS - 0952-7907 (Linking)
7 DP - 2024 Jul 9
8 TI - Machine learning: implications and applications for ambulatory anesthesia.
9 LID - 10.1097/ACO.0000000000001410 [doi]
10 AB - PURPOSE OF REVIEW: This review explores the timely and relevant applications of
11 machine learning in ambulatory anesthesia, focusing on its potential to optimize
12 operational efficiency, personalize risk assessment, and enhance patient care.
13 RECENT FINDINGS: Machine learning models have demonstrated the ability to
14 accurately forecast case durations, Post-Anesthesia Care Unit (PACU) lengths of
15 stay, and risk of hospital transfers based on preoperative patient and procedural
16 factors. These models can inform case scheduling, resource allocation, and
17 preoperative evaluation. Additionally, machine learning can standardize
18 assessments, predict outcomes, improve handoff communication, and enrich patient
19 education. SUMMARY: Machine learning has the potential to revolutionize
20 ambulatory anesthesia practice by optimizing efficiency, personalizing care, and
21 improving quality and safety. However, limitations such as algorithmic opacity,
22 data biases, reproducibility issues, and adoption barriers must be addressed
23 through transparent, participatory design principles and ongoing validation to
24 ensure responsible innovation and incremental adoption.
25 CI - Copyright © 2024 The Author(s). Published by Wolters Kluwer Health, Inc.
26 FAU - Anand, Karisa
27 AU - Anand K
28 AD - University of Texas Southwestern.

آن را با notepad باز می‌کنیم. جالب است بدانید که بعضی از مواقع، save بازی‌ها درواقع یک فایل متنی است و حتی می‌توان را به شکل دستی تغییر داد!

دانشجو: چه جالب! استاد فایل‌های باینری رو هم میشه دستی تغییر داد؟

استاد: نه به راحتی فایل‌های متنی، ولی بله! نرم‌افزارهایی وجود دارند مثلاً مانند HexEditor که می‌توانند این فایل‌ها را بخوانند و تغییر دهند.

حالا ما با پایتون چه می‌کنیم؟ ما در این فصل با فایل‌های متنی کار می‌کنیم، اما آیا این بدان معنی است که با پایتون نمی‌توانیم فایل‌های باینری را باز کنیم؟ البته که می‌توانیم! دقیقاً با همان روشی که فایل‌های متنی را باز می‌کنیم (البته با تغییر کوچکی در دستور که با یک سرچ ساده می‌توان آن را پیدا کرد) می‌توانیم انواع فایل‌ها مثل فایل ورد، پاورپوینت، عکس، فیلم، موسیقی و... را با پایتون باز کنیم، تغییر دهیم و با پایتون آن فایل را تولید کنیم. برای پایتون حتی کتابخانه‌هایی وجود دارد که بسیار راحت و در یک خط می‌توان مثلاً فایل موسیقی را خواند. ما فایل متنی که اساس است را باهم کار می‌کنیم اما همان نیز می‌تواند مبنایی برای فایل‌هایی باشد که بعداً می‌خواهید با آن‌ها کار کنید.

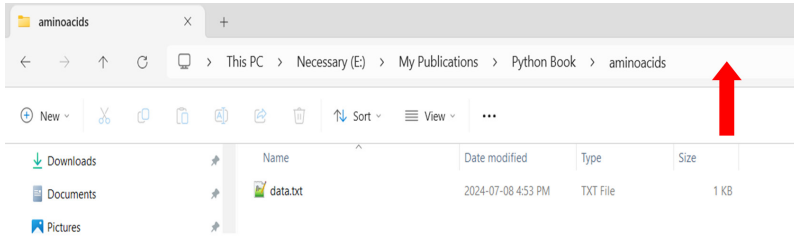
دانشجو: چه خفننن!!

استاد: من از وقتی کار با فایل‌ها را با پایتون شروع کردم، لذتم از برنامه‌نویسی صد برابر شد چون دیدم که فقط محدود به پایتون نیستم!

فهماندن آدرس فایل به پایتون!

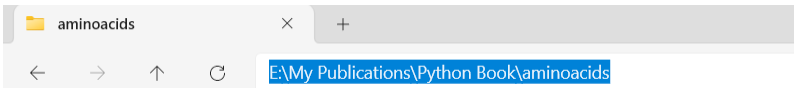
برای خواندن فایل موردنظرمان در پایتون، به آدرس آن فایل (یا در اصطلاح علوم کامپیوتر، path آن فایل) نیاز داریم. چطور می‌توانیم بفهمیم Path یا آدرس فایل ما

در کامپیوتر چیست؟ بیاید نگاهی به پوشه‌ای که فایل موردنظر ما در آن قرار دارد بیندازیم (شکل ۳۷):



شکل ۳۷- پوشه‌ای که فایل مدنظر ما، یعنی data.txt در آن قرار دارد.

اگر در بالای صفحه، در یک جای خالی از نوار آدرس موجود کلیک کنیم (جایی که با فلش قرمز مشخص کرده‌ام)، نوار آدرس به شکل زیر درمی‌آید (شکل ۳۸):



شکل ۳۸- پس از کلیک در یک جای خالی در نوار آدرس، نوار آدرس به این شکل قابل کپی کردن درمی‌آید. این آدرس یا Path پوشه ما است؛ یعنی چه؟ از سمت چپ شروع به خواندن کنید: اگر در درایو E، در پوشه My Publication وارد شویم و در آن پوشه Python Book شویم و در نهایت وارد پوشه aminoacids شویم، به اینجا می‌رسیم (می‌بینید که در ویندوز، نام پوشه‌ها یا فایل‌ها با علامت \ یا همان backslash از هم جدا می‌شوند). پس اگر بخواهیم آدرس فایل موردنظرمان را در پایتون به صورت یک رشته در یک متغیر ذخیره کنیم، کافی است این آدرس را کپی کنیم:

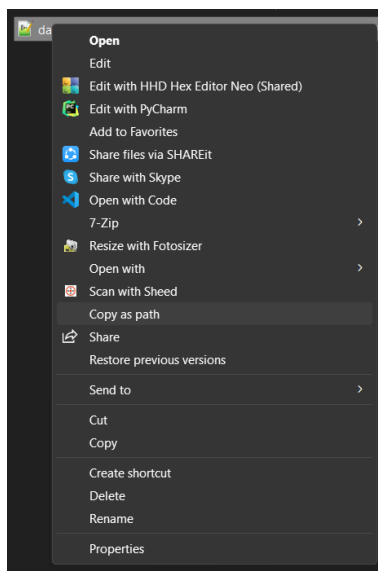
```
path = 'E:\My Publications\Python Book\aminoacids'
```

من نام متغیر را همان path گذاشتم که نام بامسمایی باشد؛ اما توجه داریم که این آدرس پوشهٔ ماست نه آدرس فایل data.txt. به عبارتی لازم است که در انتهای این آدرس، نام فایل خود را هم با یک علامت \ اضافه کنیم، یعنی:

```
path = 'E:\My Publications\Python Book\aminoacids\data.txt'
```

و حواستان هم باشد که پسوند فایل را فراموش نکنید.

یک راه بسیار راحت‌تر هم در ویندوز وجود دارد! کافی است کلید Shift را روی کیبورد پایین نگه‌داشته و روی فایل مدنظر خود کلیک راست کنید. می‌بینید که به‌صورت جالبی گزینهٔ Copy as path اضافه شده است و با زدن آن به‌راحتی آدرس فایل موردنظر (نه پوشهٔ ما) کپی می‌شود (شکل ۳۹):



شکل ۳۹- راه راحت‌تر برای کپی کردن path یک فایل (فراموش نکنید که حتماً باید کلید shift را پایین نگه دارید و در همان حین روی فایل موردنظر کلیک راست کنید).

و می‌توانیم به راحتی این آدرس را در پایتون کپی کنیم:

```
path = "E:\My Publications\Python Book\aminoacids\data.txt"
```

دانشجو: از راه قبلی خیلی راحت‌تره! پس درواقع با این کار آدرس فایلمون رو در یک متغیر ذخیره می‌کنیم.

استاد: دقیقاً! منتها با یک مشکل روبرو می‌شویم! و در خیلی از مواقع اگر در این صورت برنامه را اجرا کنیم خطا می‌گیریم!

دانشجو: اما استاد ما که هنوز برنامه‌ای ننوشتیم که بخوایم خطا بگیریم!

استاد: در همین خط! چه خطایی؟ اگر یادتان باشد، گفتیم علامت \ در رشته‌ها می‌تواند به صورت یک اپراتور کنترلی هم معنا بدهد. مثل چی؟ مثلاً \n به معنی خط جدید بود که از آن زیاد استفاده کردیم یا مثلاً \t که معنای فاصله tab است. مثلاً اگر می‌نوشتیم:

```
print('I love python\n I love Machine learning!')
```

پایتون \n را به مثابه اپراتور کنترلی خط جدید در نظر می‌گرفت و برای ما چاپ می‌کرد:

```
I love python
I love Machine learning!
```

حالا پایتون بیچاره چه کار کند که در ویندوز، آدرس‌ها و فایل‌ها با علامت \ از هم جدا شده است! او که نمی‌داند! فکر می‌کند تمامی این \ها معنی دارند!

دانشجو: پس باید یه جوری به پایتون بفهمونیم که منظورمون از \ خود این علامته نه یه اپراتور کنترلی!

استاد: دقیقاً! برای این کار دو راه داریم:

راه اول اینکه به جای `\` از `\\` استفاده کنیم. با این کار به پایتون می‌فهمانیم که منظور ما همان خود `\` است نه `\\` مثلاً اگر بنویسیم:

```
print('I love python\\nI love Machine learning!')
```

دیگر برخلاف قبل، پایتون به `\` به‌مثابه یک اپراتور کنترلی نگاه نمی‌کند و چاپ می‌کند:

```
I love python\nI love Machine learning!
```

پس یک راه این است که در آدرس فایل خود، تمامی `\\`ها را به `\\` تغییر دهیم، یعنی:

```
path = "E:\\My Publications\\Python Book\\aminoacids\\data.txt"
```

به این راه، فرار از علامت بک‌اسلش (backslash escaping) گفته می‌شود، اما همان‌طور که خودتان هم دارید می‌بینید، این کار بسیار وقت‌گیر است.

راه دوم، این است که به‌راحتی، قبل از رشته خود یک حرف `r` بنویسیم، یعنی:

```
path = r"E:\My Publications\Python Book\aminoacids\data.txt"
```

`r` مخفف raw است؛ یعنی به پایتون می‌گوییم پایتون عزیز! این رشته را همان‌طور که هست بخوان و به شکل اپراتور کنترلی با آن رفتار نکن! (قبلاً که نمی‌دانستیم، فکر می‌کردم `r` مخفف repeat است و تصور می‌کردم که می‌آید تمام `\\`ها را دوبل می‌کند ولی اشتباه می‌کردم ☹).

یک راه سوم هم هست! می‌توانید آدرس فایل خود را به جای `\`، با علامت `/` جدا کنید که با این کار دیگر نیازی به دو روش قبل نیست:

```
path = "E:/My Publications/Python Book/aminoacids/data.txt"
```

خود من به دلیلی که `path` فایل یا فولدر موردنظر را با کلیک راست کردن کپی

می‌کنم، اکثراً از همان روش دوم استفاده می‌کنم که راحت‌تر است، یعنی یک `r` قبل از رشته خود اضافه می‌کنم:

```
path = r"E:\My Publications\Python Book\aminoacids\
data.txt"
```

خواندن فایل موردنظر در پایتون

تا اینجا، آدرس فایل موردنظر خود را در یک متغیر به اسم `path` ذخیره کردیم. حالا ببینیم چگونه می‌توانیم این فایل متنی را بخوانیم. باز کردن به انگلیسی چه می‌شود؟

دانشجو: `open`

استاد: نام دستوری که برای باز کردن فایل در پایتون هم استفاده می‌شود، همین `open` است! این یک دستور است برای همین مثل بقیه دستورها و توابع در پایتون نیاز به پرانتز داریم. این دستور، آرگومان‌های اختیاری زیادی دارد که اجباری نیستند. تنها چیزی که لازم است آدرس فایل است و یک آرگومان دیگر به نام `mode` که چون ما می‌خواهیم فایل را بخوانیم و نمی‌خواهیم فایل را بنویسیم، در اینجا `mode` ما برابر با `'r'` یا همان مخفف `read` است:

```
open(path, 'r')
```

و حاصل را باید در یک متغیر ذخیره کنیم و چه نام متغیری بهتر از `file` که با دیدن آن متوجه منظور آن شویم:

```
file = open(path, 'r')
```

اگر `mode` را مشخص نمی‌کردیم، پایتون به صورت پیش فرض آن را `'r'` یا همان `read` در نظر می‌گرفت، در نتیجه در این مثال که ما می‌خواهیم فایل را بخوانیم، اگر `mode` را نمی‌نوشتیم هم مشکلی پیش نمی‌آمد:

```
file = open(path)
```

حالا چه کار کنیم؟ file متدی دارد به نام readlines که چه می‌کند؟ تمامی خطوط موجود در فایل ما را به صورت یک لیست می‌دهد؛ به همین دلیل خروجی آن را در متغیری به نام lines ذخیره کردم تا نام خوبی باشد:

```
lines = file.readlines()
```

الآن مقدار lines را اگر ببینیم، چه چیزی مشاهده می‌کنیم؟ درواقع مثلاً اگر فایل ما حاوی همچنين متنی بود:

```
I love Python
```

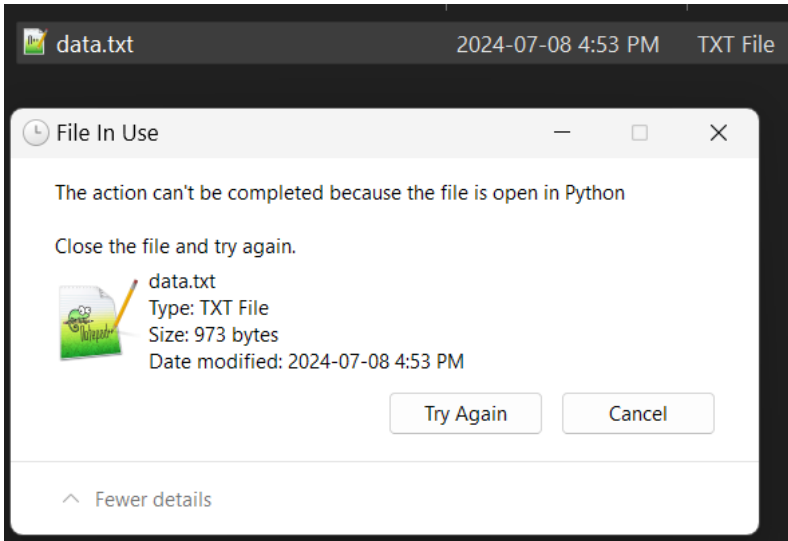
```
I love Machine learning
```

اگر این فایل را با پایتون می‌خواندیم و متد readlines را صدا می‌زدیم، lines ما همچنين لیستی می‌شد:

```
['I love Python\n', 'I love Machine learning']
```

دانشجو: آهان! یعنی الآن لیستی داریم که هر عضو اون یه خط از فایل ماست.

استاد: دقیقاً! و حالا می‌توانیم کارهایی که می‌خواهیم را انجام دهیم. اما یک نکته که نباید فراموش کنیم! ما فایل را در پایتون باز کردیم، درست است؟ و آن را هنوز نبستیم! به عبارتی هنوز این فایل باز است! مثلاً شما اگر بخواهید خودتان دستی از کامپیوتر خود فایل data.txt را پاک کنید، به شما خطایی نمایش داده می‌شود که این فایل در یک برنامه دیگر در حال استفاده است و شما نمی‌توانید آن را پاک کنید (شکل ۴۰):



شکل ۴۰- چون فایل در پایتون باز شده اما بسته نشده، حتی خارج از پایتون نیز دیگر نمی‌توانیم در فایل موردنظر تغییر ایجاد کنیم یا آن را پاک کنیم زیرا این فایل در پایتون باز بوده و نمی‌تواند هم‌زمان در دو برنامه مورد تغییر قرار بگیرد.

دانشجو: ای داد! پس باید چیکار کنیم؟

استاد: خیلی راحت! ما که تمام خطوط فایل‌مان را در lines ذخیره کردیم؛ کافی است که در پایتون فایل باز شده را ببندیم. در انگلیسی بستن فایل چه می‌شود؟

دانشجو: close

استاد: دقیقاً کافی است با متد close فایل خود را ببندیم:

`file.close()`

و با این کار دیگر فایل در پایتون بسته شده و در بیرون از پایتون نیز اگر بخواهیم تغییری در فایل بدهیم، با مشکل روبرو نمی‌شویم. پس به عبارتی با سه خط توانستیم

فایل را باز کنیم و بخوانیم و ببندیم، یعنی:

```
path = r"E:/My Publications/Python Book/aminoacids/
data.txt"
file = open(path)
lines = file.readlines()
file.close()
```

اما نکته‌ای وجود دارد! بسیاری از مواقع ما فراموش می‌کنیم که فایل را پس از اتمام کارمان ببندیم و همین موضوع باعث ایجاد خطاهایی می‌شود که انتظارش را نداریم. پایتون برای این کار یک syntax خاص دارد که خودش پس از اتمام کار فایل را می‌بندد و دیگر لازم نیست نگران باشیم که نکند بستن فایل را فراموش کنیم. برای این کار کافی است بنویسیم:

```
with open(path) as file:
    lines = file.readlines()
```

یعنی کافی است بنویسیم `with open` و دقیقاً عین قبل آدرس فایل را بنویسیم و پس از `as` نام متغیری که می‌خواهیم فایل ما در آن قرار بگیرد را ذکر می‌کنیم. درنهایت با یک دندان، تمام دستوراتی که می‌خواهیم را می‌نویسیم. پایتون، وقتی دستورات زیرمجموعه `with` تمام شد، خودش به شکل خودکار فایل را می‌بندد.

معمولاً در پایتون برای باز کردن فایل یا ... از همین گونه نوشتن استفاده می‌شود و به همین دلیل ما هم دیگر از همین گونه نوشتن استفاده می‌کنیم. اگر هم خواستیم که `mode` را بنویسیم، عین دستورات قبلی پس از آدرس فایل آن را می‌نویسیم مثلاً `'r'` برای خواندن فایل (که البته دیدیم که اگر `mode` را ننویسیم، پایتون خودش به شکل پیش فرض، `mode` را همان `'r'` در نظر می‌گیرد):

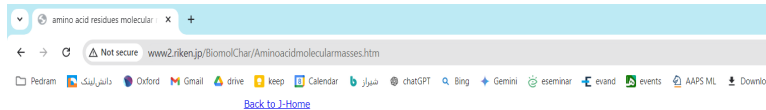
```
path = r"E:/My Publications/Python Book/aminoacids/
```

data.txt"

```
with open(path, 'r') as file:
    lines = file.readlines()
```

استفاده از اطلاعات فایل خود در پایتون

خب تا الان دیدیم که چگونه می توانیم به اطلاعات فایلمان دسترسی پیدا کنیم؛ اما هدف اصلی ما این بود که یک دیکشنری از آمینواسیدها و جرم مولکولی آنها داشته باشیم که بتوانیم بعداً از آن استفاده کنیم. برای این که دید بهتری داشته باشیم، مجدد جدولی که از سایت موردنظر کپی کردیم را ببینیم (شکل ۴۱):



amino acid residues molecular masses*

Residue	3-letter	1-letter	Formula	Average mass	Monoisotopic mass
Glycine	Gly	G	C ₂ H ₃ NO	57.051	57.02146372
Alanine	Ala	A	C ₃ H ₅ NO	71.078	71.03711378
Serine	Ser	S	C ₃ H ₅ NO ₂	87.077	87.03202840
Proline	Pro	P	C ₅ H ₇ NO	97.115	97.05276385
Valine	Val	V	C ₅ H ₉ NO	99.131	99.06841391
Threonine	Thr	T	C ₄ H ₇ NO ₂	101.104	101.04767847
Cysteine	Cys	C	C ₃ H ₅ NOS	103.143	103.00918478
Isoleucine	Ile	I	C ₆ H ₁₁ NO	113.158	113.08406398
Leucine	Leu	L	C ₆ H ₁₁ NO	113.158	113.08406398
Asparagine	Asn	N	C ₄ H ₆ N ₂ O ₂	114.103	114.04292744
Aspartic acid	Asp	D	C ₄ H ₅ NO ₃	115.087	115.02694302
Glutamine	Gln	Q	C ₅ H ₈ N ₂ O ₂	128.129	128.05857751
Lysine	Lys	K	C ₆ H ₁₂ N ₂ O	128.172	128.09496301
Glutamic acid	Glu	E	C ₅ H ₇ NO ₃	129.114	129.04259309
Methionine	Met	M	C ₅ H ₉ NOS	131.196	131.04048491
Histidine	His	H	C ₆ H ₇ N ₃ O	137.139	137.05891186
Phenylalanine	Phe	F	C ₉ H ₉ NO	147.174	147.06841391
Arginine	Arg	R	C ₆ H ₁₂ N ₄ O	156.186	156.10111102
Tyrosine	Tyr	Y	C ₉ H ₉ NO ₂	163.173	163.06332853
Tryptophan	Trp	W	C ₁₁ H ₁₀ N ₂ O	186.210	186.07931295

شکل ۴۱- جدول موجود در سایت موردنظر

که ما به آن ستون سوم که نام تک‌حرفی آمینواسیدها (ستون 1-letter) را نوشته و همچنین ستون یکی مانده به آخر که جرم مولکولی میانگین آن را نوشته (یعنی ستون Average mass) نیاز داریم.

فایلی هم که این جدول را در آن کپی کردیم و با پایتون خواندیم (data.txt) به این شکل بود (شکل ۴۲):

1	Residue	3-letter	1-letter	Formula	Average mass	Monoisotopic mass
2	Glycine	Gly G	C2H3NO	57.051	57.02146372	
3	Alanine	Ala A	C3H5NO	71.078	71.03711378	
4	Serine	Ser S	C3H5NO2	87.077	87.03202840	
5	Proline	Pro P	C5H7NO	97.115	97.05276385	
6	Valine	Val V	C5H9NO	99.131	99.06841391	
7	Threonine	Thr T	C4H7NO2	101.104	101.04767847	
8	Cysteine	Cys C	C3H5NOS	103.143	103.00918478	
9	Isoleucine	Ile I	C6H11NO	113.158	113.08406398	
10	Leucine	Leu L	C6H11NO	113.158	113.08406398	
11	Asparagine	Asn N	C4H6N2O2	114.103	114.04292744	
12	Aspartic acid	Asp D	C4H5NO3	115.087	115.02694302	
13	Glutamine	Gln Q	C5H8N2O2	128.129	128.05857751	
14	Lysine	Lys K	C6H12N2O	128.172	128.09496301	
15	Glutamic acid	Glu E	C5H7NO3	129.114	129.04259309	
16	Methionine	Met M	C5H9NOS	131.196	131.04048491	
17	Histidine	His H	C6H7N3O	137.139	137.05891186	
18	Phenylalanine	Phe F	C9H9NO	147.174	147.06841391	
19	Arginine	Arg R	C6H12N4O	156.186	156.10111102	
20	Tyrosine	Tyr Y	C9H9NO2	163.173	163.06332853	
21	Tryptophan	Trp W	C11H10N2O	186.210	186.07931295	
22						

شکل ۴۲- فایل متنی data.txt که با برنامه notepad++ باز کردیم.

خب ما دیدیم که می‌توانیم کل خطوط برنامه را بخوانیم و برنامه را تا اینجا نوشتیم:

```
path = r"E:/My Publications/Python Book/aminoacids/
data.txt"
```

```
with open(path, 'r') as file:
    lines = file.readlines()
```

اگر برنامه را اجرا کنیم، فایلمان خوانده شده و خطوط آن در لیست lines ذخیره شده‌اند. پس می‌توانیم مشابه همیشه در قسمت کنسول پایتون (پایین سمت راست اسپایدر) به وسیلهٔ دستورهای تکی، ببینیم که مثلاً خط اول حاوی چیست؟ چگونه بفهمیم خط اول ما چیست؟ یا به عبارت دیگر چگونه عضو اول لیست lines را ببینیم؟ دانشجو: ... عضو اول می‌شد [0]؛ پس می‌تونیم در کنسول بنویسیم:

```
lines[0]
```

چه خروجی عجیبی گرفتم!

```
'Residue\t3-letter\t1-letter\tFormula\tAverage mass\tMonoisotopic mass\n'
```

این خط اول فایل ماست؟!

استاد: \t در یک رشته به چه معناست؟

دانشجو: ... تا جایی که یادمه شبیه \n بود اما \n یعنی خط جدید و \t یعنی ... آهان یعنی کاراکتر فاصلهٔ tab!

استاد: دقیقاً! و فایل ما هم با tab اعضای آن از هم جدا شده‌اند! اگر می‌خواهی بهتر متوجه منظورم شوی، می‌توانی به جای نوشتن مستقیم:

```
lines[0]
```

از دستور پرینت استفاده کنی تا به جای \t را همان فاصله چاپ کند:

```
print(lines[0])
```

و می‌بینیم همان خط اولی است که در فایلمان داشتیم:

FormulaAverage	1-letter	3-letter	Residue
Monoisotopic mass			mass

دانشجو: آهان من فکر می‌کردم در فایل ما اطلاعاتمون با علامت ”جدا شدن، نگو که اشتباه می‌کردم! با کاراکتر tab از هم جدا شدن.

استاد: دقیقاً! و حالا یک سؤال دیگر! بیا مجدد خود خط اول را (بدون پرینت) ببینیم:

```
lines[0]
```

```
\t\tها را فهمیدیم! \n آخر آن به چه معناست؟
'Residue\t3-letter\t1-letter\tFormula\tAverage mass\
tMonoisotopic mass\n'
```

دانشجو: چون آخر خط بوده و در فایل ما بعدش خط جدید بوده!

استاد: حالا چگونه می‌توانیم به اطلاعات خط دوم دسترسی پیدا کنیم؟

دانشجو: خیلی راحت!

```
lines[1]
```

خروجی:

```
'Glycine\tGly\tG\tC2H3N0\t57.051\t57.02146372\n'
```

استاد: که همان ردیف مربوط به آمینواسید Glycine است. پس درواقع اولین خط ما، شامل نام ستون‌های ماست که به آن header می‌گویند.

خب حالا چطور می‌توانیم روی این لیست خطوط، حرکت کنیم و آمینواسید و جرم مولکولی آن را استخراج کنیم؟

دانشجو: تا می‌گید حرکت روی یه چیز، یاد حلقهٔ for می‌افتم!

استاد: دقیقاً! و می‌خواهیم اطلاعاتمان نیز در یک دیکشنری ذخیره شود. یک دیکشنری که همان‌طور که در جلسهٔ قبل دیدیم، کلیدهای آن همان نام تک‌حرفی

آمینواسیدها و valueهای آن، جرم مولکولی باشد. پس قبل از هر کاری، یک دیکشنری خالی به نام مثلاً aminoacids تعریف می‌کنیم:

```
aminoacids = {}
```

دانشجو: و چون می‌خواهیم در لیستمون حرکت کنیم، حلقه For رو هم من می‌نویسم! دقیقاً از بعد جایی که لیست خطوط رو از فایلمون خونديم:

```
for i in lines:
```

استاد: i هربار در حلقه چه چیزی می‌شود؟

دانشجو: چون مستقیماً داریم روی lines حرکت می‌کنیم، طبیعتاً یکی از خطوط برنامه می‌شه.

استاد: پس بهتر نیست یک نام بهتر، مثلاً line بگذاریم؟

دانشجو: استااااااااااا! چشم به خاطر شما!

```
for line in lines:
    print(line)
```

پس انگار برنامه ما تا اینجا شد:

```
path = r"E:/My Publications/Python Book/aminoacids/
data.txt"
```

```
aminoacids = {}
with open(path, 'r') as file:
    lines = file.readlines()
```

```
for line in lines:
    print(line)
```

استاد: بسیار عالی! همان‌طور که در کنسول هم چک کردیم و می‌توانیم مجدد هم چک کنیم. مثلاً یکی از خطوط ما، مثلاً خط سوم را ببینیم:

```
lines[2]
```

```
>>>
```

```
'Alanine\tAla\tA\tC3H5N0\t71.078\t71.03711378\n'
```

نوع متغیر چیست؟

دانشجو: چون کلاً در دو تا علامت ^۹ قرار گرفته، قطعاً رشته‌اس.

استاد: خب بیا برای مثال، فعلاً برای یکی از خطوط، استخراج اطلاعات را در همین محیط کنسول اسپایدر انجام دهیم و اگر دیدیم درست بود این دستورات را زیر همان For کپی کنیم تا تکرار شوند. بگذار من یک متغیر تعریف کنم که خط سوم فایل‌مان را در آن بریزم:

```
line = lines[2]
```

و یک بار دیگر line را ببینیم:

```
'Alanine\tAla\tA\tC3H5N0\t71.078\t71.03711378\n'
```

اطلاعات ما با چه چیزی از هم تفکیک شده‌اند؟

دانشجو: با کاراکتر ^۹'t' یا همون tab خودمون!

استاد: چطور می‌توانیم این رشته را به اجزای آن تفکیک کنیم؟

دانشجو: ... آهان! با متد split!

استاد: دقیقاً! پس می‌توانیم برای مثال بنویسیم:

```
line.split('\t')
```


و می‌بینیم که چون در محیط کنسول هستیم، خروجی آن را می‌بینیم:

```
[ 'Alanine', 'Ala', 'A', 'C3H5NO', '71.078',  
'71.03711378\n' ]
```

درست استخراج شده؟

دانشجو: آره استاد! چه باحال!

استاد: دقیقاً! منتها مشکلی داریم! در عضو آخر یک کاراکتر `\n` مزاحم وجود دارد. درست است که در این مثال با عضو آخر کاری نداریم و تنها به عضو سوم (حرف تک حرفی آمینواسید) و عضو پنجم آن (جرم مولکولی) نیاز داریم، اما بهتر است برای این که از باگ جلوگیری کنیم، همین الان از شر کاراکترهای `\n` موجود در متغیر خودمان خلاص شویم!

دانشجو: اما چطوری؟

استاد: یک راه ساده و مؤثر، متد `replace` در رشته‌ها است. کافی است بگوییم که در `line` هر چه `'\n'` وجود دارد را با هیچی یعنی `''` جایگزین کند:

```
line = line.replace('\n', '')
```

اگر یک‌بار `line` را ببینیم، دیگر خبری از `\n` در آن نیست:

```
line
```

```
<<<
```

```
'Alanine\tAla\tA\tC3H5NO\t71.078\t71.03711378'
```

و اگر همان‌طور که خودت هم نوشتی، به وسیله `split` آن را با کاراکتر `\t` جدا کنیم، دیگر در عضو آخر خبری از کاراکتر مزاحم `\n` نیست:

```
line.split('\t')
```

```
<<<
```

```
[ 'Alanine', 'Ala', 'A', 'C3H5NO', '71.078', '71.03711378' ]
```

دانشجو: چه باحال! ولی در این مثال که برای ما فرقی نداشت؟ داشت؟

استاد: درست است که نه، اما به هر حال ریسک نکنید و این کار را کنید، ضرری ندارد. 😊
خب ما که نمی‌خواهیم فقط `split` کنیم، می‌خواستیم نام تک حرفی آمینواسید و همچنین جرم مولکولی آن را بیرون بکشیم. چه کنیم؟

دانشجو: راهی که به ذهنم می‌رسد اینه که حاصل `split` رو در یه متغیر مثلاً `splitted` بریزیم:

```
splitted = line.split('\t')
```

بعد دو متغیر جدید تعریف کنیم؛ مثلاً `aa` برای آمینواسید و `mw` برای جرم مولکولی اون، یعنی:

```
aa = splitted[2]
mw = splitted[4]
```

الآن در کنسول هم همین رو نوشتیم و چک کردم:

```
print(aa, mw)
A 71.078>>>
```

درست پیدا کرد!

استاد: دقیقاً یک راه همینیه است که نوشتی! راه دیگر هم این است که تمامی متغیرها را در یک خط استخراج کنیم، یعنی:

```
splitted = line.split('\t')
_, _, aa, _, mw, _ = splitted
```

چون به بقیه متغیرها نیاز نداشتیم، از نامی مثل `_` استفاده کردم که بی‌جهت برنامه

را شلوغ نکنیم.

خب تا اینجا توانستیم به خوبی نام آمینواسید و جرم مولکولی آن را استخراج کنیم.
در قدم بعد باید چه کنیم؟

دانشجو: برنامه رو ببندیم، چون کارمون تموم شد؟

استاد:!

دانشجو: برنامه رو نبندیم چون کارمون تموم نشد؟

استاد:!!

دانشجو: برنامه رو ذخیره کنیم؟

استاد:!!!!

دانشجو: آهان اصلاً قرار بود به دیکشنری که ساخته بودیم (دیکشنری aminoacids)،
این اطلاعات رو اضافه کنیم!

استاد: (نفس راحتی می کشد) عالی! چطور می توانیم یک key و value جدید
اضافه کنیم؟ همان طور که در جلسه قبل در قسمت تمرین ها انجام دادیم، کافی است
بنویسیم:

```
aminoacids[aa] = mw
```

با این کار اگر کلید ما (در اینجا نام آمینواسید) قبلاً در دیکشنری وجود داشته باشد،
جایگزین و اگر وجود نداشته باشد، ایجاد می شود.

دانشجو: آهان درست! یعنی الآن اگه مقدار دیکشنری رو کنسول چک کنیم، باید
نام آمینواسید و وزن مولکولی اون بهش اضافه شده باشه:

aminoacids

چه باحال؛ اضافه شده بود!

```
{ 'A': '71.078' }
```

ولی استاد! وزن مولکولی ما داخل دو تا علامت [”] قرار گرفته!

استاد: بله دقیقاً! این نشان می‌دهد که از چه جنسی است؟

دانشجو: از جنس رشته! آهان پس باید به عدد اعشاری یعنی float تبدیلش کنیم و این رو به دیکشنری مون اضافه می‌کردیم؛ یعنی:

```
aminoacids[aa] = float(mw)
```

حالا بذارین مقدار دیکشنری رو چک کنم:

aminoacids

```
>>> { 'A': 71.078 }
```

الآن درست شد و دیگه به صورت رشته نیست!

استاد: دقیقاً! پس حالا که در کنسول اسپایدر، مرحله به مرحله کار را انجام دادیم و دیدیم که جواب درستی گرفته‌ایم، کافی است که همین دستورات را در زیر حلقه For خودمان بنویسیم که برای تمام خطوط فایل‌مان این کار انجام شده است و تمامی آمینواسیدها به دیکشنری ما اضافه شوند:

```
path = r"E:/My Publications/Python Book/aminoacids/
data.txt"
```

```
aminoacids = {}
with open(path, 'r') as file:
    lines = file.readlines()
```

```

for line in lines:
    line = line.replace('\n', '')
    splitted = line.split('\t')
    _, _, aa, _, mw, _ = splitted
    aminoacids[aa] = float(mw)

print(aminoacids)

```

و درنهایت هم دیکشنری خود را چاپ کرده‌ایم که نتیجه را ببینیم.

دانشجو: چه باحال! عجب برنامه‌نویس خفنی شدما! بدون ذره‌ای خطا، کدم کار کرد! و الآن برنامه کامل هم حتماً عالی کار می‌کنه!!! بذارین Run کنم...استادا! توی ذوقم خورد و خطا داد! (☹️) من به درد برنامه‌نویسی نمی‌خورم!!! من نمی‌تونم...!!!

```

-----
-----
ValueError                                Traceback (most recent call last)
Cell In[27], line 11
      9         splitted = line.split('\t')
     10         _, _, aa, _, mw, _ = splitted
--> 11         aminoacids[aa] = float(mw)
     13 print(aminoacids)

```

ValueError: could not convert string to float: 'Average mass'

استاد: یک نفس عمیق!! خیلی کم می‌شود که یک برنامه همان اول درست و بدون خطا کار کند و تو تنها نیستی! حتی برنامه‌نویس‌های حرفه‌ای و باتجربه هم خطا دارند.

مهم این است که ببینیم و خطا را رفع کنیم! به من می‌گویی که برنامه در کدام خط خطا داده و چه خطایی داده است؟

دانشجو: چشم استاد! در خط ۱۱ یعنی این دستور خطا داده:

```
aminoacids[aa] = float(mw)
```

و خطا هم در آخرین خط نوشته شده:

ValueError: could not convert string to float: 'Average mass'

استاد: بسیار عالی! یعنی به تو گفته است که نمی‌تواند رشته را به float تبدیل کند؛ پس خطای ما احتمالاً در قسمت float(mw) بوده است و احتمالاً mw ما چیزی بوده که نمی‌توانسته به عدد تبدیل شود.

دانشجو: یعنی چه بوده؟

استاد: قبول داری که برنامه ما جایی که خطا داده، خط به خط اجرا شده و الآن متوقف شده است؟ پس می‌توانیم در قسمت کنسول ببینیم که مثلاً مقدار mw چیست؟

دانشجو: چه جالب! راست می‌گید استاد! بذارین در کنسول چک کنم:

```
mw
```

نوشت:

```
'Average mass'
```

چرا! مگه نباید mw من یک عدد و درواقع همون وزن مولکولی باشه؟

استاد: دقیقاً! در حلقه ما:

```
for line in lines:
```

هر بار line یکی از خطوط برنامه می‌شود و وقتی برنامه به ما خطا داده، برنامه

متوقف شده است. پس اگر در کنسول بنویسیم line می‌توانیم آخرین line که در آن خطا ایجاد شده است را ببینیم و شاید بتوانیم بفهمیم چرا mW مقدار عجیبی مثل 'Average mass' شده!

دانشجو: درستہ استاد... بذارین چک کنم:

```
line
<<<
'Residue\t3-letter\t1-letter\tFormula\tAverage    mass\
\tMonoisotopic mass'
```

این خروجی، را داده!

استاد: و این کدام خط فایل ماست؟

دانشجو: آهاااان! این همون خط اول فایل ماست! همون نام ستون‌ها! چطور می‌تونیم کاری کنیم که از عضو دوم به بعد ما بر سه بشه؟

استاد: یک راہ این است! یاد است کہ دستور زیر چہ می کرد؟

```
lines[1:]
```

دانشجو: آره استاد! از عضو دوم به بعد رو در قالب یه لیست جدید به ما می‌داد!

استاد: پس، مے، تو انیم به جای:

```
for line in lines:
```

بنو یسیم:

```
for line in lines[1:] :
```

تا حلقهٔ ما روی این لیست جدید، لیستی که عضو اول را ندارد، حرکت کند! پس

در واقع برنامه کامل ما شد:

```
path = r"E:/My Publications/Python Book/aminoacids/
data.txt"
```

```
aminoacids = {}
with open(path, 'r') as file:
    lines = file.readlines()

    for line in lines[1:]:
        line = line.replace('\n', '')
        splitted = line.split('\t')
        _, _, aa, _, mw, _ = splitted
        aminoacids[aa] = float(mw)

print(aminoacids)
```

دانشجو: و برنامه رو هم الآن Run کردم به خوبی اجرا شد و در انتها این دیکشنری زیبا رو برام چاپ کرد:

```
{'G': 57.051, 'A': 71.078, 'S': 87.077, 'P': 97.115,
'V': 99.131, 'T': 101.104, 'C': 103.143, 'I': 113.158,
'L': 113.158, 'N': 114.103, 'D': 115.087, 'Q': 128.129,
'K': 128.172, 'E': 129.114, 'M': 131.196, 'H': 137.139,
'F': 147.174, 'R': 156.186, 'Y': 163.173, 'W': 186.21}
```

چقدر آموزنده بود این جلسه! داشتم از خودم برای همچنین خطایی ناامید می شدم!

استاد: هر موقع خطا دیدی، خودت را به جای کامپیوتر بگذار و از طرفی هر تغییری که به نظرت می تواند به تو نشانه ای بدهد را در کنسول چک کن! همچنین یک رویکرد رایج دیگر هم این است که در خود حلقه For، متغیرهایی که می توانند به تو نشانه ای


```

را بدهند را پرینت کنی تا بتوانیم در حین اجرا متوجه خطا شویم، مثلاً:
for line in lines[1:]:
    print(line)

    line = line.replace('\n', ' ')
    splitted = line.split('\t')
    _, _, aa, _, mw, _ = splitted
    aminoacids[aa] = float(mw)

```

که مثلاً در هر بار حلقه، line را هم در خروجی نشان دهد تا متوجه شویم که کجا با خطا روبرو شده‌ایم و وقتی خطا را رفع کردیم، این پرینت‌های اضافه را حذف کنیم. در نهایت اگر هم دیدیم که هرچه فکر کردیم، متوجه نشدیم، هم سرچ خطا در گوگل و در نهایت هم چت‌بات‌های هوش مصنوعی کمک بسیار بزرگی هستند. همچنین احتمالاً وقتی که خطای خود را در گوگل جستجو می‌کنید، به سایت [stackoverflow](https://stackoverflow.com) برخورد می‌کنید! این سایت، بسیار در زمینه برنامه‌نویسی محبوب است و معمولاً سؤال شما را کسی پیش از شما در آن پرسیده و افراد داوطلب بسیار زیادی در این سایت هستند که به این سؤال‌ها جواب می‌دهند و شما می‌توانید ببینید که از چندین جوابی که به سؤال مدنظر شما داده شده، کدامشان لایک بیشتری خورده یا تیک سبز گرفته و از همان‌ها استفاده کنید.

من به ندرت به سؤال‌های برخورد می‌کنم که در این سایت کسی نپرسیده باشد، اما در دو مورد دیدم که سؤال خود را پیدا نمی‌کنم و سؤال خود را در آنجا پرسیدم و شاید برایتان جالب باشد که در کمتر از ۱ روز، جواب‌های بسیار کامل و مفیدی به آن داده شد. خودتان هم می‌توانید هرازگاهی به آن سر بزنید و اگر می‌توانید به سؤال‌های پاسخ دهید، پاسخ دهید. من خودم سعی می‌کنم این کار را انجام دهم. جدای از رزومه‌ای که این

سایت برای شما فراهم می‌کند و همچنین امتیازاتی که این سایت به شما می‌دهد (که انگیزه شما را بیشتر می‌کند)، کار بسیار لذت بخشی است. اگر دوست داشتید می‌توانید جستجو کنید PedramPorbah stackoverflow تا صفحهٔ پروفایل مرا پیدا کنید.

نوشتن اطلاعات موردنظرمان به صورت فایل با پایتون

تا اینجا توانستیم که جدول مدنظر خود را (یا فایل اکسل مدنظر) خود را به فایل متنی تبدیل کرده و آن را بخوانیم. آیا می‌توانیم با پایتون هم یک فایل جدید ایجاد کنیم؟ صد در صد!

فرض کنید که می‌خواهیم اطلاعاتی که به دست آوردیم را به فایل تبدیل کنیم. یک فایل متنی مشابه فایلی که داشتیم اما فقط نام تک حرفی آمینواسید و جرم مولکولی آن.

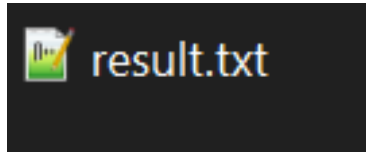
برای نوشتن فایل، دقیقاً مشابه باز کردن فایل عمل کنیم یعنی همان دستور `open` منتها اینجا mode را 'w' به معنای write می‌گذاریم:

```
with open('result.txt', 'w') as file:
```

همان‌طور که در قسمت‌های قبل در انواع فایل‌ها برایتان گفتم، هر پسوندی که دلمان می‌خواهد می‌توانیم برای فایل بگذاریم، اما من برای این که داریم یک فایل متنی می‌نویسیم، پسوند `.txt` را انتخاب کردم. برای نوشتن متن موردنظر خود می‌توانیم از متد `write` کمک بگیریم و چیزی که می‌خواهیم در فایل نوشته شود را به صورت یک رشته وارد کنیم؛ برای مثال:

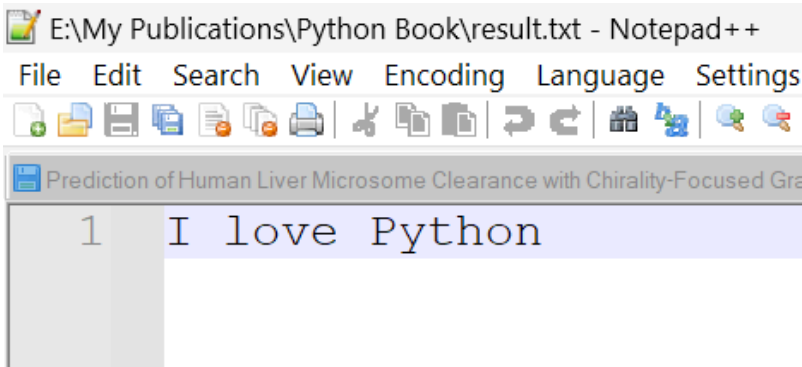
```
with open('result.txt', 'w') as file:
    file.write('I love Python')
```

اگر این برنامه را اجرا کنیم و برویم و چک کنیم، خواهیم دید محلی که کد پایتون ما قرار دارد، این فایل ایجاد شده است (شکل ۴۳):



شکل ۴۳- فایل ایجادشده به وسیله ما با پایتون

و اگر آن را با برنامه نوت پد باز کنیم، می بینیم که دقیقاً همان چیزی که نوشتیم در فایل وجود دارد (شکل ۴۴):



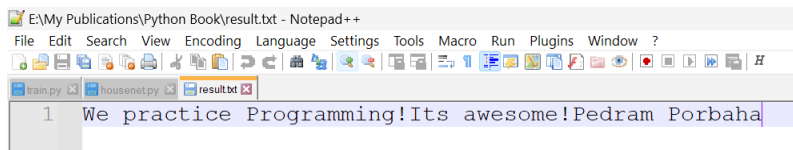
شکل ۴۴- متنی که نوشتیم در فایل دیده می شود.

شما هر چند بار که دلتان خواست می توانید از متد write استفاده کنید:

```
with open('result.txt', 'w') as file:
    file.write('We practice Programming!')
    file.write('Its awesome!')
    file.write('Pedram Porbaha')
```

و اگر مجدد فایل را باز کنید، می بینید که فایلی جدید، جایگزین فایل قبلی ما شده

است و تمامی متن‌هایی که نوشتیم پشت سر هم قرار گرفته‌اند (شکل ۴۵):



شکل ۴۵- می‌توان هر چند بار که دلمان خواست، متد write را صدا بزنیم.

دانشجو: ولی استاد! ما می‌خواهیم که هر خط، در خط جدید باشه! نه این جوری پشت سر هم!

استاد: درسته! چه کاراکتری در کامپیوتر، معنای خط جدید را می‌دهد؟

دانشجو: ... آهان همان \n!

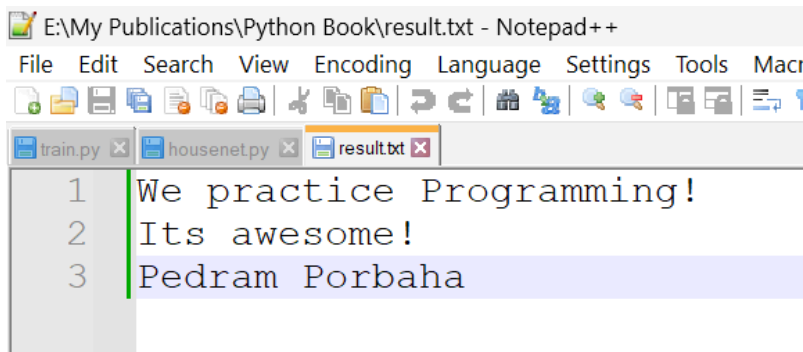
استاد: دقیقاً! پس اگر می‌خواهیم خطی جدید ایجاد شود، مشابه چیزی که تا الآن دیده بودیم، می‌توانیم آخر هر رشته (یا هر جایی که مدنظرمان است)، یک کاراکتر \n هم بگذاریم:

```
with open('result.txt', 'w') as file:
    file.write('We practice Programming!\n')
    file.write('Its awesome!\n')
    file.write('Pedram Porbaha\n')
```

و یا حتی می‌توانیم بعد از رشته خود، تنها یک کاراکتر '\n' را write کنیم:

```
with open('result.txt', 'w') as file:
    file.write('We practice Programming!')
    file.write('\n')
    file.write('Its awesome!')
    file.write('\n')
    file.write('Pedram Porbaha')
```

هر دو روش، دقیقاً یک نتیجه را می‌دهند که همان چیزی است که می‌خواستیم (شکل ۴۶):

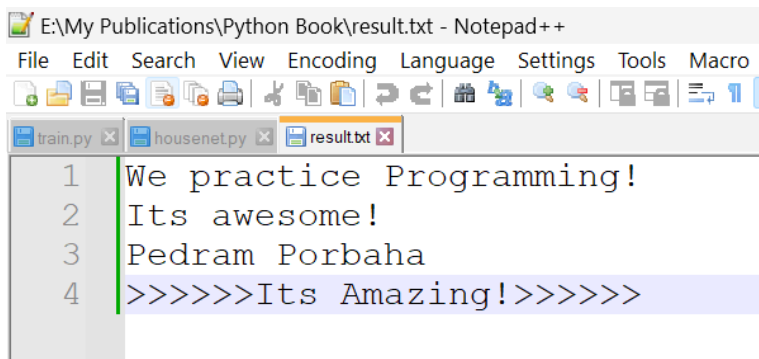


شکل ۴۶- کاراکتر `\n` به معنای خط جدید است.

فقط یک نکته بسیار مهم! همان‌طور که دیدید، حالت 'w'، یک فایل جدید ایجاد می‌کند، بنابراین اگر قبلاً فایلی هم‌نام با این اسم وجود داشته باشد، جایگزین می‌شود. حالت دیگری برای باز کردن یک فایل برای نوشتن وجود دارد که 'a' یا همان append است. با این حالت، اگر فایل از قبل وجود داشته باشد، جایگزین نمی‌شود بلکه چیزهایی که می‌نویسیم به آخر فایل اضافه می‌شوند. مثلاً اگر من بنویسم:

```
with open('result.txt', 'a') as file:
    file.write('\n>>>>>>Its Amazing!>>>>>>')
```

و بعد فایل را باز کنم (شکل ۴۷):



```

E:\My Publications\Python Book\result.txt - Notepad++
File Edit Search View Encoding Language Settings Tools Macro
train.py housenet.py result.txt
1 We practice Programming!
2 Its awesome!
3 Pedram Porbaha
4 >>>>>>Its Amazing!>>>>>>
    
```

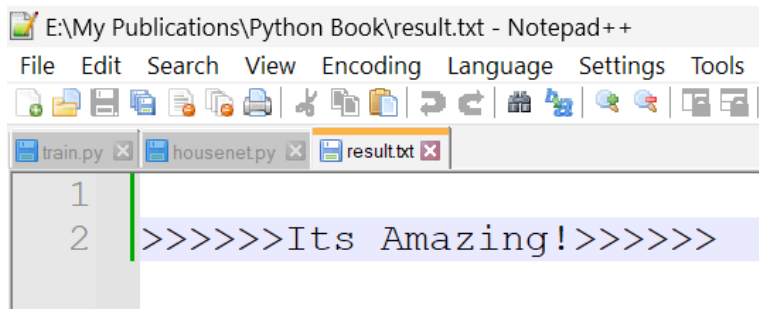
شکل ۴۷- با حالت 'a' اگر فایل همانم وجود داشته باشد، دیگر جایگزین نمی‌شود، بلکه چیزی که نوشتیم را به انتهای فایل اضافه می‌کند.

می‌بینید که چیزهایی که قبلاً نوشتیم سر جای خود هستند و متنی که نوشتیم به انتهای فایل اضافه شده است؛ اما اگر همین دستور را با حالت 'w' اجرا می‌کردم:

```

with open('result.txt', 'w') as file:
    file.write('\n>>>>>>Its Amazing!>>>>>>')
    
```

می‌بینید که اگر فایل نهایی را باز کنیم، انگار یک فایل جدید ایجاد شده است و جایگزین فایل قبلی شده است و بنابراین خبری از متن‌های قدیمی نیست (شکل ۴۸):



```

E:\My Publications\Python Book\result.txt - Notepad++
File Edit Search View Encoding Language Settings Tools
train.py housenet.py result.txt
1
2 >>>>>>Its Amazing!>>>>>>
    
```

شکل ۴۸- همان دستور با حالت 'w'

به همین دلیل اگر فایل مهمی است که به زحمت به دست آمده، به نظرم بهتر است که از حالت 'a' استفاده کنید و در غیر این صورت اگر فایل مهمی نیست یا می‌خواهد که فایل قبلی جایگزین شود، از همان حالت 'w' استفاده کنید.

دانشجو: استاد! در دستوری که نوشتیم، فقط نام فایل رو نوشتیم و فایل در همون پوشه‌ای که کد پایتون ما وجود داشت، ساخته شد. اگر بخوایم در یه پوشه یا آدرس دیگه ساخته بشه، باید چیکار کنیم؟

استاد: بسیار سؤال خوبی است! اگر شما به جای آدرس کامل، تنها نام فایل را بنویسید، در واقع دارید به پایتون می‌گویید که در همین پوشه، این فایل را ایجاد کن یا بخوان؛ اما اگر آدرس را به صورت کامل (یا به اصطلاح مرسوم‌تر، آدرس مطلق) را بدهید، پایتون در همان جایی که مدنظر تان است فایل را ایجاد می‌کند؛ مثلاً:

```
path = r"E:/My Publications/Python Book/aminoacids/result.txt"
```

```
with open(path, 'w') as file:
```

```
    file.write('\n>>>>>Its Amazing!>>>>>')
```

مثلاً الان چون آدرس را مانند قبل به صورت کامل وارد کردیم، پایتون مطابق آدرسی که نوشتیم در درایو E، در پوشه My Publications، در پوشه Python Book، در پوشه aminoacids فایل مورد نظر ما را می‌سازد.

خب برویم سراغ این که چگونه دیکشنری aminoacids که ایجاد کرده بودیم را به صورت فایل بنویسیم. در مراحل قبل به این دیکشنری رسیدیم:

```
{ 'G': 57.051,  
  'A': 71.078,  
  'S': 87.077,
```

'P': 97.115,
 'V': 99.131,
 'T': 101.104,
 'C': 103.143,
 'I': 113.158,
 'L': 113.158,
 'N': 114.103,
 'D': 115.087,
 'Q': 128.129,
 'K': 128.172,
 'E': 129.114,
 'M': 131.196,
 'H': 137.139,
 'F': 147.174,
 'R': 156.186,
 'Y': 163.173,
 'W': 186.21}

و حالا می‌خواهیم فایلی مثلاً به‌عنوان result.txt بسازیم که همین موارد را در خود داشته باشد. اول از همه باید چه کار کنیم؟ با دستور open یک فایل جدید ایجاد می‌کنیم؛ مثلاً:

```
with open('result.txt' , 'w') as file:
```

حال می‌خواهیم در کل دیکشنری خود حرکت کنیم! چگونه این کار را انجام دهیم؟ دانشجو: هر جا نام حرکت به گوش می‌رسد، نام حلقه For می‌درخشد!!! به نظر استاد چون هم به کلید و هم به مقدارها نیاز داریم، بهتره از همون متد items که من در جلسهٔ دیکشنری خیلی دوستش داشتم استفاده کنیم، یعنی:

```
for key, value in aminoacids.items():
```


استاد: بسیار عالی! اما سعی کن از نام‌های بامسماتری استفاده کنی! مثلاً در هر دور حلقه، key یکی از آمینواسیدها و value جرم مولکولی یکی از آن‌ها در دیکشنری ما می‌شود؛ پس بهتر است مثلاً از نام متغیر aa به جای key و از mw به جای value استفاده کنیم تا برنامه ما خوانا تر شود:

```
for aa, mw in aminoacids.items():
```

دانشجو: درسته استاد! حالا هم aa رو داریم و هم mw رو ... الآن چیکار کنیم؟

استاد: همان کاری که می‌خواستیم از اول انجام دهیم! نوشتن در فایل خود! تا الآن برنامه ما به این شکل درآمده:

```
with open('result.txt' , 'w') as file:
```

```
    for aa, mw in aminoacids.items():
```

پس می‌توانیم با متد write این دو مورد را در فایل خود بنویسیم تا هر دو حلقه aa و mw در فایل ما نوشته شوند.

دانشجو: دقیقاً استاد! من الآن انجامش می‌دم! کافیه که بنویسیم:

```
with open('result.txt' , 'w') as file:
```

```
    for aa, mw in aminoacids.items():
```

```
        file.write(aa, mw)
```

عالی شد! بذارین الآن چکش کنم! ای‌وای... خطا!!!!

TypeError

Traceback (most recent call last)

Cell In[12], line 3

```
1 with open('result.txt' , 'w') as file:
```

```
2     for aa, mw in aminoacids.items():
```

```
----> 3 file.write(aa, mw)
TypeError: TextIOWrapper.write() takes exactly
one argument (2 given)
```

استاد: (یک نفس عمیق) تو می‌توانی درستش کنی! چه خطایی گرفتیم؟

دانشجو: ... نوشته write فقط یه آرگومان می‌کرد ولی دو تا داده شده.

استاد: دقیقاً! این نشان می‌دهد که متد write فقط یک رشته را می‌تواند بگیرد و چاپ کند نه مثل دستور print که هر تعداد چیزی که دوست داشتیم را می‌توانستیم به آن بدهیم.

دانشجو: خب استاد الان چیکار کنم؟

یک راه این است که دو بار از دستور write استفاده کنیم. برای پایتون که فرقی ندارد! هر دو را در فایل تو می‌نویسد.

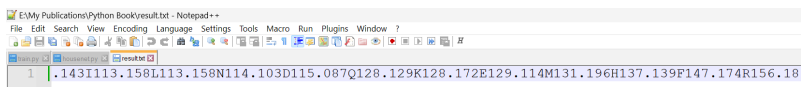
دانشجو: آهان یعنی مثلاً:

```
with open('result.txt' , 'w') as file:
    for aa, mw in aminoacids.items():
        file.write(aa)
        file.write(mw)
```

بذارین اجراش کنم... ای بابا بازم خطا!!

```
-----
-----
```

```
TypeError                                Traceback (most recent call last)
Cell In[13], line 4
      2 for aa, mw in aminoacids.items():
```

شکل ۴۹- فایل ایجادشده

استاد: به نظرت چرا این‌طور شده است؟

دانشجو: همممم ... (فکر می‌کند) نمی‌دونم استاد... بذارین یه نگاه دیگه به برنامه بندازم:

```
with open('result.txt' , 'w') as file:
    for aa, mw in aminoacids.items():
        file.write(aa)
        file.write(str(mw))
```

خب ... در هر دور حلقه هربار به یک aa و mw دسترسی پیدا می‌کنیم:::

```
for aa, mw in aminoacids.items():
```

و در این خط aa رو می‌نویسیم:

```
file.write(aa)
```

و بعدش mw رو در فایل می‌نویسیم:

```
file.write(str(mw))
```

... آهههه! ان! خب من اصلاً هیچ فاصله‌ای این وسط در فایل ننوشتم! به خاطر همین

هرچی دستش اومده پشت سر هم بدون هیچ‌گونه فاصله‌ای پرینت شده! پس می‌تونم

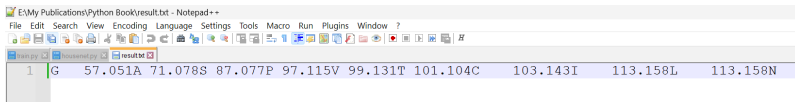
مثلاً یک کاراکتر " " که فاصله هست رو این بین بنویسم، یعنی:

```
with open('result.txt' , 'w') as file:
    for aa, mw in aminoacids.items():
        file.write(aa)
        file.write(' ')
        file.write(str(mw))
```

استاد: درود بر تو! البته معمولاً مرسوم است که به جای یک فاصله، یک کاراکتر `\t` یا همان `tab` را در بین اجزا می نویسند، یعنی:

```
with open('result.txt' , 'w') as file:
    for aa, mw in aminoacids.items():
        file.write(aa)
        file.write('\t')
        file.write(str(mw))
```

دانشجو: آهان درسته استاد! بذارین یه بار دیگه برنامه رو اجرا کنم... این فایل جدید (شکل ۵۰):

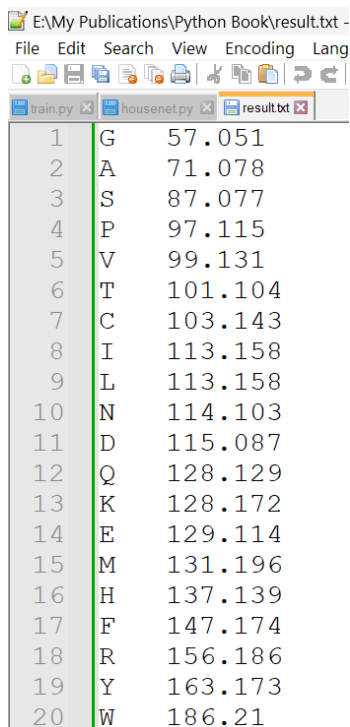


شکل ۵۰-فایل ایجادشده جدید

بهرتر شد ... اما باز پشت سر هم هستن. درحالی که باید هر ردیف در یه خط جدید باشه ... یه خط جدید ... یه ... خط جدید!!! آره خودش! وقتی آمینواسید و وزن مولکولی رو نوشتم، باید یه کاراکتر خط جدید یا همان `'n'` رو در فایل بنویسیم یعنی:

```
with open('result.txt' , 'w') as file:
    for aa, mw in aminoacids.items():
        file.write(aa)
        file.write('\t')
        file.write(str(mw))
        file.write('\n')
```

مجدداً برنامه رو اجرا کنم... باورم نمیشه!!! انگار درست شد 😊 (شکل ۵۱):



1	G	57.051
2	A	71.078
3	S	87.077
4	P	97.115
5	V	99.131
6	T	101.104
7	C	103.143
8	I	113.158
9	L	113.158
10	N	114.103
11	D	115.087
12	Q	128.129
13	K	128.172
14	E	129.114
15	M	131.196
16	H	137.139
17	F	147.174
18	R	156.186
19	Y	163.173
20	W	186.21

شکل ۵۱-فایل ایجادشدهٔ تصحیح‌شده

استاد: درود بر تو! دقیقاً به دنبال همین تفکر رفع اشکال هستیم که بتوانیم خطاهایمان را با خواندن خطا، تفکر، آزمون و خطا و نهایتاً جستجو پیدا کنیم. روشی که نوشتی کاملاً درست است، اما یک روش دیگر هم می‌تواند استفاده شود. یادت است زمانی که می‌خواستیم مثلاً به جای نوشتن:

```
print(aa, mw, 'is it')
```

می‌توانستیم از یک نوع نگارش استفاده کنیم که در نسخه‌های اخیر پایتون اضافه‌شده که به آن f-string می‌گویند؟

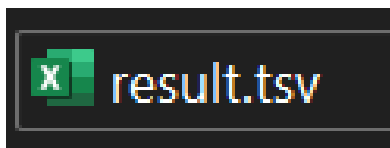
```
print(f'{aa} {mw} is it')
```

به عبارتی این روش، رشته مدنظر ما را به راحتی ایجاد می کند؛ پس می توانیم از این روش اینجا هم استفاده کنیم؛ یعنی بنویسیم:

```
with open('result.txt' , 'w') as file:
    for aa, mw in aminoacids.items():
        file.write(f'{aa}\t{mw}\n')
```

و دقیقاً به همان نتیجه برسیم!

دانشجو: عجب چیز خفنی بود استاد!!! خودش هم چون به رشته تبدیل شده دیگه نیازی به اون خط `str(mw)` هم نیست!! حتماً سعی می کنم ازش بیشتر استفاده کنم! استاد: من خودم خیلی از مواقع از این روش استفاده می کنم. راستی یک نکته جالب! همان طور که گفتیم پسوند فایل را می توانیم هر چیزی که دلمان خواست بگذاریم. مثلاً می توانیم این پسوند را `.tsv` بگذاریم (مثلاً `result.tsv`) و در این صورت فایلمان حتی با اکسل هم باز می شود (شکل ۵۲):

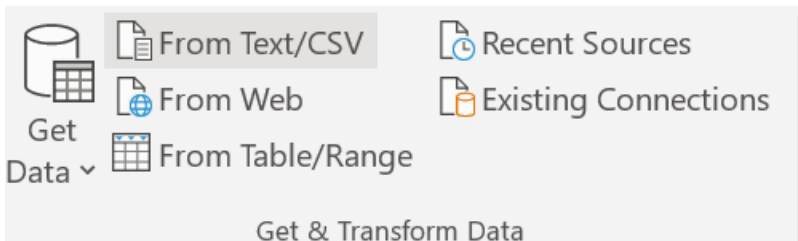


شکل ۵۲- پسوندها فقط نوع فایل را به کامپیوتر می فهمانند و نقش دیگری ندارند.

قسمت اختیاری: باز کردن فایل های متنی به صورت جدول در برنامه اکسل

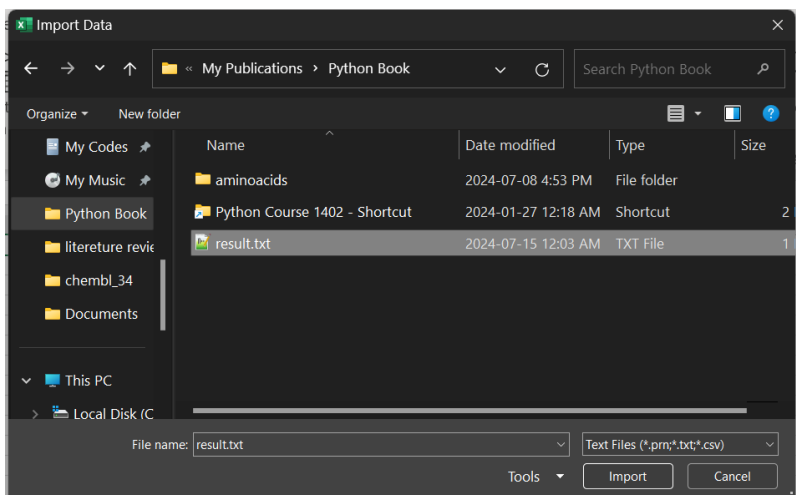
البته در همان اکسل می توان فایل های متنی را هم خواند و نیازی نیست حتماً به فرمت های رایج باشند. برای مثال برای این که همان فایل `result.txt` ایجاد شده خود را در پایتون بخوانیم، کافی است مثل قبل برنامه اکسل را باز کرده و یک `new`

file ایجاد کنیم. حالا در همان قسمت data به جای این روی گزینه web بزنیم. این بار روی گزینه From Text/CSV کلیک می‌کنیم (شکل ۵۳):



شکل ۵۳- برخلاف قبل، این بار روی گزینه FromText/CSV کلیک می‌کنیم.

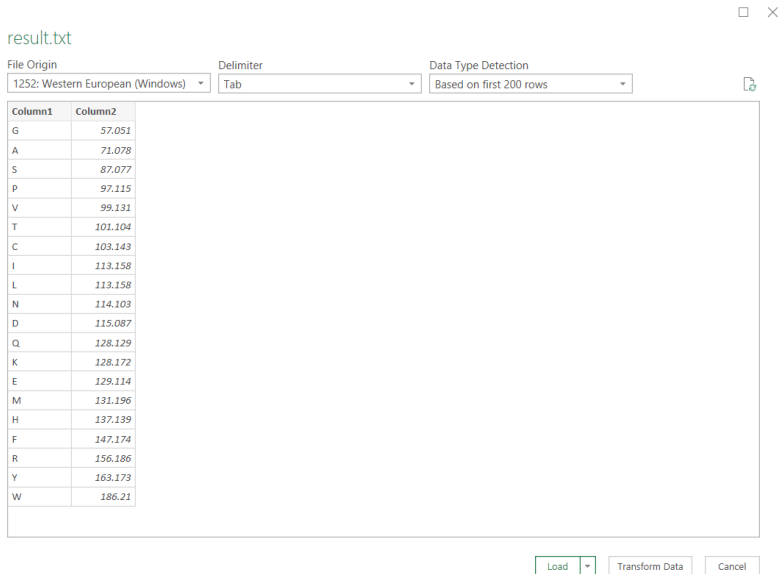
فایل result.txt خود را انتخاب می‌کنیم (شکل ۵۴):



شکل ۵۴- فایل متنی خود را انتخاب می‌کنیم (اگر فایل خود را پیدا نکردید، آن پایین سمت راست روی گزینه Text files کلیک کنید و از منویی که باز می‌شود All files را انتخاب کنید)

سپس پایتون از ما می‌پرسد که اجزای جدول شما با چه چیزی جدا شده است؛

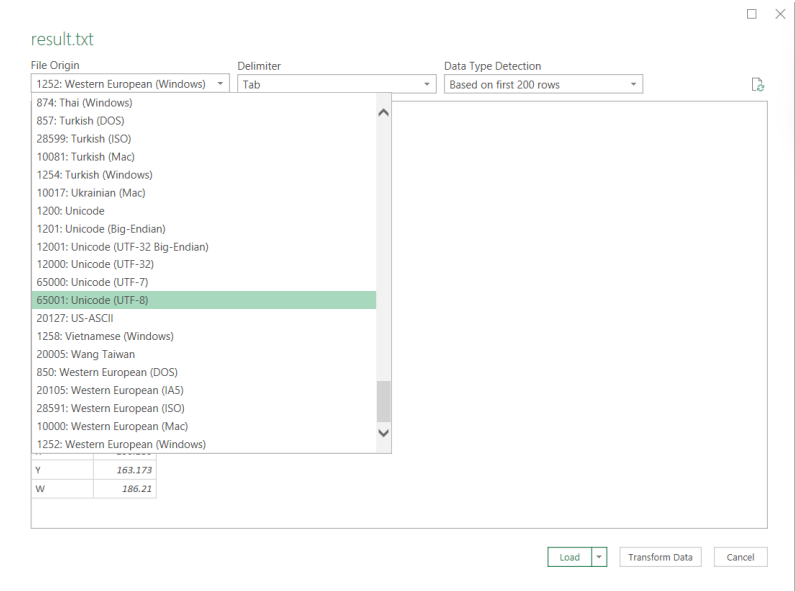
که چون ما با کاراکتر tab جدا کرده بودیم، آن را روی tab قرار دادیم. اگر با کاما جدا شده بود آن را روی comma قرار می دادیم (شکل ۵۵)؛ البته ممکن است در نسخه های مختلف اکسل این قسمت متفاوت باشد که اگر متفاوت بود، بر روی گزینه delimited کلیک کنید (در انتهای ویدئوی این جلسه، اکسل ما یک نسخه پایین تر است و می توانید آن را مشاهده کنید که البته تفاوت زیادی ندارد):



شکل ۵۵- چون اجزای جدول ما با tab جدا شده اند، delimiter یا همان جداکننده را روی کاراکتر tab می گذاریم.

و درنهایت روی گزینه Load می زنیم و فایل متنی ما به صورت جدول، وارد اکسل می شود (شکل ۵۶):

و ادامه دهید، اما اگر زبان استفاده شده، فارسی بود اگر این گزینه را تغییر ندهید، با اشکال عجیب و غریبی روبرو خواهید شد. اگر فایلتان فارسی (یا زبان دیگری بود) این گزینه را روی UTF-8 بزنید که یک انکودر متن جهانی است و فارسی را هم پشتیبانی می کند (شکل ۵۷):



شکل ۵۷ - اگر زبان فایلتان فارسی است، حتماً این گزینه را روی UTF-8 قرار دهید.

راستی، خوب شد این سؤال را مطرح کردی! در خود پایتون هم اگر بخواهید فایل فارسی را به روشی که یاد گرفتیم یعنی:

```
with open('result.txt' , 'r') as file:  
    lines = file.readlines()
```

با خطا روبرو خواهید شد و به شما خطایی نشان می‌دهد که مثلاً نتوانسته فلان کاراکتر (که یک کاراکتر فارسی است) را بخواند. دستور `open` یک آرگومان اختیاری به نام `encoding` دارد که برای فایل‌های فارسی باید حتماً آن را برابر با `'utf8'` قرار دهید:

```
with open('result.txt' , 'r', encoding='utf8') as file:  
    lines = file.readlines()
```

که دیگر به شما خطا نمی‌دهد.

دانشجو: چه باحال استاد! این نکته آخر به نظرم خیلی کاربردی بود!

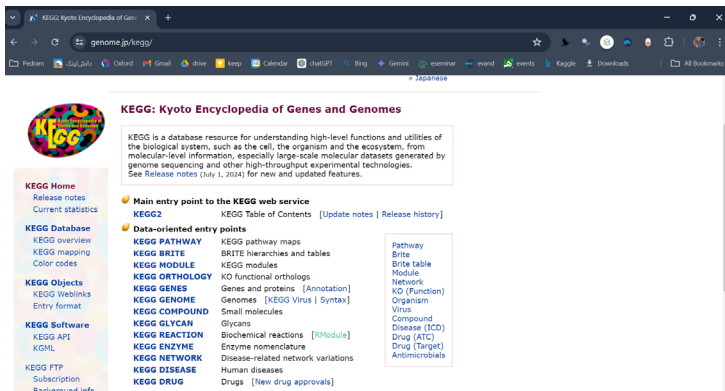
استاد: خواهش می‌کنم! درود بر شما که تا انتهای این جلسه همراه بودید. اگر موافق هستید، به سراغ بحث شیرین تمرین‌ها برویم!

تمرین‌های جلسه هفتم

در تمرین هر درس ممکن است که یک نکته جدید هم آموزش داده شود؛ پس حتماً توصیه می‌کنم آن‌ها را ببینید و انجامشان دهید. تمرین‌ها و به خصوص خطاهایی که دریافت می‌کنید و آزمون و خطاها (و همچنین سرچ‌هایی که در صورت نیاز انجام می‌دهید) بسیار بسیار به روند یادگیری برنامه‌نویسی شما کمک می‌کند. پاسخ نمونه برای تمرین‌ها نیز پس از آن‌ها قرار داده شده است.

تمرین‌ها

۱- سایت kegg را در گوگل جستجو کرده و وارد آن شوید و در قسمت Drugs روی New Drug Approvals کلیک کنید (شکل ۵۸):



شکل ۵۸- سایت KEGG (روی New drug approvals در پایین صفحه کلیک کنید)

و در صفحه بازشده، روی گزینه USA کلیک کنید تا داروهای تأیید شده سازمان FDA آمریکا را نشان دهد (شکل ۵۹):

New Drug Approvals in the USA

[Brite table menu | USA | Europe | Japan | Combined]

[New molecular entities and new therapeutic biological products | Cellular and gene therapy products | Other BLA products]

New molecular entities and new therapeutic biological products

Date	KEGG	ATC	Active ingredient	Drug name	Company	Remark
2024/7/2	D11500	N06DX05	Donanemab	Kisunla	Eli Lilly	BLA
2024/6/26	D11743		Ensfentrine	Ohtuvayre	Verona Pharma	NME
2024/6/20	D11696	L04A307	Crovalimab	Piasiky	Genentech, Inc	BLA
2024/6/18	D10909		Sofipronium bromide	Sofdra	Botanix SB Inc	BLA
2024/6/10	D11208	A05AX06	Eliofibranor	Iqirvo	Jpsen Pharmaceuticals	NME
2024/6/6	D09630	L01XX00	Immedastat sodium	Rytelo	Genon	NME
2024/5/16	D12234	(L01FX33)	Tarlatamab	Imdeltra	Ampen Inc	BLA
2024/4/26	D11510		Mavoxiafor	Xolremdi	X4 Pharmaceuticals	NME
2024/4/23	D12291		Tovorafenib	Ojemda	Day One Biopharmaceuticals	NME
2024/4/22	D12888		Ngapendesin alfa inbakicept	Ankiva	Altor Bioscience	BLA
2024/4/17	D12887		Peguciclanine acetate	Lumisight	Lumicell Inc	NME
2024/4/3	D08886	J01DI01	Cefiboprole medocaril	Zevtera	Basilea Pharmaceutica	NME
2024/3/29	D11641	L04A309	Danicopan	Voydeya	AstraZeneca	NME
2024/3/27	D11078	B03XA08	Vadadustat	Vafseo	Akebia Therapeutics	NME

شکل ۵۹ - روی USA در بالای صفحه کلیک کنید تا چنین جدولی را مشاهده کنید.

سپس برنامه‌ای بنویسید که داروها به همراه تاریخ ساخت و کمپانی سازنده را استخراج کند و به این سؤال پاسخ دهد که چه داروهایی به وسیله شرکت نوارتیس روانه بازار شده است؟

۲- در راستای سؤال قبل، به وسیله پایتون، یک فایل جدید ایجاد کنید که داروها به همراه شرکت سازنده و تاریخ ساخت در آن ذخیره شود.

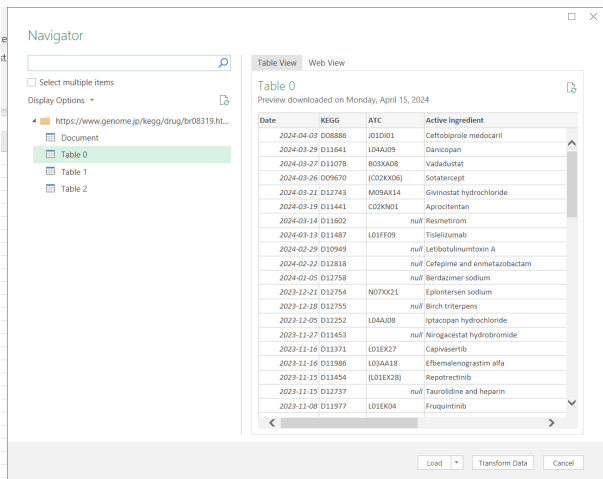
۳- تمرین اختیاری: چگونه می‌توان یک دیکشنری را به صورت فایل json ذخیره کرد؟ فایل json ذخیره شده را با برنامه نوت‌پد باز کنید و ساختار این فایل را ببینید (قاعدتاً نیاز به یک سرچ کوچک دارید 😊)

پاسخ تمرین‌های جلسه هفتم

اکیداً توصیه می‌کنم که اول خودتان تمرین‌ها را حل کنید و سپس به پاسخ‌ها مراجعه کنید. در ضمن این پاسخ‌ها تنها یکی از حالت‌های ممکن نوشتن برنامه هستند و شما ممکن است به شیوه‌ای دیگر برنامه را نوشته باشید و کاملاً صحیح هم باشد. همچنین برای مشاهده توضیحات شفاهی مربوط به این تمرین‌ها، ابتدای ویدئوی جلسه بعد را ببینید (لینک ویدئو در صفحه اول کتاب گذاشته شده است):

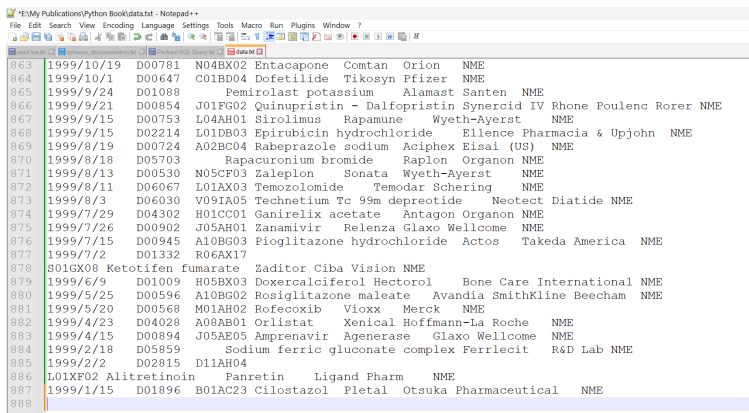
« پاسخ تمرین اول:

ابتدا نیاز داریم که اطلاعات را به صورت یک فایل متنی داشته باشیم تا با پایتون آن را بخوانیم. برای این کار می‌توانیم مشابه درس، یک فایل متنی ایجاد کنیم و با موس جدول را انتخاب کرده و در آن فایل کپی کنیم. گفتیم این روش خطاهای زیادی دارد؛ بنابراین از نرم‌افزار اکسل قسمت From Web کمک گرفته و لینک وبسایت را وارد می‌کنیم (شکل ۶۰):



شکل ۶۰- چند جدول توسط اکسل پیدا شده که جدول اول مدنظر ماست.

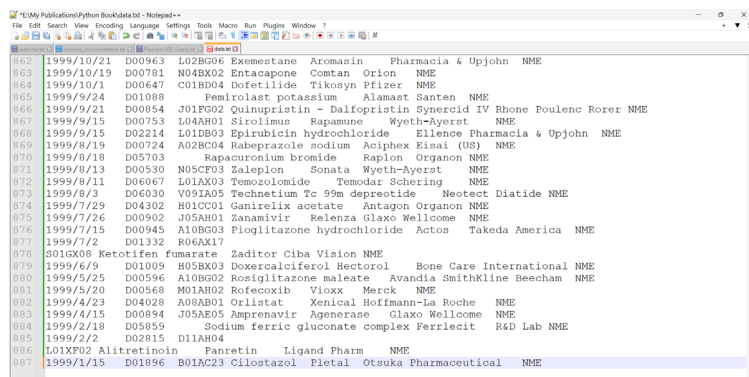
و در نهایت آن را به شکل فایل متنی مثلاً data.txt ذخیره کنیم و به چنین فایلی برسیم. توجه داشته باشید که در انتهای فایل، خط خالی وجود نداشته باشد، مثلاً به این شکل نباشد (شکل ۶۱):



863	1999/10/19	D00781	N04BX02	Entacapone	Comtan	Orion	NME
864	1999/10/1	D00647	C01BD04	Dofetilide	Tikosyn	Pfizer	NME
865	1999/9/24	D01088		Pemirolast potassium	Alamast	Santen	NME
866	1999/9/21	D00854	J01FG02	Quinupristin - Dalfopristin	Synercid IV	Rhone Poulenc Rorer	NME
867	1999/9/15	D00753	L04AH01	Sirolimus	Rapamune	Wyeth-Ayerst	NME
868	1999/9/15	D02214	L01DB03	Epirubicin hydrochloride	Ellence	Pharmacia & Upjohn	NME
869	1999/8/19	D00724	A02BC04	Rabeprazole sodium	Aciphex	Eisai (US)	NME
870	1999/8/18	D05703		Rapacuronium bromide	Raplon	Organon	NME
871	1999/8/13	D00530	N05CF03	Zaleplon	Sonata	Wyeth-Ayerst	NME
872	1999/8/11	D06067	L01AX03	Temozolomide	Temodar	Schering	NME
873	1999/8/3	D06030	V09IA05	Technetium Tc 99m depreotide	Neotect	Diatide	NME
874	1999/7/29	D04302	H01CC01	Ganirelix acetate	Antagon	Organon	NME
875	1999/7/26	D00902	J05AH01	Zanamivir	Relenza	Glaxo Wellcome	NME
876	1999/7/15	D00945	A10BG03	Pioglitazone hydrochloride	Actos	Takeda America	NME
877	1999/7/2	D01332	R06AX17				
878	1999/6/9	D01009	H05BX03	Doxercalciferol	Hectorol	Bone Care International	NME
879	1999/5/25	D00596	A10BG02	Rosiglitazone maleate	Avandia	SmithKline Beecham	NME
880	1999/5/20	D00568	M01AH02	Rofecoxib	Vioxx	Merck	NME
881	1999/4/23	D04028	A08AB01	Orlistat	Xenical	Hoffmann-La Roche	NME
882	1999/4/15	D00894	J05AE05	Ampranavir	Agenerase	Glaxo Wellcome	NME
883	1999/2/18	D00859		Sodium ferric gluconate complex	FerriLeit	R&D Lab	NME
884	1999/2/2	D02815	D11AH04				
885	1999/1/15	D01896	B01AC23	Cilostazol	Pletal	Otsuka Pharmaceutical	NME
886							

شکل ۶۱ - یک خط اضافه خالی مزاحم در انتهای فایل که می‌توانیم با زدن دکمه Backspace روی کیبورد آن را پاک کنیم.

بلکه این خط‌های اضافه را پاک کنید تا به این شکل دربیاید (شکل ۶۲):



862	1999/10/21	D00963	L02BG06	Exemestane	Aromasin	Pharmacia & Upjohn	NME
863	1999/10/19	D00781	N04BX02	Entacapone	Comtan	Orion	NME
864	1999/10/1	D00647	C01BD04	Dofetilide	Tikosyn	Pfizer	NME
865	1999/9/24	D01088		Pemirolast potassium	Alamast	Santen	NME
866	1999/9/21	D00854	J01FG02	Quinupristin - Dalfopristin	Synercid IV	Rhone Poulenc Rorer	NME
867	1999/9/15	D00753	L04AH01	Sirolimus	Rapamune	Wyeth-Ayerst	NME
868	1999/9/15	D02214	L01DB03	Epirubicin hydrochloride	Ellence	Pharmacia & Upjohn	NME
869	1999/8/19	D00724	A02BC04	Rabeprazole sodium	Aciphex	Eisai (US)	NME
870	1999/8/18	D05703		Rapacuronium bromide	Raplon	Organon	NME
871	1999/8/13	D00530	N05CF03	Zaleplon	Sonata	Wyeth-Ayerst	NME
872	1999/8/11	D06067	L01AX03	Temozolomide	Temodar	Schering	NME
873	1999/8/3	D06030	V09IA05	Technetium Tc 99m depreotide	Neotect	Diatide	NME
874	1999/7/29	D04302	H01CC01	Ganirelix acetate	Antagon	Organon	NME
875	1999/7/26	D00902	J05AH01	Zanamivir	Relenza	Glaxo Wellcome	NME
876	1999/7/15	D00945	A10BG03	Pioglitazone hydrochloride	Actos	Takeda America	NME
877	1999/7/2	D01332	R06AX17				
878	1999/6/9	D01009	H05BX03	Doxercalciferol	Hectorol	Bone Care International	NME
879	1999/5/25	D00596	A10BG02	Rosiglitazone maleate	Avandia	SmithKline Beecham	NME
880	1999/5/20	D00568	M01AH02	Rofecoxib	Vioxx	Merck	NME
881	1999/4/23	D04028	A08AB01	Orlistat	Xenical	Hoffmann-La Roche	NME
882	1999/4/15	D00894	J05AE05	Ampranavir	Agenerase	Glaxo Wellcome	NME
883	1999/2/18	D00859		Sodium ferric gluconate complex	FerriLeit	R&D Lab	NME
884	1999/2/2	D02815	D11AH04				
885	1999/1/15	D01896	B01AC23	Cilostazol	Pletal	Otsuka Pharmaceutical	NME

شکل ۶۲ - پس از حذف خط خالی مزاحم انتهای کار

برویم به سراغ پایتون!

```
data = {}  
with open('data.txt', 'r') as file:  
    lines = file.readlines()  
    for line in lines[1:]:  
        date, _, ingredient, drug_name, company,  
        _, _ = line.split('\t')  
        print(date, drug_name, company)
```

ابتدا این خط کد را نوشتیم که ببینیم درست موارد مورد نظر را استخراج کرده ایم یا نه، که با همچنین خطایی روبرو می شویم:

UnicodeDecodeError: 'utf-8' codec can't decode byte 0xe9 in position 6734: invalid continuation byte

شک می کنیم که شاید زبان بعضی از شرکت ها انگلیسی نباشد و برای همین خطا می گیریم. به همین دلیل encoding را روی حالت utf8 هم می گذاریم:

```
data = {}  
with open('data.txt', 'r', encoding='utf8') as file:  
    lines = file.readlines()  
    for line in lines[1:]:  
        date, _, ingredient, drug_name, compa-  
        ny, _, _ = line.split('\t')  
        print(date, drug_name, company)
```

و باز هم همان خطا در باز کردن فایل. در این گونه موارد اگر در گوگل جستجو کنید، آرگومانی وجود دارد به نام errors که اگر آن را برابر با ignore بگذارید از این گونه موارد که خطا دارند می گذرد و فایل را به درستی می خواند:

```
data = {}  
with open('data.txt', 'r', encoding='utf8', errors='ig-
```

```
nore') as file:
```

```
lines = file.readlines()
for line in lines[1:]:
    date, _, ingredient, drug_name, company,
    _, _ = line.split('\t')
    print(date, drug_name, company)
```

البته زمانی که پانداز را یاد گرفتیم کارمان راحت‌تر است اما فعلاً این آرگومان در بعضی مواقع به کمک ما می‌آید. خب حالا کد را تکمیل کنیم به دیکشنری خود نام دارو و شرکت را اضافه کنیم:

```
data = {}
with open('data.txt', 'r', encoding='utf8', errors='ignore') as file:
    lines = file.readlines()
    for line in lines[1:]:
        date, _, ingredient, drug_name, company,
        _, _ = line.split('\t')
        data['drug_name'] = company
```

و این باعث می‌شود که هر دارو به همراه کمپانی آن در دیکشنری ما ذخیره شود. می‌توانستیم از لیستی از لیست‌ها هم بهره ببریم، مثلاً:

```
data = []
with open('data.txt', 'r', encoding='utf8', errors='ignore') as file:
    lines = file.readlines()
    for line in lines[1:]:
        date, _, ingredient, drug_name, _, company,
        _ = line.split('\t')
```

```
data.append([drug_name, company])
```

که در این مورد فرقی ندارد و هر دو مورد جواب می‌دهد. من از همان دیکشنری استفاده کردم.

دانشجو: استاد اما من خطا می‌گیرم:

ValueError Traceback (most recent call last)

Cell In[22], line 5

```
3 lines = file.readlines()
4 for line in lines[1:]:
----> 5     date, _, ingredient, drug_name, company, _,
_ = line.split('\t')
6     print(line)
7     if company in data:
```

ValueError: not enough values to unpack (expected 7, got 3)

استاد: همان‌طور که در خطا می‌بینیم، نوشته است که انتظار هفت مقدار را داشته ولی سه مقدار را گرفته است. الان برنامه ما اجرا شده، به خطا برخورد و متوقف شده است. درست است؟ پس اگر متغیر line را در کنسول چک کنیم، آخرین خط را می‌توانیم ببینیم:

line

```
>>>
```

```
'2010-08-13\tD09687\t"G03AD02\n'
```

دانشجو: این خط چرا این‌قدر کوتاهه!

استاد: می‌توانیم همان فایل متنی خودمان یعنی data.txt را باز کنیم و با زدن کلید Ctrl+F عبارت 2010-08-13 را جستجو کنیم؛ اما راه دیگری هم وجود دارد. مثلاً نظرت چیست شماره خط را هم پرینت کنیم تا بدانیم در کدام خط، خطا

داشتیم؟ یعنی:

```
data = {}
with open('data.txt', 'r', encoding='utf8', errors='ignore') as file:
    lines = file.readlines()

    for idx, line in enumerate(lines[1:]):
        print(idx)
        date, _, ingredient, drug_name, _, company, _ = line.split('\t')

        if company in data:
            data[company].append(drug_name)
        else:
            data[company] = []
```

دانشجو: بله استاد فکر خوبیه! تمامی خطوط رو نوشت و آخرین خط، خط 590 هست. البته چون از صفر شروع شده منظورش خط ۵۹۱ هست. بذارین در فایل اصلی، خط ۵۹۱ رو نگاه کنیم (شکل ۶۳):

582	2011-01-21	D09699	N06AX24	Vilazodone hydrochloride	ViiVbryd Forest Laboratories	NME
583	2011-01-18	D09384	Spinosad	Natroba ParaPRO LLC	NME	
584	2011-01-14	D10014	V09AB03	Ioflupane I-123 Datscan GE Healthcare	NME	
585	2010-11-15	D08914	L01XX41	Eribulin mesylate	Eribulin mesylate	Einai NME
586	2010-11-10	D09015	H01AC06	Tesamotelin Egrifta	Theratechnologies	NME
587	2010-10-29	D08884	J01DI02	Ceftaroline fosamil	Ceftaroline fosamil for injection	Cereza NME
588	2010-10-28	D04820	N05AE05	Lurasidone hydrochloride	Lurasidone HCl	Sunovion Pharmaceuticals NME
589	2010-10-19	D07082	B01AE07	Dabigatran etexilate mesylate	Pradaxa	Boehringer Ingelheim NME
590	2010-09-21	D04187	L04AE01	Fingolimod hydrochloride	Fingolimod HCl oral capsules	Novartis Pharmaceu
591	2010-09-14	D09316	M04AX02	Pegloticase Krystexxa	Savant Pharmaceuticals	BLA
592	2010-08-13	D09687	"G03AD02"			
593	G03XB02"	Ulipristal acetate	Elia	Laboratoire HRA Pharma	NME	
594	2010-07-30	D10012		Incobotulinumtoxin A	Xeomin	Merz Pharmaceuticals GmbH BLA
595	2010-07-28	D06552	S01GX11	Alcaftadine Lastacast	Vistakon Pharmaceuticals LLC	NME
596	2010-06-17	D10432	L01CD04	Cabazitaxel Jevtana	Sanofi-Aventis US	NME
597	2010-06-01	D03684	M05BX04	Denosumab	Prolia	Amgen BLA
598	2010-05-24	D03207	A16AB07	Alglucosidase alfa2	Lumizyme	Genzyme
599	2010-05-06	D10461	G03AB08	Estradiol valerate - Dienogest	Natazia	Bayer HealthCare Pharmaceuticals NM
600	2010-03-30	D01993	C05BB02	Polidocanol Asclera	Chemische Fabrik Kreussler & Co GmbH	NME
601	2010-03-18	D07130	A16AA05	Carglumic acid	Carbaglu	Orphan Europe NME

شکل ۶۳ - خط ۵۹۲

البته خط ۵۹۲... آهان درسته چون که خط اول رو هم نخوندیم... این خط نصفه‌اش
رفته خط بعدی!!!

استاد: درست است. می‌توانی آخر این خط کلید Delete را بزنی تا خط بعدی به
این خط بچسبد، یعنی (شکل ۶۴):

1999-12-17	D01205	N05CM18	Dexmedetomidine hydrochloride	Precedex	Abbott Labs	NME
1999-12-17	D00589	"J01MA16S01AE06"	Gatifloxacin	Tequin	Bristol-Myers Squibb	NME
1999-12-10		Mequinol - tretinoin	Solage	Bristol-Myers Squibb		
1999-12-10	D00874	"J01MA14				
S01AE07"		Moxifloxacin hydrochloride	Avelox	Bayer	NME	
1999-12-08	D01646	V08CA06	Gadoversetamide	Optimark	Mallinckrodt	NME
1999-12-03	D02908	L01XD04	Aminolevulinic acid hydrochloride	Levulan	Kerastick	DUSA Pharm
1999-11-30	D00709	N03AX14	Levetiracetam	Keppra	UCB Pharm	NME
1999-11-18	D08402		Poractant alfa	Curosurf	Dey Labs	NME
1999-10-27	D00900	J05AH02	Oseltamivir phosphate	Tamiflu	Hoffmann-La Roche	NME
1999-10-21	D00963	L02BG06	Exemestane	Aromasin	Pharmacia & Upjohn	NME
1999-10-19	D00781	N04RX02	Entacapone	Comtan	Orion	NME

شکل ۶۴- زدن دکمه Delete

و اگر تا انتهای فایل پایین بیایی، با چشم می‌بینی که کدام خط‌ها اشتباه شده‌اند و
می‌توانی آن‌ها را اصلاح کنی. حالا دوباره امتحان کن.

دانشجو: استاد الان درست شد! ولی این جوری که خیلی سخته استاد!

استاد: درسته! احتمالاً در خود سایت مشکلی وجود داشته یا در تبدیل اکسل به
فایل این مشکل پیش آمده است. انشا... بعداً با کتابخانه پانداس به راحتی در یک خط،
همان فایل اکسل را می‌خوانیم.

« پاسخ تمرین دوم:

کافی است در ادامه برنامه قبل دیکشنری خود حرکت کنیم و هروقت در نام
کمپانی، لغت Novartis وجود داشت، یکی به کیسه count اضافه کنیم:

```
count = 1
for drug_name, company in data.items():
    if 'novartis' in drug_name.lower():
```

```
count += 1
```

```
print(f'Novartic drug counts: {count}')
```

سؤال تکمیلی:

سؤال یک را به گونه‌ای تغییر دهید که در دیکشنری نهایی، نام هر کمپانی به‌عنوان کلید و لیست داروهای آن شرکت به‌عنوان value باشد.

پاسخ:

کافی است در دیکشنری ایجادشده، هر بار به شرکتی برخورد کردیم که در دیکشنری ما وجود نداشت، آن را به دیکشنری اضافه کنیم و اگر شرکت وجود داشت، دارو را به آن append کنیم یعنی:

```
data = {}
with open('data.txt', 'r', encoding='utf8', errors='ignore') as file:
    lines = file.readlines()
    for idx, line in enumerate(lines[1:]):
        date, _, ingredient, drug_name, _, company, _ = line.split('\t')
        if company not in data:
            data[company] = []

        data[company].append(drug_name)
```

```
print(data)
```

و در نتیجه کار می‌بینیم که یک دیکشنری داریم که نام هر شرکت به همراه لیستی از داروهای آن شرکت به ما نشان داده می‌شود.

« پاسخ تمرین سوم:

همان طور که دیدیم، می توانیم یک دیکشنری یا لیست یا ... را به صورت یک جدول متنی ذخیره کنیم، اما آیا راهی وجود دارد که بتوان دیکشنری را مستقیماً به صورت یک فایل ذخیره کرد؟ پاسخ بله است. یکی از انواع فایل های بسیار محبوب json است که اگر آن را با نوتپد باز کنید، دقیقاً یک دیکشنری است. حال چگونه می توان دیکشنری خودمان را به صورت یک فایل json ذخیره کنیم؟ اگر به انگلیسی جستجو کنید، برای مثال `how to write a dictionary as json file in Python`، می توانیم ببینیم که چگونه این کار را به راحتی انجام دهیم. می توانیم این کار را با کتابخانه `json` در پایتون انجام دهیم (کتابخانه ای که در پایتون به صورت پیش فرض وجود دارد و نیاز به نصب ندارد و کافی است آن را `import` کنیم). این کتابخانه دو دستور دارد که فعلاً کار ما را راه می اندازد:

```
import json
```

که دستور `dumps` برای ذخیره دیکشنری مدنظرمان به صورت فایل و `load` برای خواندن فایل `json` مدنظرمان استفاده می شود. هیچ چیزی مانند عمل کردن، برنامه نویسی را به ما نمی آموزد؛ پس برویم در عمل ببینیم:

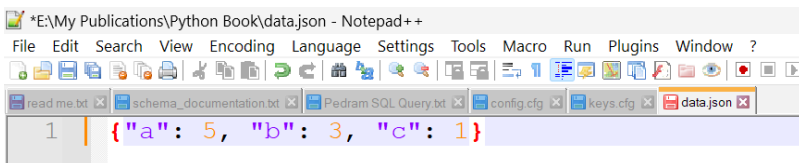
نوشتن `json`: فرض کنید می خواهیم یک دیکشنری فرضی با نام `data` را به صورت `json` ذخیره کنیم:

```
data = {'a': 5,
        'b': 3,
        'c': 1}
```

برای نوشتن آن به صورت فایل مثلاً `data.json` کافی است بنویسیم:

```
with open('data.json', 'w') as file:
    file.write(json.dumps(data))
```

و اگر در کامپیوتر خود نگاه کنید، می‌بینید که یک فایل جدید به نام data.json ایجاد شده است که می‌توانید آن را با نوت‌پد باز کنید و می‌بینید عین همان ساختار دیکشنری است (شکل ۶۵):



شکل ۶۵- اگر فایل ایجادشده را با نوت‌پد باز کنیم، می‌بینیم همان ساختار دیکشنری است.

حالا اگر خواستیم این فایل json را سریعاً به صورت دیکشنری بخوانیم، این بار تابع load از کتابخانه json به ما کمک می‌کند:

```
with open('data.json', 'r') as f:
    data = json.load(f)
```

```
print(data)
```

بسیاری از سایت‌ها داده‌های خود را به صورت json برای دانلود قرار داده‌اند و خودتان هم اگر خواستید به سرعت دیکشنری خود را ذخیره کنید، می‌توانید از این ساختار استفاده کنید. حتی اگر data به صورت لیست بود هم باز می‌توانید از همین دو دستور برای ذخیره آن استفاده کنید که در آن صورت پس از خواندن فایل json، خروجی به صورت یک لیست خواهد بود:

```
data = [1, 2, 3]
with open('data.json', 'w') as file:
    file.write(json.dumps(data))
```


If you try you **may** fail,
if you **don't** try
you're **guaranteed** to fail !

Jesse Jackson

• فصل هشتم •

تابع! راهی برای کارهای تکراری و منظم شدن!

سلام و درود دوستان! ما تا الآن با بسیاری از موارد اساسی پایتون آشنا شدیم؛ متغیرها، شرط‌ها، تکرار کارها با حلقه، ساختار داده‌های لیست و دیکشنری و در انتها خواندن و نوشتن فایل‌های متنی با پایتون. ساختار داده دیگری به نام set را نیز باهم کار خواهیم کرد که بسیار ساده و اما کاربردی است؛ اما در ابتدای این جلسه سراغی مفهومی مهم به نام function یا همان تابع می‌رویم. وقتی این‌ها را یاد گرفتیم زمان آشنایی با کتابخانه‌های کاربردی numpy، pandas و matplotlib (به قول یکی از اساتیدم، جناب دکتر قاسمی، سه تفنگدار پایتون) است و در نهایت هم یک سری کاربرد عملی مانند regex (کتابخانه re) و گذری بر webscraping و شی‌گرایی که اگر عمری باقی بود در جلد بعدی کتاب به آن‌ها می‌پردازیم؛ اما با مواردی که یاد گرفتید، حتی خودتان هم می‌توانید با یک ویدئو در اینترنت یا حتی خرید یک دوره کوتاه با این موارد برای تحلیل داده با پایتون آشنا شوید.

برویم به سراغ تابع!

چرا تابع؟

تا الآن، هر برنامه‌ای که نوشتیم ساختاری خطی داشت؛ یعنی چه؟ به این معنی که

از اول شروع می‌شد و خط به خط جلو می‌رفت (حتی اگر حلقه هم داشتیم باز پس از اتمام حلقه، به سراغ خط بعدی می‌رود) تا به انتهای برنامه برسیم. برای نمونه برنامه‌ای قرار بود مثلاً یک رشته DNA را دریافت کرده و مقدار GC_content آن را حساب کند (به عبارتی تعداد حروف G و C در کل این رشته):

```
seq = 'ATCG'
count = 0
for nu in seq:
    if seq in ['G', 'C']:
        count += 1
```

```
print(count)
```

حال فرض کنید می‌خواهیم همین کار را برای یک رشته پروتئین دیگر هم انجام دهیم. باید تمام این کد را مجدداً کپی کنیم:

```
seq2 = 'CTTA'
count = 0
for nu in seq:
    if seq in ['G', 'C']:
        count += 1
```

```
print(count)
```

اگر بخواهیم چند بار دیگر این کار را کنیم چه؟ می‌بینیم که باید کپی کنیم!!!

دانشجو: ولی استاد همون‌طور که گفتین هر موقع کار تکراری داشتیم می‌تونیم از حلقه استفاده کنیم؛ مثلاً به لیست از رشته‌های پروتئین خود داشته باشیم:

```
seqs = ['ATCG', 'CTTA']
```

و بعد روی این حلقه حرکت کنیم که هر بار seq، یکی از اینا بشه:

```

for seq in seqs:
    count = 0
    for nu in seq:
        if seq in ['G', 'C']:
            count += 1

print(count)

```

استاد: درست است اما اگر خواستیم مثلاً یک جای دیگر همین کار را انجام دهیم و وزن مولکولی یک رشته پروتئین را حساب کنیم، مجدداً باید همین کار را انجام دهیم و دوباره این برنامه را بنویسیم. از طرفی می‌بینیم که برنامه ما هم پراز دندانده شده و خوانایی آن برای کسی که اولین بار این کد را می‌بینید کاهش پیدا کرده است. کاش می‌شد مثلاً به کامپیوتر یک بار می‌گفتیم که به این شکل مقدار GC_content را حساب می‌کنند و هر وقت نیاز بود، تنها به اون می‌گفتیم وزن مولکولی این پروتئین را حساب کن و دیگر لازم نبود هر بار به کامپیوتر از ابتدا مثل الان این کد را کپی می‌کردیم. در زندگی روزمره ما هم همین‌طور هستیم. به عبارتی شبیه استاد شما که به شما یک بار کاری را آموزش می‌دهد و سری‌های بعد تنها به شما می‌گوید که این کار را انجام دهید و دوباره نمی‌آید صفر تا صد کار را قدم‌به‌قدم برای شما توضیح دهد.

دانشجو: چقدر خوب می‌شد اگه این‌طور بود و اول کار به برنامه برای کامپیوتر رو می‌نوشتیم و سری‌های بعد فقط می‌گفتیم فلان کار رو انجام بده.

استاد: و دقیقاً اینجاست که تابع به درد می‌خورد و کار ما را بسیار راحت‌تر و تمیزتر می‌کند و از طرفی می‌توان از آن در برنامه‌های دیگر نیز استفاده کرد.

دانشجو: پس باید چیز خیلی باحالی باشه!

چگونه تابع بنویسیم؟

برای یادگرفتن تابع هیچ چیزی بهتر از این نیست که مثال بزنیم. فرض کنید می‌خواهیم یک تابع بنویسیم که دو عدد را گرفته و جمع آن دو را نشان دهد. اسم این تابع را هم Jam (همان جمع خودمان) می‌گذاریم!

```
def jam(a, b):  
    result = a + b  
    return result
```

دانشجو: استاد این رو نوشتم و اجرا کردم، اتفاقی نیفتاد که!

استاد: درست است! زیرا الآن فقط تابع را تعریف کردیم و از آن استفاده نکردیم! مثلاً اگر بعد از آن بنویسیم (این را در کنسول امتحان کن) چه اتفاقی می‌افتد؟

```
jam(1, 3)
```

دانشجو: آهان! الآن نوشت ۴! چه جالب یعنی انگار یه دستور ایجاد کردیم که دو تا عدد رو می‌گیره و حاصل جمع رو برمی‌گردونه که هر موقع خواستیم می‌تونیم ازش استفاده کنیم؟ بذارین امتحان کنم:

```
jam(1, 2)
```

درسته، نوشت ۳! چه باحال!

استاد: دقیقاً! همان‌طور که دیدید یک تابع ورودی و خروجی دارد. ورودی را دریافت می‌کند و با انجام کاری که برایش تعریف می‌کنیم درنهایت مورد مدنظر ما را خروجی می‌دهد (یا اصطلاحاً return می‌کند). برای تعریف تابع از لغت کلیدی def استفاده می‌کنیم (که به معنای define یا همان تعریف کردن است) و نامی که می‌خواهیم برای دستور را می‌نویسیم. درنهایت دستورات مدنظر را می‌نویسیم و درنهایت:

def نام تابع:

...

return چیزی که می‌خواهیم.

مثلاً فرض کنید می‌خواهیم یک تابع بنویسیم که یک عدد را گرفته و حاصل جمع تمام اعداد کوچک‌تر از آن را بنویسد. اول از همه بیایید آن را به شکل یک برنامه عادی (بدون تابع بنویسیم) و فکر کنید می‌خواهیم این کار را برای اعداد کوچک‌تر از ۱۰ انجام دهیم. چه کنیم؟

دانشجو: بذارین الآن من می‌گم! ما اول از همه نیاز داریم که به تمام اعداد کوچک‌تر از ۱۰ دسترسی داشته باشیم. این کار رو می‌تونیم با حلقه for انجام بدیم:

```
for num in range(1, 10):
```

و یادمه که یک کیسه خالی داشتیم که هر عدد رو داخل آن می‌انداختیم که باعث می‌شد در انتهای کار حاصل جمع تمامی اعداد رو داشته باشیم. پس من یه متغیر مثلاً به نام total با مقدار اولیه صفر، قبل از حلقه تعریف می‌کنم:

```
total = 0
```

```
for num in range(1, 10):
```

```
    total += i
```

و در نهایت total همان حاصل جمع مدنظر ماست.

استاد: بسیار عالی! حال به من می‌گویی که این برنامه تو چه کار می‌کند؟ به عبارتی چه نوع ورودی می‌گیرد و چه نوع خروجی تحویل می‌دهد؟

دانشجو: ... ورودی خاصی که نمی‌گیرد، اما ... در نهایت حاصل جمع اعداد کوچک‌تر از ۱۰ را می‌دهد.

استاد: بسیار عالی! پس می‌توانیم یک تابع تعریف کنیم که این کار را انجام دهد. اسم این تابع را چه بگذاریم؟ مثلاً `sum_calculate` خوب است؟

دانشجو: اسم خوبیه استاد!

استاد: بسیار عالی! پس از همان روش تعریف تابع استفاده می‌کنیم:

```
def sum_calculator():
```

دانشجو: استاد چرا بعد از اسم تابع، یه پرانتز خالی گذاشتیم؟

استاد: چون همان‌طور که خودت گفتی، اینجا ورودی خاصی نداریم؛ اما در تعریف تابع در پایتون حتماً لازم است که این پرانتز باشد. حالا اگر مثل تابع قبلی (تابع `jam`) ورودی داشتیم، در داخل پرانتز می‌نوسیم، وگرنه یک پرانتز باز و بسته خالی می‌گذاریم.

و حالا زیرمجموعهٔ تابع (حواسمان هست که در پایتون هر جا منظورمان زیرمجموعهٔ یک چیز است، باید با یک دندانۀ این کار را انجام دهیم):

```
def sum_calculator():  
    total = 0  
    for i in range(1, 10):  
        total += i
```

و در نهایت برنامهٔ ما قرار است چه چیزی را به عنوان خروجی تحویل دهد؟

دانشجو: حاصل جمع رو ... یعنی همون `total` رو.

استاد: دقیقاً! پس می‌توانیم در نهایت این را `return` کنیم:

```
def sum_calculator():  
    total = 0  
    for i in range(1, 10):
```

```
total += i
```

```
return total
```

دانشجو: چه جالب! حالا اگر من این برنامه رو اجرا کنم و بعداً از این دستور جدید استفاده کنم؛ مثلاً بنویسم:

```
sum_calculator
```

باید حاصل جمع اعداد ۱ تا کوچک‌تر از ۱۰ رو به من بده. بذارین در کنسول امتحان کنم... عه! چرا نوشت:

```
<function __main__.sum_calculator(>
```

استاد: دستورها در پایتون، برای اجرا شدن نیاز به چه علامتی دارند؟

دانشجو:... آهان پرانتز! یعنی:

```
sum_calculator()
```

حالا درست شد و نوشت ۴۵ که درست هم هست.

استاد: اگر ما اسم تابع را به‌تنهایی بنویسیم و از پرانتز استفاده نکنیم، به معنی این است که آن را اجرا نکرده‌ایم، بلکه منظورمان خود آن تابع است به همین دلیل در خروجی با جملهٔ `function` روبرو شده بودیم.

حال تابع ما حاصل جمع اعداد کوچک‌تر از ۱۰ را حساب می‌کند. اگر بخواهیم به‌جای عدد ۱۰، عدد مدنظر را از کاربر به‌عنوان یک ورودی تابع بگیریم، چه کنیم؟ می‌توانیم در تعریف تابع در داخل پرانتز، اسم متغیر مدنظرمان که می‌خواهیم از کاربر بگیریم را بنویسیم، یعنی:

```
def sum_calculator(number):
```

و بعداً در تابع خود از مقدار این متغیر استفاده کنیم، مثلاً:

```
def sum_calculator(number):  
    total = 0  
    for i in range(1, number):  
        total += i  
  
    return total
```

و حال اگر بخواهیم تابع خود را صدا بزنیم، لازم است که عدد را هم وارد کنیم
یعنی مثلاً:

```
sum_calculator(10)
```

که به عنوان خروجی نوشت ۴۵.

دانشجو: چه جالب! یعنی مثلاً الآن من می‌تونم بنویسم:

```
sum_calculator(5)
```

مقدار حاصل جمع اعداد کوچک‌تر از ۵ رو به من می‌ده و می‌تونم هربار که دلم
خواست، به راحتی به جای نوشتن کل برنامه، فقط اسم تابع رو به همراه عدد مدنظرم
صدا بزنم.

استاد: دقیقاً! و این ویژگی بسیار مفید است. راستی تا یادم نرفته می‌تونیم به جای:

```
sum_calculator(5)
```

به این شکل هم تابع را صدا بزنیم:

```
sum_calculator(number=5)
```

که خوانایی بیشتر باشد اما در این مثال ما فرقی ندارد.

دانشجو: یعنی مثلاً در همون مثال اول که تابع جمع کردن بود:

```
def jam(a, b):  
    result = a + b  
    return result
```

```
print(jam(1, 2))
```

می‌تونستم بنویسم:

```
print(jam(a=1, b=2))
```

... آره انگار جواب داد! چه جالب!

یک سؤال! ما فهمیدیم که یه تابع می‌تونه یک یا چند ورودی داشته باشه یا اصلاً ورودی نداشته باشه. اگر بخوایم چند خروجی داشته باشیم چیکار کنیم؟

استاد: سؤال بسیار خوبی است! کافی است بعد از return با علامت کاما، هر تعداد چیزی که خواستید را بنویسید، مثلاً:

```
def sum_calculator(number):  
    total = 0  
    for i in range(1, number):  
        total += i  
  
    return total, number
```

در این صورت اگر بنویسیم:

```
sum_calculator(10)
```

چه چیزی خروجی می‌گیریم؟

دانشجو: استاد من امتحان کردم؛ نوشت:

```
(45, 10)
```

استاد: دقیقاً! یعنی مقدار خروجی ما خودش یک tuple (توپل) است و ما می‌توانیم مثلاً بنویسیم:

```
result, x = sum_calculator(10)
```

که همان‌طور که قبلاً دیدیم، دو متغیر جدید به نام result و x تعریف می‌شوند که مقدار ما در آن ذخیره می‌شوند.

دانشجو: استاد اگر یه تابع return نداشته باشه چی می‌شه؟

استاد: مثلاً فرض کنید در تابعی که نوشتیم، return را برداریم:

```
def sum_calculator(number):  
    total = 0  
    for i in range(1, number):  
        total += i
```

در این صورت انگار تابع ما return None دارد، یعنی به عبارتی این تابع که نوشتیم معادل این است:

```
def sum_calculator(number):  
    total = 0  
    for i in range(1, number):  
        total += i  
  
    return None
```

یعنی مثلاً اگر تابع را صدا بزنیم و خروجی را در یک متغیر به نام result بریزیم:

```
result = sum_calculator(10)
```

مقدار result در این صورت برابر با None است. حالا یک سؤال؛ برنامه زیر چه کار می‌کند؟

```
def sum_calculator(number):
    total = 0
    for i in range(1, number):
        total += i

    print(total)
```

```
result = sum_calculator(10)
print(result)
```

دانشجو: بذارین امتحان کنم... برای من این رو نوشت:

۴۵

None

چرا ... خب یه پرینت که در دل خود تابع داریم ... آهان همون طور که گفتید اگر چیزی رو return نکنیم، انگار به صورت پیش فرض نوشتیم return None؛ یعنی انگار برنامه ما به این شکل بوده:

```
def sum_calculator(number):
    total = 0
    for i in range(1, number):
        total += i

    print(total)
    return None
```

```
result = sum_calculator(10)
print(result)
```

به همین دلیل خروجی تابع ما همون None بوده و به همین دلیل متغیر result ما حاوی None است.

استاد: بسیار عالی!

متغیرهای محلی و جهانی!

بیایید یک برنامه جدید باز کنیم و همان تابعی که همان اول کار نوشته بودیم (jam) که حاصل جمع دو عدد وارد شده را حساب می کرد و خروجی می داد را بنویسیم. به نظر تو این برنامه چه می کند؟

```
def jam(a, b):  
    result = a + b  
    return result
```

```
ans = jam(1, 2)  
print(ans)  
print(b)
```

دانشجو: خب این قسمت...

```
jam(1, 2)
```

که می شه حاصل جمع ۱ و ۲، یعنی همون ۳ که در متغیر ans ذخیره کرده و در نهایت پرینت کردیم؛ و بعد هم گفته a رو پرینت کن. a هم که همون عدد اوله؛ پس باید برای ما روی صفحه چاپ بشه ۱. بذارین امتحان کنم... عه چرا خطا داد!!

```
----> 6 print(a)
```

NameError: name 'a' is not defined

نوشته a تعریف نشده! یعنی چی!

استاد: دقیقاً! نکته بسیار مهمی که باید حواسمان باشد.

تابع ما یک ورودی می‌گیرد و خروجی را حساب می‌کند. همین! تمامی متغیرهایی که در تابع استفاده شده‌اند، بعد از استفاده پاک می‌شوند. متغیرهایی مانند `a` و `b` و `result`، متغیرهایی محلی (`local`) هستند و فقط در داخل تابع تعریف شده‌اند و در داخل تابع می‌توان به آن‌ها دسترسی داشت؛ مثلاً می‌بینید که در کد زیر بدون هیچ مشکلی به مقدار `b` دسترسی داریم:

```
def jam(a, b):
    result = a + b
    print(b)
    return result
```

```
ans = jam(1, 2)
print(ans)
```

خروجی:

```
2
3
```

همان‌طور که می‌بینید در داخل تابع چون هنوز به متغیر `b` دسترسی داریم و پاک نشده است، دستور

```
print(b)
```

به خوبی اجرا می‌شود اما اگر در بیرون از تابع بنویسیم:

```
print(b)
```

با خطا مواجه خواهیم شد. اگر علاوه بر `result` به مقدار `b` هم بیرون از تابع نیاز داریم، بهتر است که آن‌ها را هم `return` کنیم:


```
def jam(a, b):
    result = a + b
    return result, b
```

```
ans, b = jam(1, 2)
print(ans)
print(b)
```

حالا به عنوان بیشتر بدانید، به نظرتان خروجی کد زیر چیست؟

```
result = 100
def jam(a, b):
    result = a + b
    print(result)
    return result
```

```
ans = jam(1, 2)
print(result)
```

نکته بسیار مهم اینجا است که ما در ابتدا یک متغیر به نام `result` با مقدار ۱۰۰ تعریف کرده ایم. این یک متغیر `global` است، به این معنا که متغیری است که بیرون از تابع تعریف شده است. وقتی به این خط می‌رسیم یعنی:

```
ans = jam(1, 2)
```

تابع `jam` را صدا زده ایم، یک متغیر `local` (محلی) `result` دیگر تعریف شده است که با متغیر `result` قبلی فرق دارد!! پایتون وقتی به این خط می‌رسد:

```
print(result)
```

اول در محوطه خودش نگاه می‌کند. چون ما در محیط داخل تابع هستیم، پایتون نگاه می‌کند و می‌بیند در متغیرهای محلی (درون تابع) آیا متغیری با نام `result` وجود دارد؟ بله وجود دارد به همین دلیل آن را پرینت می‌کند، یعنی عدد ۳. پایتون کارش با تابع تمام شده و در نهایت تابع مقدار را `return` می‌کند که ما آن را در متغیر `ans` ذخیره کردیم و دیگر کارمان با تابع تمام شده است و تمامی متغیرهای درون تابع هم پاک شده است:

```
ans = jam(1, 2)
```

حالا به خط آخر می‌رسیم:

```
print(result)
```

الآن در فضای بیرون از تابع هستیم (فضای `global`). پایتون نگاه می‌کند و می‌بیند که آیا متغیری به نام `result` در این محیط تعریف شده دارد؟ بله دارد (در همان خط اول قبل از تابع تعریفش کردیم) و به همین دلیل همان را پرینت می‌کند که عدد ۱۰۰ بود. در صورتی که تعریفش نکرده بودیم، با خطای این که `result` تعریف نشده است روبرو می‌شدیم. پس خروجی کلی برنامه می‌شود:

```
3
```

```
100
```

به‌صورت کلی در فضای داخل تابع (`local`) می‌توان به متغیرهای بیرون از تابع (`global`) دسترسی داشت اما برعکس خیر.

البته این جزو مباحث بیشتر بدانید بود اما بد نبود که از آن مطلع باشیم که این خطا برایتان عجیب نباشد.

یک مثال دیگر در حیطه ژنتیک!

برویم برنامه‌ای که ابتدای کار برای محاسبه GC content نوشتیم را به صورت تابع بنویسیم. ابتدا یک با دیگر به برنامه‌ای که نوشتیم دقت کنید:

```
seq = 'ATCG'
count = 0
for nu in seq:
    if seq in ['G', 'C']:
        count += 1

print(count)
```

هدف از این برنامه چیست؟

دانشجو: ... خب همان طور که گفتیم برنامه‌ای نوشتیم که یک رشته رو گرفته و تعداد حروف G و C اون رو بشماره.

استاد: بسیار عالی! پس موافقی نام تابعمان را مثلاً gc_content_calculate بگذاریم؟ یعنی:

```
def gc_content_calculate()
```

دانشجو: اسم خوبیه! سریع می فهمیم که منظور از این تابع چیه.

استاد: حالا می توانی برای من بگویی که برنامه ما چه چیزی را می گیرد و چه چیزی تحویل می دهد؟

دانشجو: استاد، رشته DNA (همون seq) را ورودی می گیره و count رو به عنوان خروجی تحویل می ده.

استاد: بسیار عالی. پس تابع ما یک ورودی دارد:

```
def gc_content_calculate(seq):
```

و دستورات داخل تابع هم که عین همان برنامه قبلی ما است:

```
def gc_content_calculate(seq):
```

```
    count = 0
```

```
    for nu in seq:
```

```
        if seq in ['G', 'C']:
```

```
            count += 1
```

و در نهایت هم که باید count را به عنوان خروجی تحویل دهد:

```
def gc_content_calculate(seq):
```

```
    count = 0
```

```
    for nu in seq:
```

```
        if seq in ['G', 'C']:
```

```
            count += 1
```

```
    return count
```

اگر بخواهیم از تابع خود استفاده کنیم، کافی است آن را صدا بزنیم مثلاً:

```
result = gc_content_calculate('ATCG')
```

```
print(result)
```

که البته من print را هم نوشتم که نتیجه برای ما هم چاپ شود.

حالا یک بار دیگر به برنامه ما قبل از نوشتن به صورت تابع نگاه کنید:

```
seq = 'ATCG'
```

```
count = 0
```

```
for nu in seq:
```

```
    if seq in ['G', 'C']:
```

```
        count += 1
```

```
print(count)
```

باید کلی فکر کنیم تا متوجه شویم که این برنامه قرار است چه کار کند، اما در برنامه‌ای که به صورت تابع نوشتیم، همین که اسم تابع را ببینیم، می‌فهمیم هدف از این تابع چیست و چه چیزی را خروجی می‌دهد و دیگر نیازی نیست درگیر کشتی گرفتن با کد شویم تا بفهمیم چه می‌کند. حتی نکته جالب این که می‌توانیم برای تابع خود یک help هم بنویسیم؛ مثلاً:

```
def gc_content_calculate(seq):  
    '''  
    This function get a DNA sequence and calculate  
    the count of G or C nucleotides  
    '''  
    count = 0  
    for nu in seq:  
        if seq in ['G', 'C']:  
            count += 1  
  
    return count
```

و حالا جالب است که اگر در کنسول بنویسیم:

```
help(gc_content_calculate)
```

می‌بینیم که همان توضیحی که نوشتیم (اصطلاحاً doc string نوشته شده برای تابع) به ما نشان داده می‌شود:

```
gc_content_calculate(seq)  
    This function get a DNA sequence and calculate  
    the count of G or C nucleotides
```

دانشجو: چه خفن!

استاد: یا مثلاً یک نکتهٔ دیگر که آن هم جالب است! فرض کنید تابعی یا توابعی که نوشتیم را در یک فایل پایتون مثلاً به اسم `main.py` ذخیره کنیم.

حالا یک فایل جدید پایتون در همان پوشه می‌سازیم. حال می‌توانیم از توابعی که در فایل `main` نوشتیم استفاده کنیم. چگونه؟ کافی است آن را `import` کنیم!

```
import main
```

و حالا دقیقاً انگار یک کتابخانه را وارد کرده‌ایم و می‌توانیم به دستورات آن دسترسی داشته باشیم، مثلاً:

```
main.gc_content_calculate('ATTC')
```

و یا می‌توانستیم به جای وارد کردن کل فایل، فقط همین تک دستور را وارد کنیم:

```
from main import gc_content_calculate
gc_content_calculate('ATTC')
```

دانشجو: چه جالب! یعنی انگار خودمون به کتابخانه نوشتیم!

استاد: دقیقاً! و بعد می‌توانید این کتابخانه‌ای که نوشتید را در سایت `pip` بارگذاری کنید که بقیه هم بتوانند از آن استفاده کنند.

نوشتن برنامهٔ ترجمهٔ پروتئین

همان طور که می‌دانید در بدن رشتهٔ `DNA` با طی مراحل می‌تواند به پروتئین ترجمه شود که بعداً پروتئین حاصل نقش بسیاری در بدن ایفا می‌کند. بیاید با یک مثال این موضوع را باهم مرور کنیم. فرض کنید که رشتهٔ `DNA` زیر را داریم:

ATGGCTTAG

در بدن ما ریبوزوم، رشته را سه حرف سه حرف می‌خواند. بیاید از سمت چپ شروع کنیم.

ATGCTTAG

سه حرف اول ATG است (به این سه حروف، کدون هم گفته می‌شود). این رمز در بدن به آمینواسید M ترجمه می‌شود:

M

سپس ریبوزوم به سراغ سه حرف بعدی می‌رود یعنی:

ATGCTAAG

سه حرف GCT در بدن ما به آمینواسید A ترجمه می‌شود که به آمینواسید قبلی (یعنی M) اضافه می‌شود. پس رشته پروتئین تولیدی ما فعلاً این است:

MA

سپس سه حرف بعدی:

ATGGCTTAG

این سه حرف، TAG در بدن ما به کدون پایان تعبیر می‌شود، یعنی عملیات ترجمه را ختم کن. پس عملیات ترجمه ختم شده و رشته پروتئین تولیدی ما شد:

MA

این شد خلاصه‌ای از این که عملیات ترجمه پروتئین از DNA در بدن ما چگونه انجام می‌شود. برای جلوگیری از پیچیدگی مطلب، دیگر وارد این مرحله که ابتدا DNA به یک رشته دیگر رونویسی می‌شود و بعد ترجمه می‌شود، نشدیم.

خب! حال می‌خواهیم یک برنامه بنویسیم که رشته DNA را که به آن دادیم، خودش آن را به پروتئین ترجمه کرده و به ما تحویل دهد. برای این کار نیاز به جدول کدون‌ها و آمینواسید معادل آن‌ها داریم. کافی است در گوگل جستجو کنیم و این جدول را بیابیم. برای مثال من به انگلیسی جستجو کردم:

DNA codons and amino acid table

اما برای سادگی بیشتر، من از یک چت بات هوش مصنوعی (مثل چت جی پی تی) خواستم که یک دیکشنری که کلیدهای آن کدون های DNA و مقادیر آن نام تک حرفی آمینواسید معادل آن باشد را به من بدهد:

Please make a python dict of DNA codones as keys and one letter aminoacid as value

و این هم خروجی:

```
codon_to_amino_acid = {
    'ATA': 'I', 'ATC': 'I', 'ATT': 'I', 'ATG': 'M',
    'ACA': 'T', 'ACC': 'T', 'ACG': 'T', 'ACT': 'T',
    'AAC': 'N', 'AAT': 'N', 'AAA': 'K', 'AAG': 'K',
    'ATA': 'I', 'ATC': 'I', 'ATT': 'I', 'ATG': 'M',
    'CTA': 'L', 'CTC': 'L', 'CTG': 'L', 'CTT': 'L',
    'CCA': 'P', 'CCC': 'P', 'CCG': 'P', 'CCT': 'P',
    'CAC': 'H', 'CAT': 'H', 'CAA': 'Q', 'CAG': 'Q',
    'CGA': 'R', 'CGC': 'R', 'CGG': 'R', 'CGT': 'R',
    'GTA': 'V', 'GTC': 'V', 'GTG': 'V', 'GTT': 'V',
    'GCA': 'A', 'GCC': 'A', 'GCG': 'A', 'GCT': 'A',
    'GAC': 'D', 'GAT': 'D', 'GAA': 'E', 'GAG': 'E',
    'GGA': 'G', 'GGC': 'G', 'GGG': 'G', 'GGT': 'G',
    'TAA': '*', 'TAG': '*', 'TGA': '*',
    'TAC': 'Y', 'TAT': 'Y', 'TAA': '*', 'TAG': '*',
    'TCA': 'S', 'TCC': 'S', 'TCG': 'S', 'TCT': 'S',
    'TTC': 'F', 'TTT': 'F', 'TTA': 'L', 'TTG': 'L',
    'TAC': 'Y', 'TAT': 'Y', 'TAA': '*', 'TAG': '*',
    'TGC': 'C', 'TGT': 'C', 'TGA': '*', 'TGG': 'W'
}
```

که کدون های پایان را با علامت * مشخص کرده است. خب برویم ببینیم که چطور

برنامه ترجمه را بنویسیم. بیاید از یک رشته DNA فرضی کوچک استفاده کنیم، مثلاً همان که داشتیم:

`seq = 'ATGGCTTAG'`

حالا باید چه کنیم؟

دانشجو: ...نمی دونم استاد...

استاد: خودت به عنوان یک انسان، وقتی یک رشته DNA رو در جلویت می گذارند، چطور آن را به پروتئین تبدیل می کنی؟

دانشجو: ... من ... اول از همه سه تا حرف اول رو می بینم. بعد اون سه حرف رو در جدول نگاه می کنم که معادل چه آمینواسیدی و اون حرف رو یک جا یادداشت می کنم. بعد میرم سه حرف دیگه رو می بینم معادل چه آمینواسیدی و به حرف قبلی که نوشته بودم اضافه می کنم و همین کار رو تا آخر انجام میدم.

استاد: درود بر تو! حالا بیا همین چیزی که نوشتی را مرحله به مرحله بنویسیم:

۱- سه حرف اول را می بینم؛

۲- در جدول معادل آن را می بینم؛

۳- به رشته ای که دارم اضافه می کنم؛

و این مراحل را تا آخر رشته تکرار می کنم.

خب؛ بیا به سراغ اصل اساسی خودمان برویم. هر برنامه پیچیده، زمانی یک برنامه ساده بوده که کار می کرده است. بیا ببینیم چطور سه حرف اول را می توانیم پیدا کنیم:

`seq = 'ATGGCTTAG'`

دانشجو: آگه می‌خواستیم بگیم حرف اول، کافی بود بنویسیم:

```
seq[0]
```

استاد: اگر می‌خواستیم بنویسیم از حرف اول تا سوم؟

دانشجو: آهان یادم اومد. می‌شد نوشت:

```
seq[0:2]
```

بنارین در کنسول چک کنم. نوشت:

```
AT
```

یعنی خود شماره ۲ رو نگرفت. پس باید بنویسیم:

```
seq[0:3]
```

آهان الان درست شد:

```
'ATG'
```

استاد: بسیار عالی! حالا اگر بخواهیم مثلاً سه حرف بعدی را پیدا کنیم، چطور؟

دانشجو:... منطقاً می‌شه:

```
seq[3:6]
```

که شد:

```
'GCT'
```

استاد: و سه حرف بعدی؟

دانشجو:

```
seq[6:9]
```

که می‌شه:

```
'TAG'
```

استاد: پس انگار با چنین الگویی می‌توانیم سه حرف را جدا کنیم:

```
seq[i:i+3]
```

که i هم به ترتیب ۰، و بعد ۳ و بعد ۶ و ... باشد.

دانشجو: آره چه جالب استاد!

استاد: چطور می‌توانیم i را ۰ و بعد ۳ و بعد ۶ و ... کنیم؟ به عبارتی سه‌تا سه‌تا حرکت کنیم؟

دانشجو: حرکت رو می‌تونستیم با حلقهٔ for انجام بدیم:

```
for i in range(len(seq)):
    print(i)
```

اما کدی که نوشتیم i رو به ترتیب ۰، ۱، ۲ و ... می‌ذاره، در حالی که می‌خواستیم سه‌تا سه‌تا حرکت کنه...

استاد: یادت است که می‌توانستیم در range گام حرکت را هم تعریف کنیم؟

```
for i in range(0, len(seq), 3):
    print(i)
```

دانشجو: شروع، پایان و گام ... خودش استاد!!! پس می‌تونیم در حلقهٔ خودمون ازش استفاده کنیم:

```
for i in range(0, len(seq), 3):
    seq[i:i+3]
```

استاد: دقیقاً! نظرت چیست که این سه حرف جدا شده را نیز در یک متغیر ذخیره کنیم، مثلاً codon:

```
seq = 'ATGGCTTAG'
for i in range(0, len(seq), 3):
```

```
codon = seq[i:i+3]
print(codon)
```

که می‌بینیم به خوبی دارد در هر بار حلقه، سه حرف سه حرف جدا می‌کند:

ATG

GCT

TAG

حالا در قدم بعد باید چه کنیم؟

دانشجو: هدفمون این بود که در دیکشنری خودمون این رمز رو ببینیم که معادل کدوم آمینواسیده.

استاد: دقیقاً! که می‌توانیم از دیکشنری خود استفاده کنیم:

```
seq = 'ATGGCTTAG'
for i in range(0, len(seq), 3):
    codon = seq[i:i+3]
    aa = codon_to_amino_acid[codon]
    print(aa)
```

که من نتیجه را در یک متغیر به نام aa ذخیره کردم و می‌بینیم که به خوبی دارد آمینواسیدها را شناسایی می‌کند:

M

A

*

دانشجو: استاد! ولی ما نیاز داشتیم که یه رشته نهایی (مثلاً در متغیری به نام protein) داشته باشیم. چطوری این‌ها رو به هم اضافه کنیم؟

استاد: پایتون ویژگی جالبی دارد که اگر دو رشته را به هم با علامت جمع اضافه

کنیم، به هم وصل می شوند. مثلاً:

‘a’ + ‘b’

به عنوان خروجی می دهد:

‘ab’

پس می توانیم برای مثال قبل از شروع حلقه یک رشته خالی مثلاً به نام protein تعریف کنیم و هر آمینواسید را که یک رشته است، با آن جمع کنیم:

```
protein = ''  
for i in range(0, len(seq), 3):  
    codon = seq[i:i+3]  
    aa = codon_to_amino_acid[codon]  
    protein = protein + aa
```

```
print(protein)
```

دانشجو: چه خفن استاد! اما وقتی من برنامه رو اجرا می کنم علامت * هم در پروتئین ما وجود داره:

M A *

استاد: اگر یادت باشد در دیکشنری خود برای کدون های پایان، علامت * را تعریف کردیم. برنامه ما باید وقتی به این کدون پایان رسید، عملیات ترجمه را متوقف کند. چطور می توان یک حلقه را شکست و از آن بیرون آمد؟

دانشجو: آهان الان یادم اومد! دستور break! می تو نیم بگیم که اگه به کدون پایان رسیدیم (یعنی کدون برابر با * بود)، حلقه رو بشکن و ادامه برنامه رو بده، یعنی:

```
seq = 'ATGGCTTAG'
```

```

protein = ''
for i in range(0, len(seq), 3):
    codon = seq[i:i+3]
    aa = codon_to_amino_acid[codon]

    if aa == '*':
        break
    protein = protein + aa

print(protein)

```

بذارین امتحانش کنم... چه خوب، درست شد:

MA

استاد: دقیقاً! حالا می‌توانید اگر علاقه داشتید، مثلاً یک رشته DNA (ژن) واقعی را از سایتی مثل uniprot پیدا کنید و برنامه شما به راحتی آن را ترجمه کند.

حالا برویم به سراغ سؤال اصلی مان! ما می‌خواهیم برنامه‌ای که نوشتیم، یک تابع باشد. یک دستور که هر موقع خواستیم بتوانیم راحت آن را صدا بزنیم تا رشته ما را ترجمه کند. اول از همه نام این تابع را چه بگذاریم؟

دانشجو: چون داره ترجمه رو انجام می‌ده، به نظرم translate مناسب باشه.

استاد: بسیار عالی. پس کافی است بنویسیم:

```

def translate():
    seq = 'ATGGCTTAG'

    protein = ''
    for i in range(0, len(seq), 3):

```

```
codon = seq[i:i+3]
aa = codon_to_amino_acid[codon]

if aa == '*':
    break
protein = protein + aa

return protein
```

و چقدر بهتر که seq را هم به عنوان ورودی از مخاطب بگیریم:

```
def translate(seq):
    protein = ''
    for i in range(0, len(seq), 3):
        codon = seq[i:i+3]
        aa = codon_to_amino_acid[codon]

        if aa == '*':
            break
        protein = protein + aa

    return protein
```

و حال دستور ما این قابلیت را دارد که به راحتی آن را صدا بزنیم؛ مثلاً:

```
result = translate('ATCGTAACC')
print(result)
```

و می بینید که به راحتی برای ما ترجمه کرد و نتیجه را نمایش داد:

IVT

من توصیه می کنم که دیکشنری حاوی کدون ها و آمینواسیدها هم داخل خود

تابع باشد:

```
def translate(seq):
    codon_to_amino_acid = {
        'ATA': 'I', 'ATC': 'I', 'ATT': 'I', 'ATG': 'M',
        'ACA': 'T', 'ACC': 'T', 'ACG': 'T', 'ACT': 'T',
        'AAC': 'N', 'AAT': 'N', 'AAA': 'K', 'AAG': 'K',
        'ATA': 'I', 'ATC': 'I', 'ATT': 'I', 'ATG': 'M',
        'CTA': 'L', 'CTC': 'L', 'CTG': 'L', 'CTT': 'L',
        'CCA': 'P', 'CCC': 'P', 'CCG': 'P', 'CCT': 'P',
        'CAC': 'H', 'CAT': 'H', 'CAA': 'Q', 'CAG': 'Q',
        'CGA': 'R', 'CGC': 'R', 'CGG': 'R', 'CGT': 'R',
        'GTA': 'V', 'GTC': 'V', 'GTG': 'V', 'GTT': 'V',
        'GCA': 'A', 'GCC': 'A', 'GCG': 'A', 'GCT': 'A',
        'GAC': 'D', 'GAT': 'D', 'GAA': 'E', 'GAG': 'E',
        'GGA': 'G', 'GGC': 'G', 'GGG': 'G', 'GGT': 'G',
        'TAA': '*', 'TAG': '*', 'TGA': '*',
        'TAC': 'Y', 'TAT': 'Y', 'TAA': '*', 'TAG': '*',
        'TCA': 'S', 'TCC': 'S', 'TCG': 'S', 'TCT': 'S',
        'TTC': 'F', 'TTT': 'F', 'TTA': 'L', 'TTG': 'L',
        'TAC': 'Y', 'TAT': 'Y', 'TAA': '*', 'TAG': '*',
        'TGC': 'C', 'TGT': 'C', 'TGA': '*', 'TGG': 'W'
    }

    protein = ''
    for i in range(0, len(seq), 3):
        codon = seq[i:i+3]
        aa = codon_to_amino_acid[codon]

        if aa == '*':
```



```

        break
    protein = protein + aa

```

```

    return protein

```

البته همان‌طور که دیدید، مقداری کد ما را شلوغ کرد. اگر دوست داشتیم که اصولی‌تر کنیم، بهتر است دیکشنری خود را به شکل فایل (فایل متنی یا فایل json که راحت بتوانیم آن را بخوانیم) ذخیره کنیم و بعد در داخل تابع translate آن فایل را بخوانیم، مثلاً:

```

import json
with open('data.json', 'r') as f:
    codon_to_amino_acid = json.load(f)

```

که می‌بینید من این قسمت را داخل تابعمان قرار می‌دهم:

```

import json

def translate(seq):
    with open('data.json', 'r') as f:
        codon_to_amino_acid = json.load(f)

    protein = ''
    for i in range(0, len(seq), 3):
        codon = seq[i:i+3]
        aa = codon_to_amino_acid[codon]

        if aa == '*':
            break
    protein = protein + aa

```

```
return protein
```

یا اگر بخواهیم دیگر خیلی اصولی‌تر کار کنیم (بیشتر بدانید)، می‌توانیم مثلاً یک تابع به نام `read_file` تعریف کنیم که کارش خواندن فایل کدون‌ها و تحویل دادن دیکشنری مدنظر ما باشد:

```
def read_file():
    with open('data.json', 'r') as f:
        codon_to_amino_acid = json.load(f)

    return codon_to_amino_acid
```

و بعد می‌توانیم در داخل تابع `translate` از این تابع `read_file` استفاده کنیم:

```
codon_to_amino_acid = read_file()
```

به عبارتی برنامهٔ کامل ما شد:

```
import json

def read_file():
    with open('data.json', 'r') as f:
        codon_to_amino_acid = json.load(f)

    return codon_to_amino_acid

def translate(seq):
    codon_to_amino_acid = read_file()

    protein = ''
    for i in range(0, len(seq), 3):
```

```
codon = seq[i:i+3]
aa = codon_to_amino_acid[codon]

if aa == '*':
    break
protein = protein + aa

return protein
```

دانشجو: استاد همون چیزی که اول کار نوشتیم راحت تر نبود؟

استاد: به آن شکل هم بنویسید برنامه شما اشتباه نیست، اما هر چه بتوانید برنامه خود را قسمت قسمت به صورت تابع بنویسید (حتی قسمت های ساده و کوچک مثل همین قسمت خواندن فایل)، برنامه شما برای مخاطب (و بعداً خودتان) خواناتر خواهد شد و به این شکل راحت تر می توانید برنامه خود را بعداً تغییر بدهید. اگر کد افراد حرفه ای را ببینید، مشاهده می کنید که توابعی که نوشته اند، اکثراً چند خط بیشتر نیستند، اما تعداد زیادی تابع نوشته اند و مدام ارجاع داده اند.

خود من به شخصه قبلاً که با رویکرد تابع محور (یا اصطلاحاً رویکرد ماژولار) آشنا نبودم، همین جور پشت سر هم برنامه خود را می نوشتم و برنامه های من یک برنامه شلوغ پر از دندانه می شد که الآن وحشت داشتم نگاهشان کنم؛ چون بسیار سخت می توانستم متوجه بشوم که چه می کنند و سخت می توانستم تغییرشان دهم. از بعد از آشنا شدن با رویکرد تابع محور، سعی می کنم برنامه خود را تکه تکه، هر قسمت که کار مشخصی می کند را به صورت یک تابع بنویسم (مثلاً تابع خواندن فایل و ...) و در انتها کافی است که این تابع ها را یکی یکی صدا بزنم و استفاده کنم. به این شکل برنامه های من بسیار خواناتر از قبل شده است.

خب اگر آماده هستید، برویم به سراغ بحث شیرین تمرین ها!

تمرین‌های جلسه هشتم

در تمرین هر درس، ممکن است که یک نکته جدید هم آموزش داده شود؛ پس حتماً توصیه می‌کنم آن‌ها را ببینید و انجامشان دهید. تمرین‌ها و به‌خصوص خطاهایی که دریافت می‌کنید و آزمون و خطاها (و همچنین سرچ‌هایی که در صورت نیاز انجام می‌دهید) بسیار بسیار به روند یادگیری برنامه‌نویسی شما کمک می‌کند. پاسخ نمونه برای تمرین‌ها نیز پس از آن‌ها قرار داده شده است.

تمرین‌ها

۱- یکی از برنامه‌های قبلی خود را به شکل تابع بنویسید.

۲- سایتی به نام Disgenet وجود دارد که می‌توانیم یک بیماری (مثلاً افسردگی یا همان Major depressive disorder) را در آن جستجو کنیم و این سایت لیست ژن‌های پیداشده درگیر در آن بیماری را به ما نشان می‌دهد و قابلیت خروجی دادن فایل اکسل (یا فایل متنی با فرمت tsv) را دارد. یک بیماری را در آن جستجو کرده و فایل خروجی را دانلود کنید. حال با پایتون، تابعی بنویسید که فایل حاصل از سایت Disgenet را خوانده و به‌عنوان خروجی لیست نام ژن‌ها و همچنین تعداد ژن‌ها را برگرداند. (این سایت برای ثبت‌نام به ایمیل دانشگاهی نیاز دارد. در صورتی که دانشجو هستید، می‌توانید با تماس با مرکز IT دانشگاه‌تان، ایمیل دانشگاهی خود را دریافت کنید).

۳- تمرین اختیاری: برای پیدا کردن لیست تمامی فایل‌های موردنظر در یک فولدر (برای مثال تمامی فایل‌های pdf)، کتابخانه‌ای به نام glob وجود دارد. چگونه می‌توان این کار را انجام داد؟

پاسخ تمرین‌های جلسه هشتم

اکیداً توصیه می‌کنم که اول خودتان تمرین‌ها را حل کنید و سپس به پاسخ‌ها مراجعه کنید. درضمن این پاسخ‌ها تنها یکی از حالت‌های ممکن نوشتن برنامه هستند و شما ممکن است به شیوه‌ای دیگر برنامه را نوشته باشید و کاملاً صحیح هم باشد. همچنین برای مشاهده توضیحات شفاهی مربوط به این تمرین‌ها، ابتدای ویدئوی جلسه بعد را ببینید (لینک ویدئو در صفحه اول کتاب گذاشته شده است):

« پاسخ تمرین اول:

برای مثال برنامه‌ای که تمام مقسوم‌علیه‌های یک عدد را به صورت لیست به ما برمی‌گرداند. برنامه ما بدون تابع:

```
num = 10

result = []
for i in range(1, num):
    if num%i==0:
        result.append(i)

print(result)
```

برنامه ما به شکل تابع که به عنوان ورودی، عدد مورد نظر را بگیرد و به عنوان خروجی تمامی مقسوم‌علیه‌های آن را بدهد:

```
def calculate_divisors(num):
    result = []
    for i in range(1, num):
        if num%i==0:
            result.append(i)
```

```
return result
```

```
print(calculate_divisors(10))
```

خروجی در هر دو صورت:

```
[1, 2, 5]
```

« پاسخ تمرین دوم:

کافی است همانند تمرین مربوط به فایل‌ها، فایل متنی را خط به خط بخوانیم و نام ژن‌ها را در یک لیست بریزیم (البته کماکان می‌گوییم که با یادگیری کتابخانه، خواندن فایل‌های جدولی و اکسل و ... بسیار راحت‌تر خواهد بود):

```
def read_file():
```

```
    file_path = r'Chronic constipation.csv'
    genes=[]
    with open (file_path, 'r') as file:
        lines = file.readlines()
        for line in lines[1:]:
            gene_full_name = line.split(',')
            [5]
            genes.append(gene_full_name)

    return genes, len(genes)
```

```
genes, count = read_file()
print(count)
```

« پاسخ تمرین سوم:

برای این که لیست فایل‌های موجود در یک فولدر را بخواهیم، دستور `os.listdir` بسیار مناسب است. مثلاً می‌خواهیم بدانیم در پوشه زیر چه فایل‌هایی وجود دارد:

```
import os
path = r"E:\My Publications\Python Book"
files = os.listdir(path)
print(files)
```

که می‌بینید لیستی حاوی تمام فایل‌های موجود در این پوشه را داریم:

```
['08 P101 Func.mp4 - Shortcut.lnk', 'Python 01.docx',
'Python 02.docx', 'Python 03.docx', 'Python 04.docx',
'Python 05.docx', 'Python 06.docx', 'Python 07.docx',
'Python 08.docx', 'Python Course 1402 - Shortcut.lnk',
'test.ipynb', '~$thon 08.docx']
```

که می‌توانیم با حلقه `for` روی این لیست حرکت کنیم و مثلاً هر فایل را بخوانیم و کار خاصی روی آن انجام بدهیم (مثلاً آن را به نامی دیگر `rename` کنیم یا ...); اما بسیاری از مواقع ما می‌خواهیم مثلاً فقط لیست فایل‌های با پسوند `.docx` را داشته باشیم. در این مواقع کتابخانه `glob` بسیار کاربردی است. دستورات را ببینید و نتیجه را با قبل مقایسه کنید:

```
from glob import glob
path = r"E:\My Publications\Python Book\*.docx"
files = glob(path)
print(files)
```

نتیجه:

```
['E:\\My Publications\\Python Book\\Python 01.docx',
'E:\\My Publications\\Python Book\\Python 02.docx',
```

```
'E:\\My Publications\\Python Book\\Python 03.docx',
'E:\\My Publications\\Python Book\\Python 04.docx',
'E:\\My Publications\\Python Book\\Python 05.docx',
'E:\\My Publications\\Python Book\\Python 06.docx',
'E:\\My Publications\\Python Book\\Python 07.docx',
'E:\\My Publications\\Python Book\\Python 08.docx']
```

می‌بینید که فقط فایل‌های ورد در این لیست به‌صورت آدرس مطلق هستند که کار را برای پردازش‌های بعدی بسیار راحت‌تر می‌کنند. برای مثال می‌توانید یک حلقه `for` بنویسید و با کتابخانهٔ مربوط به فایل‌های `word` در پایتون مثلاً به ابتدای تمامی این فایل‌ها موردی را اضافه کنید که حتماً توصیه می‌کنم اگر علاقه‌مند بودید جستجو کنید و یکی از این کارها را انجام دهید. برای مثال در یک پروژه مشغول به کار بودم که نیاز بود به نام تمامی فایل‌ها، یک عدد خاص هم اضافه شود؛ مثلاً فایل `test.pdf` به فایل `test_123.pdf` تغییر نام پیدا کند. خوب برای یک فایل کار سختی نیست، اما وقتی هزاران فایل دارید این کار بسیار حوصله‌سر بر است. در ده دقیقه یک برنامهٔ پایتون چند خطی نوشتم که این کار را ظرف چند ثانیه انجام داد؛ یا برای مثال من موقع کار، موسیقی گوش می‌کنم و نکته این بود که این موسیقی‌ها هرکدام بلندی صدای خود را داشتند. با کمک چت‌جی‌پی‌تی و جستجو، برنامه‌ای را با پایتون نوشتم که کافی است آدرس فولدر حاوی موسیقی را به آن بدهیم و خود پایتون تک‌تک فایل‌ها را به‌نوبت خوانده، آن را به بلندای استاندارد تغییر داده و فایل را ذخیره می‌کند و به همین راحتی من می‌توانم از موسیقی بیشتر لذت ببرم. (اگر تمایل داشتید می‌توانید این کار را در صفحهٔ گیت‌هاب من به آدرس زیر ببینید و دانلود کنید).

<https://github.com/pporbaha/SoundBalancer-Your-Music-Volumes-at-One-Level>

• فصل نهم •

set، حلقة while و مدیریت خطاها

سلام و درود بر شما! امیدوارم حالتان عالی باشد. ما تا این فصل مطالب پایه‌ای مهمی را در مورد برنامه‌نویسی پایتون باهم یاد گرفتیم و تمرین کردیم. ساختار داده‌های مهم و کاربردی، لیست‌ها، دیکشنری‌ها و توپل‌ها را دیدیم. یک مورد دیگر مانده است به نام set که از قبلی‌ها مطالب کمتری دارد اما به شما قابلیت‌های بسیار خوبی را می‌دهد که باهم این مورد را در این جلسه کار خواهیم کرد. از طرفی با حلقه for نیز آشنا شدیم؛ اما یک حلقه دیگر به نام حلقه while نیز در پایتون وجود دارد که در بعضی موارد بسیار به کار شما می‌آید. اگر آماده هستید، برویم و شروع کنیم!

setها چیستند و به چه دردی می‌خورند؟

مثل همیشه با یک مثال شروع کنیم. فرض کنید لیستی از اسامی داروها در پایتون داریم:

```
drugs = ['acetaminophen', 'diclofenac', 'ibuprofen']
```

همان‌طور که تا الآن دیدیم با لیست‌ها کارهای بسیاری می‌توان انجام داد و بسیار هم مفید بوده‌اند؛ اما یک نکته وجود داشت. در لیست‌ها می‌توانستیم عضو تکراری هم داشته باشیم. برای مثال اگر لیست drugs را به این شکل تعریف کنیم:

```
drugs = ['acetaminophen', 'diclofenac', 'ibuprofen',  
         'acetaminophen', 'diclofenac']
```

و بعد این لیست را پرینت کنم (یا در کنسول مقدار آن را ببینم):

```
print(drugs)
```

می‌بینیم که نام‌های تکراری در آن وجود دارند:

```
['acetaminophen', 'diclofenac', 'ibuprofen', 'acetami-  
nophen', 'diclofenac']
```

اگر همین چیزی را که داشتیم، به جای علامت [] که در پایتون نشان‌دهنده لیست

بود، با علامت {} می‌ساختم، آن موقع چیزی به نام set داشتیم:

```
drugs = {'acetaminophen', 'diclofenac', 'ibuprofen',  
         'acetaminophen', 'diclofenac'}
```

دانشجو: استاد دیکشنری رو هم با همین علامت {} تعریف نمی‌کردیم؟ پس چطور

بفهمیم set هست یا دیکشنری؟

استاد: بسیار سؤال خوبی است. در دیکشنری ما key و value را داریم که با علامت

«:» مشخص شده‌اند؛ اما در اینجا به این شکل نیست و فقط تعدادی چیز داریم.

حال اگر این set را پرینت کنیم، با موضوع جالبی روبرو می‌شویم:

```
print(drugs)
```

خروجی:

```
{'acetaminophen', 'diclofenac', 'ibuprofen'}
```

دانشجو: چه جالب! انگار عضوهای تکراری فقط یک‌بار نمایش داده شدن.

استاد: دقیقاً! set به معنی مجموعه است و اگر خاطرمาน باشد، مجموعه‌ها

نمی‌توانستند اعضای تکراری داشته باشند؛ به عبارت دیگر پایتون خودش اعضای

تکراری را حذف می‌کند. این خاصیت set ها بسیار مفید است و به راحتی می‌توانیم ببینیم چند عضو منحصر به فرد داریم:

```
print(len(drugs))
```

که می‌بینیم به ما خروجی ۳ را داد، زیرا فقط ۳ داروی منحصر به فرد داریم.

دانشجو: استاد می‌شه لیستی که داشتیم رو به set تبدیل کنیم؟

استاد: بله چرا که نشه! کافیه از دستور set() استفاده کنیم:

```
drugs = ['acetaminophen', 'diclofenac', 'ibuprofen',
         'acetaminophen', 'diclofenac']
new = set(drugs)
print(new)
```

خروجی:

```
{'acetaminophen', 'diclofenac', 'ibuprofen'}
```

و می‌بینیم که به زیبایی یک set داریم که می‌توانیم با آن کار کنیم. فقط یک نکته را حواسمان باشد؛ این دستور را چک کنید:

```
print(new[0])
```

دانشجو: منطقاً باید عضو اول set ما رو نشان بده یعنی 'acetaminophen'. بذارین امتحان کنم... عه چرا خطا داد؟

TypeError

Traceback (most recent call last)

Cell In[5], line 3

```
1 drugs = ['acetaminophen', 'diclofenac',
           'ibuprofen', 'acetaminophen', 'diclofenac']
```

```
2 new = set(drugs)
----> 3 print(new[0])
TypeError: 'set' object is not subscriptable
```

استاد: دقیقاً! setها دارای ترتیب نیستند؛ به عبارتی اندیس در آنها بی معنی است (همان طور که در متن خطا می بینید، اصطلاحاً setها subscriptable نیستند). این بدین معنی است که روی ترتیب اعضا در setها نمی توانیم حساب باز کنیم؛ پس اگر مثلاً با حلقه for خواستیم که روی این set حرکت کنیم، ترتیب اعضا به احتمال زیاد به همین ترتیب نیست؛ اما همان طور که گفتیم یک مزیت بسیار بزرگ دارند؛ فقط دارای اعضای منحصر به فرد هستند.

خب حالا به نظر تان چطور می توان یک ست خالی را ایجاد کنیم؟
دانشجو: کافیه بنویسیم؟

```
s = {}
```

استاد: حالا type آن را چک کنید.

دانشجو: چشم استاد. بذارین در داخل کنسول ببینم:

```
type(s)
```

انگار دیکشنریه!

استاد: دقیقاً! اگر بخواهیم یک set خالی تعریف کنیم، باید از همان دستور set() استفاده کنیم:

```
s = set()
```

و چون داخل دستور set() چیزی ننوشتیم، یک set خالی تعریف می شود.

متمدهای set برای مثال داروها

همانند لیست‌ها و دیکشنری‌ها، set‌ها نیز متمدهای مفیدی دارند که با آن‌ها آشنا خواهیم شد. برای مثال می‌توان از متد add برای اضافه کردن یک عضو به مجموعه استفاده کرد:

```
s = {'acetaminophen', 'diclofenac', 'ibuprofen'}
s.add('aspirin')
```

و می‌بینید که به ست خالی ما، داروی aspirin اضافه شد:

```
{'acetaminophen', 'aspirin', 'diclofenac', 'ibuprofen'}
```

و می‌توانیم از متد remove برای حذف کردن یک عضو مشخص از ست خودمان استفاده کنیم؛ مثلاً:

```
s.remove('ibuprofen')
```

و می‌بینیم که اگر مقدار آن را ببینیم، این دارو از ست ما حذف شده است:

```
{'acetaminophen', 'aspirin', 'diclofenac'}
```

فرض کنید یک ست به نام s1 و ست دیگری به نام s2 داریم:

```
s1 = {'acetaminophen', 'diclofenac', 'ibuprofen'}
s2 = {'acetaminophen', 'ketorolac', 'opmeprazole'}
```

همان‌طور که می‌بینید در این دو مجموعه، برخی داروها (مثلاً استامینوفن) مشترک هستند و برخی از داروها فقط در یکی از این ست‌ها هستند. حالا فرض کنید ست‌ها شامل هزاران دارو بود؛ چگونه می‌توانیم خیلی سریع ببینیم که کدام دارو در بین این دو ست مشترک هستند؟ به عبارتی اشتراک این دو مجموعه چیست؟ کافی است از متد intersection (به معنای اشتراک) استفاده کنیم:

```
s1.intersection(s2)
```


و می بینید که این دستور به راحتی به ما یک ست از داروهای مشترک را داد (که در اینجا فقط داروی استامینوفن مشترک است):

```
{ 'acetaminophen' }
```

یا شاید بخواهیم ببینیم روی هم رفته چه تعداد دارو داریم؛ به عبارتی اجتماع این دو مجموعه را بخواهیم. کافی است از متد union استفاده کنیم:

```
s1.union(s2)
```

خروجی:

```
{ 'acetaminophen', 'diclofenac', 'ibuprofen', 'ketorolac',  
'opmeprazole' }
```

می توانیم خروجی این دستور را در یک متغیر تعریف کنیم و بعد با len ببینیم که چه تعداد دارو در کل وجود داشته است:

```
all_drugs = s1.union(s2)  
print(len(all_drugs))
```

خروجی: ۵ عدد

یا می توانیم ببینیم که مثلاً کدام داروها در مجموعه اول هستند که در مجموعه دومی وجود ندارند، می توانیم از متد difference یا همان تفاوت استفاده کنیم:

```
s1.difference(s2)
```

خروجی:

```
{ 'diclofenac', 'ibuprofen' }
```

و می بینیم که دو داروی دیکلوفناک و ایبوپروفن در مجموعه اولی هستند، اما در دومی نیستند.

دانشجو: چقدر جالب بود! استاد این موضوع در رشته ما چه کاربردهایی می تونه

داشته باشه؟

استاد: مثلاً می‌خواهیم ببینیم که بین افسردگی و بیماری کبد چرب رابطه‌ای وجود دارد یا نه، می‌توانیم از سایتی مثلاً *disgenet* بیماری افسردگی مازور را جستجو کنیم و این سایت تمامی ژن‌های کشف‌شده در رابطه با این بیماری را به‌صورت فایل به ما می‌دهد. سپس می‌توانیم همین کار را برای بیماری کبد چرب نیز انجام دهیم و مجموعه ژن‌های کشف‌شده برای این بیماری را هم دریافت کنیم. حال می‌توانیم اشتراک این دو مجموعه را ببینیم و ببینیم چند ژن مشترک در هر دو بیماری وجود دارند. هرچه این ژن‌ها بیشتر باشند، این فرضیه که احتمال دارد این دو بیماری از جنبه‌هایی به هم مرتبط باشند را تقویت می‌کند.

حالا نکته جالب ماجرا اینجاست که سایت‌هایی مثلاً *DGIdb* (مخفف *Drug Gene Interaction Database*) به شما نشان می‌دهد که چه داروهایی بر روی این ژن‌ها می‌توانند مؤثر باشند. حالا شما برای مثال می‌دانید که داروی *x* که الان بر روی کبد چرب در پزشکی استفاده می‌شود، چون بر روی ژن‌های مشترک با بیماری کبد چرب و افسردگی مازور تأثیر دارد، پس شاید بتواند بر روی افسردگی مازور هم مؤثر باشد. ما خودمان مقاله مشابهی را در رابطه با افسردگی کار کردیم و دیدیم مثلاً *omeprazole* که یک دارو برای اسید معده است می‌تواند روی افسردگی هم مؤثر باشد! و جالب بود که مقالات دیگر نیز این فرض را تأیید کرده بودند. البته از الگوریتم‌های دیگر سیستم بیولوژی هم در این کار استفاده شد. این همین است که حتی شما می‌توانید این یافته را به شکل یک فرضیه (*hypothesis*) در ژورنالی مربوط به این حیطه (مثلاً *Medical hypothesis*) که فرضیه‌ها را چاپ می‌کند منتشر کنید.

جالب است که بدانید این ژورنال ایمپکت فاکتور حدود ۴.۶ دارد زیرا بسیاری از

ایده‌های مقالات از همین فرضیات می‌آیند و این مقالات می‌توانند توسط دیگر مقالات مورد ارجاع قرار بگیرند.

دانشجو: چقدر جالب!!

استاد: خب دوستان! این هم از set ها. دیدید چقدر کوتاه بود؟ پس از این به سراغ یکی از حلقه‌های دیگر پایتون برویم؛ حلقهٔ while.

حلقهٔ while چیست؟

حلقه‌ها برای تکرار کردن قسمتی از کد به کار می‌روند. ما تا الآن تنها حلقه‌ای که یاد گرفتیم، حلقهٔ for بود. بیاییم باهم یک بار ویژگی‌های مهم این حلقه را مرور کنیم. به عبارتی این تکرار کردن به وسیلهٔ حلقهٔ for چه مزایا و معایبی داشت؟

دانشجو: باهاش می‌تونستیم به اعضای یه لیست یا دیکشنری یا ست یا... به ترتیب دسترسی داشته باشیم.

استاد: دقیقاً! پس برای حرکت کردن روی لیست‌ها یا ... بسیار خوب بود؛ منتها حلقهٔ for یک ویژگی اساسی دارد. این حلقه برای کارهایی مناسب است که ما تعداد تکرار را می‌دانیم و می‌دانیم که کد قرار است چند بار تکرار شود.

دانشجو: استاد میشه یه بار دیگه این رو توضیح بدید؟

استاد: بله حتماً. با یک مثال پیش می‌رویم. مثلاً کد زیر که با حلقهٔ for نوشته شده را ببینید:

```
data = []
for i in range(5):
    name = input('please input your name:')
```

`data.append(name)`

این کد ۵ بار نام فرد را پرسیده و به لیست `data` اضافه می‌کند. اما سؤال اینجاست! شاید مخاطب بخواهد ۳ نام را وارد کند. یا بخواهد ۱۰ نام را اضافه کند. به عبارتی ما شاید ندانیم که باید این کد را چند بار تکرار کنیم. شاید بهتر باشد به مخاطب بگوییم هر وقت کارت تمام شد، یک حرف `q` را وارد کن که به معنی `quit` باشد و حلقه ما بشکند.

به عبارتی ما خیلی از مواقع دقیقاً نمی‌دانیم که قرار است آن کار چند بار انجام شود. ما در زندگی روزمره می‌گوییم، **مادامی این کار را انجام بده که فلان شرط برقرار است**. مادام به انگلیسی می‌شود همان حلقه `while`؛ یعنی در حلقه `while` برعکس حلقه `for` ما تعداد هر بار را نمی‌دانیم بلکه یک شرط می‌دهیم و می‌گوییم تا وقتی که این شرط برقرار است، این کار را انجام بده و اگر این شرط نقض شد، حلقه را بشکن. پس یک سؤال! کدام حلقه می‌تواند یک کار را تا بی‌نهایت برای ما تکرار کند؟

دانشجو: قاعدتاً حلقه `while`!

استاد: بیاییم با همان حلقه `for` یک برنامه بنویسیم که از عدد ۰ تا ۴ را برآیمان چاپ کند.

دانشجو: بله حتماً استاد! مثلاً می‌تونیم به این شکل بنویسیم.

```
for i in range(5):
    print(i)
```

استاد: بسیار عالی! خوب بیایید باهم همین را با حلقه `while` بنویسیم. می‌خواهیم از صفر شروع کنیم، پس می‌توانیم یک متغیر تعریف کنیم و مقدار آن را ۰ بگذاریم:

```
i = 0
```

همان طور که گفتیم، جلوی حلقه while یک شرط قرار می گیرد.

```
while i <= 4:  
    print(i)
```

و بعدش...

دانشجو: عجب چیزی شد! بذارین چکش کنم! ... ای وای استاد! کامپیوترم داره فقط می نویسه:

```
0  
0  
0  
0  
0
```

چیکار کنم؟

استاد: می توانی با زدن کلید C + Ctrl برنامه را متوقف کنی؛ اما بیا خودمان را جای کامپیوتر بگذاریم و ببینیم چرا فقط دارد ۰ چاپ می کند.

کامپیوتر اول به این خط می رسد:

```
i = 0
```

و مقدار i برابر با ۰ گذاشته می شود؛ سپس به این خط می رسد.

```
while i <= 4:
```

آیا شرط $i \leq 4$ برقرار است؟

دانشجو: بله استاد! مقدار i برابر با ۰ است و ۰ هم از ۴ کوچک تره.

استاد: پس وارد حلقه می شود و دستورات داخل حلقه را اجرا می کند:

```
print(i)
```

حال دوباره به ابتدای حلقه برمی‌گردد و مجدد شرط را چک می‌کند:

```
while i <= 4:
```

و چون مقدار `i` هنوز ۰ است، دستور داخل حلقه اجرا می‌شود:

```
print(i)
```

و دوباره به ابتدای حلقه برمی‌گردد و شرط را چک می‌کند:

```
while i <= 4:
```

دانشجو: آهان! ما مقدار `i` رو تغییر ندادیم و مقدار `i` همیشه ۰ است و به‌خاطر همین، هیچ‌وقت این شرط نقض نمی‌شه که حلقه شکسته بشه. برای همین کامپیوتر من در یه حلقه بی‌نهایت گیر کرده!

استاد: دقیقاً! ما می‌توانستیم پس از پرینت کردن `i`، یکی به مقدار `i` اضافه کنیم:

```
i = 0
```

```
while i <= 4:
```

```
    print(i)
```

```
    i += 1
```

و می‌بینیم که این برنامه به‌صورت درست کار کرده و از ۰ تا ۴ را چاپ می‌کند. در بار آخر حلقه، مقدار `i` برابر با ۵ می‌شود. وقتی کامپیوتر به ابتدای حلقه برمی‌گردد:

```
while i <= 4:
```

دیگر این شرط برقرار نیست و به همین دلیل وارد دستورات داخل حلقه نمی‌شود.

همان‌طور که دیدید این برنامه از حلقه `for` کمی پایه‌ای‌تر است و به همین دلیل است که گفته می‌شود که حلقه `while` از حلقه `for` جامع‌تر است، اما همان‌طور که تا الآن استفاده می‌کردیم برای کارهایی که میزان تکرار ما مشخص است، حلقه `for` انتخاب ساده‌تری است.

دانشجو: استاد نمی‌شد از break استفاده کنیم تا حلقه while شکسته بشه؟

استاد: درود بر تو! راه دیگری که از حلقه while استفاده می‌شود، این است که یک شرط همیشه درست را جلوی while می‌گذارند:

```
while 0==0:
```

چون همیشه ۰ برابر با ۰ است این شرط همیشه برقرار است. البته مرسوم است که در پایتون می‌نویسند True:

```
while True:
```

و بعد در خود حلقه بگوییم: اگر i بزرگ‌تر از ۴ شد، حلقه را بشکن:

```
i = 0
```

```
while True:
```

```
    print(i)
```

```
    i += 1
```

```
    if i>4:
```

```
        break
```

دانشجو: این از قبلی هم پیچیده‌تر شد!

استاد: دقیقاً! این روش برای کجا بهتر است؟ مثلاً همان مثالی که زدیم و می‌خواهیم تا مادامی که مخاطب کلمه q را تایپ نکرده، کامپیوتر این حلقه را تکرار کند. این برنامه قبلی ما با for بود:

```
data = []
```

```
for i in range(5):
```

```
    name = input('please input your name:')
```

```
    data.append(name)
```

```
print(data)
```

با while چه کنیم؟ دقیقاً عین قبلی منتها از while True استفاده می‌کنیم:

```
data = []
while True:
    name = input('please input your name:')
    if name == 'q':
        break
    data.append(name)
print(data)
```

بیاییم باهم این برنامه را trace کنیم! کامپیوتر ابتدا یک لیست خالی به نام data ایجاد می‌کند:

```
data = []
```

حال به این خط می‌رسیم:

```
while True:
```

این شرط True است، پس برقرار است و وارد حلقه می‌شویم:

```
name = input('please input your name:')
```

مخاطب name را مثلاً Pedram وارد می‌کند. حال در خط بعدی این شرط چک می‌شود:

```
if name == 'q':
```

چون name برابر با q نیست، این دستور اجرا نمی‌شود و حلقه ادامه دارد. سپس

```
data.append(name)
```

به data اضافه می‌شود. دوباره کامپیوتر به ابتدای حلقه برمی‌گردد:

```
while True:
```

این شرط True و برقرار است؛ پس دوباره وارد حلقه می‌شویم و کامپیوتر از کاربر

نام جدید را می پرسد:

```
name = input('please input your name:')
```

حال فرض کنید مخاطب خسته شده و می خواهد نام برنامه دیگری را از او نپرسد و q را وارد کند.

حال به خط بعدی می رسیم:

```
if name == 'q':
    break
```

چون name برابر با q است، این شرط برقرار است و دستور زیرمجموعه شرط یعنی break اجرا شده، حلقه تمام می شود و ادامه برنامه اجرا می شود:

```
print(data)
```

و در نهایت data پرینت می شود:

```
['pedram']
```

یا مثلاً فرض کنید برنامه ای که به عنوان تمرین جلسه if نوشتیم، که اگر کاربر مثلاً عدد ۱ را وارد کرد، برنامه یک کار خاص انجام دهد یا اگر عدد ۲ را وارد کرد، کار دیگری انجام دهد و اگر عددی غیر از این دو را وارد کرد، پیغام خطا ظاهر شود؛ اما در آنجا اگر کاربر عدد اشتباهی وارد می کرد، برنامه تمام می شد. ما می خواهیم تا وقتی که عدد درست وارد نشده، برنامه به کاربر گیر دهد تا وقتی که کاربر عدد درست را وارد کند. در اینجا نیز می توانیم از while استفاده کنیم:

```
choice = input('Please input 1 for male or 2 for female:')
while choice not in ['1', '2']:
    choice = input('You entered wrong number. please enter 1 or 2:')
```

```
print(choice)
```

همان طور که می بینید کامپیوتر ابتدا از مخاطب choice را می پرسد. سپس به این خط می رسد:

```
while choice not in ['1', '2']:
```

اگر کاربر عددی غیر از ۱ یا ۲ را وارد کرده باشد، پس شرط `choice not in ['1', '2']` درست است و وارد حلقه می شویم. در حلقه choice دوباره از مخاطب پرسیده می شود:

```
choice = input('You entered wrong number. please enter 1 or 2:')
```

و دوباره به ابتدای حلقه برمی گردیم:

```
while choice not in ['1', '2']:
```

اگر باز کاربرد choice را درست وارد نکرده باشد، این آن قدر موضوع تکرار می شود که بالاخره choice درست وارد شود.

دانشجو: استاد همین برنامه رو با همون روش break هم می تونیم بنویسیم؟

استاد: بله؛ چرا که نه! یک روش می تواند این باشد.

```
choice = input('Please input 1 for male or 2 for female:')
```

```
while True:
```

```
    if choice not in ['1', '2']:
```

```
        choice = input('You entered wrong number. please enter 1 or 2:')
```

```
    else:
```

```
        break
```

```
print(choice)
```

و می بینید که در حلقه بی نهایت هستیم. در حلقه بررسی می شود که اگر choice

عدد ۱ یا ۲ نبود به کاربر می‌گوید که عدد نادرستی وارد کرده و عدد جدیدی از او می‌گیرد. در غیر این صورت یعنی در صورتی که کاربر عدد درستی وارد کرده باشد، حلقه شکسته می‌شود. شما به شکل‌های دیگری هم می‌توانید این برنامه را بنویسید، فقط مهم این بود که با مفهوم حلقهٔ `while` آشنا شوید.

مدیریت کردن خطاها با `try except`

تابه حال هرگاه که به خطا برخورد می‌کردیم، برنامه ما متوقف می‌شد و خطا را به ما نشان می‌داد. زمان‌هایی است که می‌خواهیم در صورت بروز خطا، برنامه ما ادامه پیدا کند و برای مثال اگر خطا داد، کار دیگری انجام دهد. چطور می‌توان در پایتون این کار را انجام داد؟ هیچ چیز بهتر از یک مثال نیست.

برنامه زیر را در نظر بگیرید:

```
print('Hello')
print(1/0)
print('Bye')
```

در این حالت ابتدا برنامه ما، `hello` را چاپ می‌کند:

Hello

اما در خط بعد، چون در کامپیوتر تقسیم بر صفر تعریف نشده است، با خطای زیر روبرو می‌شویم:

```
-----
-----
ZeroDivisionError      Traceback (most recent call last)
Cell In[4], line 2
      1 print('Hello')
----> 2 print(1/0)
```

ZeroDivisionError: division by zero

و برنامه ما متوقف شده و خط بعدی اجرا نمی‌شود. در بسیاری از زمان‌ها این خطاها خوب هستند و به من برنامه‌نویس می‌فهمانند که مشکلی وجود دارد و ما آن را برطرف می‌کنیم؛ اما حالتی را تصور کنید که برنامه‌ای را از شب تا صبح گذاشتید تا برای مثال نام داروهایی که داشته‌اید را تک‌به‌تک در یک سایت جستجو کرده و نتیجه را در یک فایل ذخیره کند. قطعاً در این صورت تمایل ندارید صبح از خواب بیدار شوید و به خاطر این که اطلاعات یک دارو در سایت وجود نداشته، با یک خطای بزرگ و یک برنامه متوقف شده، روبرو شوید. در این گونه موارد تمایل دارید اگر خطا رخ داد (مثلاً آن دارو پیدا نشد) مثلاً نام دارو در یک فایل جداگانه نوشته شود و برنامه ادامه پیدا کرده و به سراغ داروهایی بعدی برود. این کار را می‌توانیم با ساختار try except انجام دهیم. به عبارتی به پایتون می‌گوییم این قسمت از کد را سعی کن انجام دهی (try) و اگر به خطا خوردی (except) فلان کار را انجام بده. مثلاً در کد بالا:

```
print('Hello')

try:
    print(1/0)
except:
    print('you entered the wrong operation!')

print('Bye')
```

بیاییم برنامه را باهم ردگیری کنیم. ابتدا کامپیوتر بدون اشکال خط اول را اجرا کرده و Hello را چاپ می‌کند:

```
print('Hello')
```

سپس وارد قسمت try می شود:

```
try:  
    print(1/0)
```

به دستور $1/0$ رسیده و با خطا روبرو می شود ولی به جای متوقف شدن برنامه، وارد قسمت except می شود:

```
except:  
    print('you entered the wrong operation!')
```

و این جمله را چاپ می کند. در نهایت خط آخر اجرا می شود:

```
print('Bye')
```

این هم خروجی کلی برنامه است:

Hello

you entered the wrong operation!

Bye

اما توصیه می شود حتی الامکان از try except به این شکل استفاده نکنید.

دانشجو: چرا استاد! این که عالیه!

استاد: به این دلیل که خطاها در بیشتر مواقع به ما کمک می کنند که ایراد برنامه را بگیریم و بیشتر از مواقع نادیده گرفتن آن ها عوارض دارد! برای مثال کد زیر را در نظر بگیرید:

```
print('Hello')  
try:  
    prin('Hi')  
except:
```

```
print('you entered the wrong operation!')
```

```
print('Bye')
```

در این مثال ما به اشتباه دستور print را به صورت prin وارد کرده ایم ولی چون در try except به پایتون گفته ایم هر خطایی که دیدی، قسمت try را رها کن و قسمت except را اجرا کن، با این خروجی مواجه می شویم:

```
Hello
```

```
you entered the wrong operation!
```

```
Bye
```

به همین دلیل برنامه نویسان حرفه ای، استفاده از try except را بسیار مکروه می دانند.

دانشجو: پس استاد باید چیکار کنیم؟

استاد: راهکار بهتر این است که بعد از except، نام خطای خود را بنویسیم. اگر یادتان باشد وقتی این دستور را قبلاً نوشتیم:

```
print(1/0)
```

با این خطا روبرو شدیم:

```
ZeroDivisionError
```

به همین دلیل می توانیم جلوی except، نام خطا را بنویسیم:

```
print('Hello')
```

```
try:
```

```
    prin('Hi')
```

```
except ZeroDivisionError:
```

```
    print('you entered the wrong operation!')
```

و در این صورت وقتی کد را اجرا می‌کنیم، چون به پایتون گفته‌ایم تنها وقتی که با خطای `ZeroDivisionError` روبرو شدی این کار را انجام بده و ما با خطای دیگری روبرو شده‌ایم، می‌بینیم که کماکان مانند سابق برنامه متوقف شده و ما متوجه اشکال کارمان می‌شویم:

```

NameError                                Traceback (most recent call last)
Cell In[8], line 4
      1 print('Hello')
      3 try:
----> 4     prin('Hi')
      5 except ZeroDivisionError:
      6     print('you entered the wrong operation!')
```

NameError: name 'prin' is not defined

ولی اگر با خطای تقسیم بر صفر روبرو شویم، این خطا به خوبی مدیریت می‌شود:

```
print('Hello')
```

```
try:
    print(1/0)
except ZeroDivisionError:
    print('you entered the wrong operation!')

print('Bye')
```

خروجی برنامه (هیچ خطایی نمایش داده نشد):

```
Hello
```

you entered the wrong operation!

Bye

دانشجو: استاد می‌تونیم چند خطا رو هم بنویسیم؟

استاد: بله حتماً! منتها با کاما خطاها را جدا کرده و دور آن‌ها یک پرانتز بگذارید

نمی‌دانم چرا ولی فعلاً در پایتون اگر پرانتز نگذاریم، با خطا روبرو می‌شویم):

```
print('Hello')
```

```
try:
```

```
    print(1/0)
```

```
except (ZeroDivisionError, AttributeError):
```

```
    print('you entered the wrong operation!')
```

```
print('Bye')
```

که در اینجا می‌بینید دو خطا را مدیریت کرده‌ایم. حتی می‌توانیم برای خطاهای

دیگر یک `except` مجزا بنویسیم که در صورت وقوع خطایی دیگر، یک کار دیگر

انجام شود:

```
print('Hello')
```

```
try:
```

```
    print(1/0)
```

```
except ZeroDivisionError:
```

```
    print('you entered the wrong operation!')
```

```
except AttributeError:
```

```
    print('We have gotten attribute error')
```

```
print('Bye')
```


دانشجو: استاد درسته که فهمیدیم try except رو بهتره به شکل کلی استفاده نکنیم و خطا رو مشخص کنیم، اما آیا راهی وجود نداره که try except رو به صورت کلی استفاده کنیم و خطای ایجادشده رو هم بفهمیم؟

استاد: در این صورت یک راهکار در پایتون وجود دارد که می‌توانید آن را حفظ کنید:

```
print('Hello')

try:
    print(1/0)
except Exception as e:
    print(str(e))

print('Bye')
```

در این کد گفته‌ایم که لطفاً exception (در پایتون به خطاها، exception می‌گوییم) را در متغیر e ذخیره کن. در نهایت آن را با دستور str(e) به رشته تبدیل کردیم و نمایش دادیم. همان‌طور که مشاهده می‌کنید، الآن می‌توانیم بفهمیم چه خطایی رخ داده (در اینجا division by zero بوده) و نسبت به علت خطا کور نباشیم:

```
Hello
division by zero
Bye
```

خب اگر موافق باشید بعد از این مبحث شیرین و کاربردی، به سراغ تمرین‌ها برویم! 😊

تمرین‌های جلسه نهم

در تمرین هر درس ممکن است که یک نکته جدید هم آموزش داده شود، پس حتماً توصیه می‌کنم آن‌ها را ببینید و انجامشان دهید. تمرین‌ها و به خصوص خطاهایی که دریافت می‌کنید و آزمون و خطاها (و همچنین سرچ‌هایی که در صورت نیاز انجام می‌دهید) بسیار بسیار به روند یادگیری برنامه‌نویسی شما کمک می‌کند. پاسخ نمونه برای تمرین‌ها نیز پس از آن‌ها قرار داده شده است.

تمرین‌ها

۱- در تمرین هفته پیش با disgenet آشنا شدیم. لطفاً همان برنامه هفته پیش را به گونه‌ای تغییر دهید که در انتها مجموعه ژن‌های مشترک این دو بیماری نیز به دست آمده و در یک فایل tsv ذخیره شود.

۲- برنامه‌ای بنویسید که تا مخاطب کلمه q را وارد نکرده، اسم میوه و کالری را از مخاطب دریافت کرده و در یک فایل ذخیره کند.

۳- تمرین سرچ محور: همان‌طور که تا الآن دیده‌اید، ما همیشه به صورت متنی با مخاطب در ارتباط بوده‌ایم (از طریق صفحه سیاه‌رنگی که خروجی یا ورودی‌ها در آنجا نمایش داده می‌شد). راه دیگری نیز وجود دارد که مثلاً وقتی می‌خواهیم از مخاطب ورودی بگیریم، یک پنجره باز شود و مخاطب در آنجا وارد کند. برای این کار کتابخانه‌های زیادی وجود دارند، مانند tkinter، اما یک کتابخانه بسیار ساده به نام easygui وجود دارد. این کتابخانه را به وسیله pip نصب و از آن استفاده کنید و بعضی دستورات آن را ببینید.

پاسخ تمرین‌های جلسه نهم

اکیداً توصیه می‌کنم که اول خودتان تمرین‌ها را حل کنید و سپس به پاسخ‌ها مراجعه کنید. در ضمن این پاسخ‌ها تنها یکی از حالت‌های ممکن نوشتن برنامه هستند و شما ممکن است به شیوه‌ای دیگر برنامه را نوشته باشید و کاملاً صحیح هم باشد. همچنین برای مشاهده توضیحات شفاهی مربوط به این تمرین‌ها، ابتدا ویدئوی جلسه بعد را ببینید (لینک ویدئو در صفحه اول کتاب گذاشته شده است):

« پاسخ تمرین اول:

این برنامه را در هفته‌های قبلی نوشتیم، اما برای اینکه با رویکرد برنامه‌نویسی تابع‌محور هم بیشتر آشنا شویم، این برنامه را به شکل تابع‌محور نوشتیم. به این شکل که قسمت‌قسمت کرده و هر قسمت را به صورت یک تابع نوشتیم و درنهایت تنها کاری که لازم بوده، این است که این سه تابع را صدا بزنیم. ببینید خوانایی کد چقدر افزایش پیدا کرده است و مخاطب برای درک برنامه ما کافی است سه خط آخر برنامه (قسمت main) را بخواند.

```
def read_gene_data(file_path):  
    gene_list = []  
    with open(file_path, 'r') as file:  
        lines = file.readlines()  
        for line in lines[1:]:  
            gene_symbol = line.split(',')[0]  
            gene_list.append(gene_symbol)  
    return gene_list  
  
def load_data():  
  
    genes_disease_a = read_gene_data('Parkinson.csv')
```

```

genes_disease_b = read_gene_data('Asthma.csv')

set_a = set(genes_disease_a)
set_b = set(genes_disease_b)
return set_a, set_b

def intersection(set_a, set_b):
    shared_genes = set_a.intersection(set_b)
    shared_genes_list = list(shared_genes)

    print("Shared Genes:", shared_genes_list)
    return shared_genes

def write_result(genes):
    with open('result.tsv', 'w') as f:
        for gene in genes:
            f.write(gene)
            f.write('\n')

# main
set_a, set_b = load_data()
shared_genes = intersection(set_a, set_b)
write_result(shared_genes)

```

« پاسخ تمرین دوم:

این برنامه را در جلسه لیست‌ها نوشته بودیم:

```

data = []
for i in range(2):
    txt = input('please enter food and calories with

```

```

space:')
food, calorie = txt.split(' ')
data.append([food, float(calorie)])
print(data)

```

کافی است به جای حلقه for که یک حلقه با تکرار مشخص است، از حلقه while استفاده کنیم؛ مثلاً:

```

data = []
while True:
    txt = input('please enter food and calories with
space:')
    if txt == 'q':
        break
    food, calorie = txt.split(' ')
    data.append([food, float(calorie)])
print(data)

```

« پاسخ تمرین سوم:

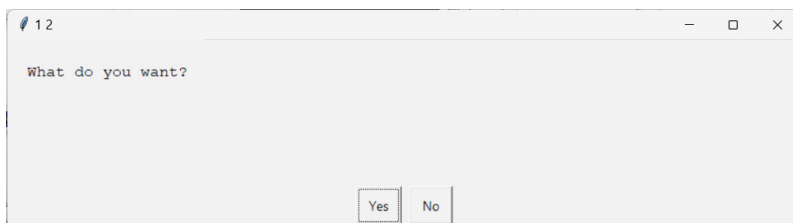
کتابخانه easygui یک راه حل بسیار ساده برای این است که برنامه خود را بیشتر به شکل gui (مخفف graphical user interface) نزدیک کنیم. پس از این که آن را با دستور `pip install easygui` نصب کردیم، می توانیم کمی آزمون و خطا کنیم و بیشتر با آن آشنا شویم. اگر اسم کتابخانه را در گوگل جستجو کنیم، در همان صفحه pypi هم مقدار مثال گذاشته است. مثلاً دستور `ynbox` (که مخفف `yes no box` است):

```

import easygui
choice = easygui.ynbox('What do you want?', ('Yes',
'No'))

```

اگر آن را باز کنیم، با یک پنجره روبرو می‌شویم (شکل ۱):

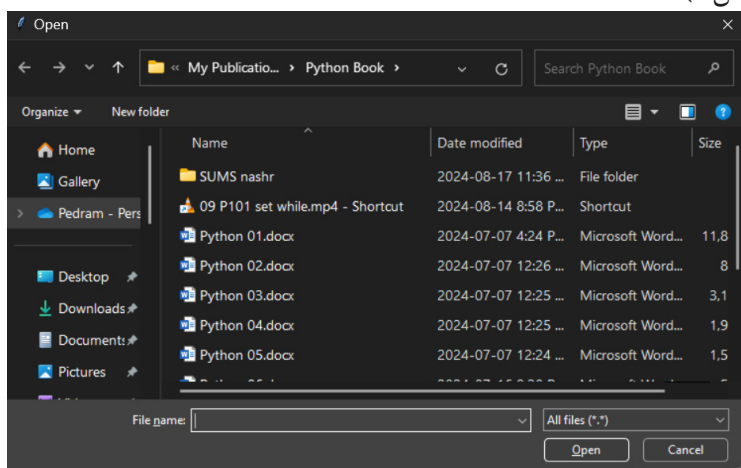


شکل ۱- خروجی دستور ynbox

و جالب است که اگر مخاطب روی `yes` کلیک کند، مقدار متغیر `choice` برابر با `true` و اگر روی `no` بزند مقدار آن برابر با `false` می‌شود. یا مثلاً یک دستور دیگر جالب این کتابخانه، `fileopenbox` است که با اجرای آن یک پنجره باز می‌شود:

`path = easygui.fileopenbox()`

و می‌توان فایل را انتخاب کرد و پس از آن آدرس فایل در `path` ذخیره می‌شود (شکل ۲):



شکل ۲- `fileopenbox`

راستی تا یادم نرفته است، این کتابخانه یک دستور دارد به نام `egdemo` که با اجرای آن یک پنجره باز می شود و به شما تمامی امکانات این کتابخانه را به شکل نمونه نشان می دهد:

`easygui.egdemo()`

دانشجو: عجب چیز جالبیه! استاد میشه گرافیک این پنجره ها رو بهتر کرد؟

استاد: اگر تمایل داشتید بله! هر کدام از این دستورها آرگومان های اختیاری زیادی دارند که مثلاً می توان عنوان پنجره و ... را تغییر داد. اگر هم خواستید که به شکل حرفه ای تر در این زمینه کار کنید، می توانید از کتابخانه هایی مانند `tkinter`، `kivy` یا ... استفاده کنید که هر کدام از این کتابخانه ها را اگر در اینترنت جستجو کنید دوره های آموزشی کوتاه چندساعته از آن ها پیدا خواهید کرد که کار با این کتابخانه ها را فرامی گیرید (برای مثال می توانید از سایت فرادرس استفاده کنید). به صورت کلی تا الآن برنامه هایی که ما می نوشتیم، به شکل متن محور بودند، اما اکثر برنامه هایی که در ویندوز و ... می بینیم، به شکل پنجره محور و گرافیکی طور هستند و با موس و ... کار می کنند. با پایتون می توانیم از طریق این کتابخانه ها، برنامه های خود را به شکل `gui` در بیاوریم و کار با آن ها را برای مخاطب عام راحت تر کنیم.

فصل دهم

ادامه یادگیری و منابع!

تبریک می‌گوییم. تا اینجا ادامه دادن، کار ساده‌ای نیست و بسیاری از افراد آن را رها می‌کنند و همیشه به دنبال فرصتی هستند که یک وقت مناسب و خوب برای یادگیری کامل و بی‌نقص داشته باشند؛ اما شما با وجود مشکلات و کارهایی که در کنار این دوره داشتید، با وجود تمام این بهانه‌ها، دوره را به هر شکل که بود به پایان رساندید. گام بعدی، آموزش کتابخانه‌های `numpy`، `pandas` و `matplotlib` (البته `seaborn` هم از همان `matplotlib` استفاده می‌کند) است. در این رابطه اگر عمری باقی باشد، در جلد بعدی این مجموعه به آن می‌پردازیم. اگر هم تمایل دارید که از پایتون به جای تحلیل داده و یادگیری ماشین و هوش مصنوعی، به حیطه‌های دیگر برای مثال طراحی سایت یا ... بپردازید، می‌توانید با جستجو در اینترنت و یا شرکت در دوره‌های متفاوت کوتاه‌مدت (فرادرس در سایت‌های فارسی برای این کار بسیار عالی است) آن‌ها را فرا بگیرید. یادتان باشد، همان‌طور که در طول دوره بارها به آن اشاره کردیم، هیچ برنامه‌نویسی همه‌چیز را نمی‌داند و مهم‌ترین کار ما سرچ کردن است که با وجود هوش مصنوعی‌هایی مانند چت‌جی‌پی‌تی این کار از قبل ساده‌تر هم شده است.

در این کتاب از منابع زیر استفاده شده است:

1) Effective Python: 90 Specific Ways to Write Better Python

در این کتاب از Brett Slatkin که برنامه‌نویس بسیار با سواد است، ۹۰ راه برای زیباتر، خواناتر و بهینه‌تر نوشتن کدهای پایتون نوشته شده است. خواندن این کتاب بسیار به من کمک کرده است. در این رابطه، دوره پایتون عمیق از استاد عزیزم، جناب دکتر حامد قاسمی از آکادمی روباتک که به شرح این کتاب و همچنین مباحث شی گرای به زبان بسیار ساده و جذاب توضیح داده‌اند را توصیه می‌کنم.

2) Python Official Docs

مستندات و help خود پایتون در سایت رسمی آن؛

3) Automate the Boring Stuff with Python

این کتاب بسیار جذاب از Al Sweigart است. این نویسنده کارهای خسته‌کننده در حیطه‌های مختلف و کاربردی را با پایتون انجام داده است. این کتاب به صورت رایگان در سایتی به همین عنوان کتاب قرار داده شده است. در این رابطه خود من هم دو دوره در سایت فرادرس تدریس کرده‌ام که می‌توانند کمک کنند باشند. دوره پروژه‌محور مدیریت فایل‌ها با پایتون (در این دوره کارهای خسته‌کننده انبوه با فایل‌ها را با پایتون به راحتی انجام می‌دهیم) و همچنین دوره خودکار سازی برنامه‌های ویندوز با کتابخانه pywinauto (در این دوره کنترل موس و کیبورد را به پایتون می‌دهیم و به راحتی کارهای خسته‌کننده وابسته به موس و کیبورد را با پایتون انجام می‌دهیم) که اگر تمایل داشتید می‌توانید از محتوای آن‌ها که مثل همین کتاب به زبان ساده و با مثال‌های کاربردی گفته شده‌اند، استفاده کنید.

4) GeeksforGeeks

در اکثر موارد که موضوعی را در مورد پایتون جستجو می‌کنیم، با این سایت محبوب

روبرو می‌شویم. در این سایت، مبحث موردنظر به زبان ساده و با مثال آموزش داده شده است که می‌تواند بسیار کمک‌کننده باشد و در مباحث این کتاب هم جاهایی از این وب‌سایت استفاده شده است.

5) Grokking Algorithms: An Illustrated Guide for Programmers and Other Curious People

کتاب درک الگوریتم؛ راهنمای تصویری برای برنامه‌نویس‌ها و افراد کنجکاو با زبان ساده و کاریکاتورهای جذاب و مثال‌های روزمره، الگوریتم‌های رایج در کامپیوتر را مورد بحث قرار می‌دهد که به‌شخصه برای من بسیار مفید بوده است. خوشبختانه ترجمه فارسی این کتاب را جناب مهران افشار نادری به‌رایگان منتشر کرده است که می‌توانید آن را با جستجو در گوگل یا از این لینک دانلود کنید.

از طرفی همین دوره با کیفیت ویدئوی بهتر و تجارب بیشتر و همچنین با امکان ارسال تکالیف و منتورینگ شخص بنده نیز در «ایزی آکادمی» وجود داشته که با جستجو کردن «دوره پایتون برای علوم پزشکی از صفر ایزی آکادمی» در گوگل می‌توانید آن را پیدا کرده و در آن دوره را نیز ثبت‌نام کنید؛ اما درعین حال همان دوره رایگان نیز دوره کاملی است و هیچ اجباری به ثبت‌نام در این دوره نیست.



ایزی آکادمی / برنامه نویسی / دوره جامع برنامه‌نویسی پایتون از صفر برای علوم پزشکی و دارویی!

شکل ۱- رسالت ما در ایزی آکادمی، آموزش مباحث تخصصی دانشگاهی به زبان ساده و خودمانی است



شکل ۲- <https://www.easyacademy.ir>

این هم راه ارتباطی (لینکدین، تلگرام و ایمیل) در رابطه با پرسش‌های شما:

p.porbaha@gmail.com

@PedramPorbaha

منابع

1. Parvizi, Y., Sadati, S. M., Porbaha, P., Masumi, S., Mahdian, S., Vafaei, S. A., & Afshar, S. (2024). Identifying Key Genes and Approved Medications Associated with Major Depressive Disorder Using Network Analysis and Systems Biology. *Iranian Journal of Psychiatry*, 19(4), 367-383.
2. Porbaha, Pedram, et al. "A Comparative Mathematical Analysis of Drug Release from Lipid-Based Nanoparticles." *AAPS PharmSciTech* 25.7 (2024): 208.
3. Soleimanifard, M., Porbaha, P, et al. (2024). Evaluating the Stability of Three Protein Drugs: A Ft-Ir and Clustering Approach for Levemir, Pdpoe-tin, and G-Csf. (In submitting process)
4. Porbaha, Pedram ,Mohammad Hasannejad, et al. Exploring Potential Drug-Drug Interactions: A Cross-Sectional Study of 1 Million e-Prescriptions Across Medical Specialties in Shiraz, Iran (2021-2024), (In submitting

process)

5. Porbaha, Pedram ,Mohammad Hasannejad, A Specialized Dataset of Persian Pharmaceutical Questions and Expert Responses (In submitting process)

6. Slatkin, B. (2019). Effective Python: 90 Specific Ways to Write Better Python. Addison-Wesley Professional.

7. Sweigart, A. (2019). Automate the boring stuff with Python: practical programming for total beginners. No Starch Press.

8. Bhargava, A. Grokking Algorithms: An illustrated guide for programmers and other curious people. 2016.

واژه‌نامه

C

C++..... ۱۶، ۱۷، ۱۸، ۵۰

D

Disgenet..... ۴۴۷، ۴۶۱، ۴۷۷

DNA..... ۸۰، ۸۱، ۲۰۲، ۲۰۳، ۲۰۶، ۲۱۱، ۲۱۲،
۲۱۳، ۲۲۰، ۲۲۱، ۲۲۲، ۲۲۵، ۲۸۶، ۴۱۶، ۴۳۰، ۴۳۲، ۴۳۳، ۴۳۴، ۴۳۵، ۴۳۶، ۴۴۱**I**

IDE..... ۳۷، ۸۷

M

matplotlib..... ۷، ۱۷۴، ۴۱۵، ۴۸۵

N

numpy..... ۷، ۱۷۴، ۴۱۵، ۴۸۵

P

pandas..... ۷، ۱۷۴، ۴۱۵، ۴۸۵

S

seaborn..... ۷، ۴۸۵

/

اسمبلی..... ۱۶

آ

آناکوندا..... ۱۹۱، ۸۶، ۸۵، ۳۷، ۳۴، ۳۳

پ

پابمد..... ۳۵۵

پروتئین..... ۴۴۰، ۴۳۶، ۴۳۴، ۴۳۳، ۴۱۷، ۴۱۶، ۳۳۵، ۳۲۲، ۳۱۹، ۲۸۶

ت

تحلیل گره داده..... ۲۷

ج

جاوا..... ۲۰

د

داده کاوی..... ۲۶

دارو..... ۴۷۱، ۴۶۱، ۴۶۰، ۴۵۹، ۴۰۹، ۴۰۵، ۳۳۳، ۳۲۹، ۳۲۸، ۳۲۶، ۱۱۴، ۱۰۶، ۲۶

ز

زبان برنامه نویسی R..... ۱۵

ع

علم داده..... ۲۶، ۲۰

ک

کامپایلر..... ۱۵

گ

گوگل..... ۶، ۹، ۲۱، ۲۸، ۲۹

۳۰، ۳۱، ۳۲، ۳۴، ۵۰، ۵۱، ۷۶، ۷۷، ۷۹، ۸۶، ۱۰۲، ۱۳۱، ۱۳۲، ۱۵۸، ۱۶۰، ۱۶۱، ۱۷۵، ۱۷۶، ۱۷۹، ۱۸۰، ۱۸۱، ۱۹۲، ۱۹۷، ۱۹۸، ۱۹۹، ۲۰۰، ۲۲۷، ۲۳۴، ۲۳۹، ۲۴۸، ۲۸۴، ۳۱۱، ۳۳۵، ۳۵۱، ۳۵۳، ۳۵۴، ۳۸۰، ۴۰۰، ۴۰۴، ۴۳۴، ۴۸۰، ۴۸۷

گوییدیدووان روسوم..... ۱۸، ۱۹

گیت‌هاب..... ۴۵۱

م

متلب..... ۲۱

مفسر..... ۱۵

ن

نماد علمی..... ۴۷

ه

هوش مصنوعی..... ۵، ۷

۲۰، ۲۷، ۲۸، ۳۰، ۳۱، ۳۲، ۳۳، ۷۵، ۷۶، ۷۷، ۱۰۲، ۱۱۳، ۱۳۲، ۱۶۱، ۱۶۲، ۱۶۳، ۱۶۴، ۱۷۳، ۱۷۷، ۱۷۹، ۲۷۶، ۳۱۱، ۳۱۲، ۳۱۹، ۳۲۱، ۳۳۳، ۳۳۵، ۳۸۰، ۴۳۵، ۴۸۵

ی

یادگیری ماشین..... ۵، ۷، ۲۰، ۲۷، ۳۲، ۳۳، ۱۷۳، ۱۸۰، ۳۵۵، ۴۸۵

