

# قدم به قدم با پایتون برای درک برنامه نویسی

نویسنده: محمد کمال

۳ -----

نمونه آموزش

۷ -----

مقرمات برنامه نویسی

(آشنایی با سفت افزار - تبدیل مبنا - معرفی وبسایت Quera - آشنایی با تاریخچه زبان های برنامه نویسی)

۲۲ -----

پایتون

به ترتیب موارد زیر:

print() - Variable naming & Working with variable - input() - if -  
while - for - boolean - Time complexity - Function - String - for +  
string - Comment - String concatenation - ASCII - List - List  
comprehension - Nested for - Sorting algorithms - 2D list - Recursive  
function - Dictionary - Match case

## نمونه آموزش

توی این آموزش سعی کردم که اون اون خمیرمایه ذهن شما رو درست شکل بدم. یعنی قدم به قدم همراه شیم و مثل یه فردی که جلوتون نشسته داره بهتون توضیح میده و خط به خط داره همراه شما کد می‌زنه عمل کنم. یعنی نیام پاسخ کامل رو بذارم جلو روتون و بگم خب تموم! بلکه پیام خط فکری بدم بهتون. بگم خب برای فلان کار نیاز به فلان چیز داریم. پس این خط رو می‌نویسیم. بعدش فلان نیاز داریم. پس فلان می‌کنیم. یعنی قدم به قدم پیش میریم.

یکی از مهم‌ترین نکات اینه که الان که شما در ابتدای راه یادگیری برنامه‌نویسی هستین، خمیرمایه ذهنتون درست شکل بگیره. یعنی از همین الان یاد بگیرین تمیز کد بزنین. درست پیش برین. بتونین کدتون رو بهتر و سریع‌تر کنین. برای همین هم من تلاش کردم که از همین الان این موضوع رو بهتون یاد بدم تا ذهنتون بهتر شکل بگیره.

«ریشه بزرگترین مشکلات برنامه‌نویس‌ها، آموزش بدی هست که در ابتدا یادگرفتن. اگر شما برنامه‌نویسی رو بد یاد بگیرید، خمیرمایه ذهنت بد شکل می‌گیره و تا ابد درگیر مشکلات و سفتیاش هستی!»

اینجا در کنار هم سعی می‌کنیم برنامه‌نویسی رو به شکل بهتری یاد بگیریم.

برخلاف بعضی آموزش‌های دیگه، من همه قوانین یه موضوع رو همونجا نمیگم. بلکه تلاش کردم کاربردیش کنم و هروقت نیاز شد، اون موضوع رو یادتون بدم. اینطوری با انبوهی از نکات مواجه نمیشین و بهتر می‌تونین یاد بگیرین.

مثل یه بچه کوچیک که تازه می‌خواد زبون یاد بگیره، از روز اول تمام کلمات رو بهش یاد نمیدن که! بلکه مثلاً بچه می‌خواد سیب بخوره، بهش میگن این سیبه. کم‌کم در زمانی که نیاز به اون مسأله داره، یادش می‌گیره.

برای همین هم من سعی کردم آموزش رو همینطور پیش ببرم. یعنی یه مسأله مطرح می‌کنم و بعدش میگم که این مسأله نیاز به فلان داره. حالا بیایم با فلان آشنا شیم.

- این آموزش پیش‌نیازی داره؟ یعنی اگر دانشجوی کامپیوتر هم نیستم می‌تونم بخونم؟ اگر دبیرستان یا راهنمایی هستم هم می‌تونم بخونم؟

+ پیش‌نیاز نداره و بله آموزش رو از صفر نوشتم و همه می‌تونن استفاده کنن.

- چه زمانی شروع به یادگیری برنامه‌نویسی کنم؟ در چه سنی؟

+ درواقع یادگیری برنامه‌نویسی زمان نداره! دنیا دنیای کامپیوترهاست. کامپیوتر همه جا حضور داره. از دستگاه‌های پزشکی تا دستگاه‌های فیزیکی. کامپیوترها کارهای خسته‌کننده رو به صورت سریع‌تر و بهتری

انجام میدن. به نظرتون یه فیزیک‌دان خودش بشینه دستی محاسبات رو انجام بده براش ساده‌تره یا بده کامپیوتر براش انجام بده؟ خب مطمئناً کامپیوتر. چون دقیق‌تر و سریع‌تره.

درواقع برنامه‌نویسی شبیه ابزار هست. دنیا کمی عوض شده. همونطور که ما یه زمانی ضرب و تقسیم یاد می‌گرفتیم، الان نیازه که در حد ابتدایی بتونیم با کامپیوتر صحبت کنیم (برنامه‌نویسی کنیم)، تا فارغ از رشته‌ای که داریم درس می‌خونیم (پزشکی، شیمی، فیزیک و...) کامپیوترها در کارها به ما کمک کنند. زمان خاصی هم نداره. از حتی اواخر دبستان و اوایل راهنمایی هم میشه شروع به یادگیری برنامه‌نویسی کرد. همونطور که در کتاب «کار و فناوری» پایه هفتم به این موضوع پرداخته شده.<sup>۱</sup>

آیا شنیدین که میگن هوش مصنوعی قراره آدما رو از کار بیکار کنه؟  
درواقع هوش مصنوعی شما رو قرار نیست بیکار کنه! آدم‌هایی که می‌تونن از هوش مصنوعی کمک بگیرن، شما رو بیکار می‌کنن. همونطور که تاکسی‌های اینترنتی که اومدن، شغل تاکسی‌های عادی تحت تأثیر قرار گرفت، هوش مصنوعی هم که اومده، شغل کسانی که نتونستن از این چیز جدید برای بهبود شغلشون استفاده کنن، تحت تأثیر قرار می‌گیره.

تا وقتی ما از کامپیوترها رو یه عنصر کمکی ببینیم و از کامپیوترها استفاده کنیم - و نه کامپیوترها از ما -<sup>۲</sup> نگرانی بابت اینکه قراره شغلمون تحت تأثیر قرار بگیره، نداریم.  
به قول استیفن هاو‌کینگ:

"If machines produce everything we need, the outcome will depend on how things are distributed." -Stephen Hawking

یعنی اگر روزی برسه که اکثر چیزها رو کامپیوترها انجام بدن، نتیجه نهایی بیشتر بستگی به این داره که آیا سود این موضوع به همه میرسه یا نه؟! اگر مثل تاکسی‌های اینترنتی باشه و ما نتونیم همه رو با کامپیوترها آشنا کنیم و سودش رو بهشون برسونیم، بله یه سری از افراد شغلشون رو از دست میدن.

- راهنماییا چی هستن؟

+ ببینین برنامه‌نویسی مثل ریاضیاته. شما با ریاضی خوندن، ریاضی‌دان نمیشین! بلکه باید تمرین کنین و تکرار انجام بدین و خودتون تلاش به حل مسأله کنین.

برنامه‌نویسی هم همینطوره. یعنی شما تا وقتی خودتون به جواب سؤال پی نبرین، صرفاً پاسخنامه‌خوانی به درد نمی‌خوره. برای همین هم من یه سری راهنمایی براتون گذاشتم که اگر متوجه حل سؤال نشدین، راهنمایی‌ها رو بخونین و تلاش کنین با راهنمایی که کردم، پاسخ رو به دست بیارین. بدیهیه که اگر بدون خوندن راهنمایی تلاش به حل سؤال کنین، ذهنتون بیشتر فکر می‌کنه و بهتر درگیر ماجرا میشین. پس سعی کنین از ذهنتون کار بکشین :)

- هایلایتا که رنگاشون متفاوت، دلیل خاصی دارن؟

۱ طبق نظرسنجی stackoverflow هم، حدود ۴۰ درصد افرادی که تازه شروع به یادگیری برنامه‌نویسی کردن، از دوران قبل دانشگاه این رو رقم زدن! یعنی در دوران مدرسه (راهنمایی و دبیرستان):

<https://survey.stackoverflow.co/2023/#education-ed-level-learn>

۲ منظور این هست که عملاً درگیر و پایند کامپیوترها نشویم. بدونیم داریم چیکار انجام میدیم و عملاً به قولی جوری نباشه که همیشه در شبکه‌های اجتماعی باشیم و عملاً کامپیوتر بخشی از زندگی ما رو به هدر بده یا اینکه ما رو کنترل کنه. (با چیزایی که توشه) درواقع هر چیز مزایا و معایبی داره. اگر سعی کنیم درگیر معایب نشیم و سودهاشو بگیریم، مشکلی نخواهیم داشت.

+ آره!

خاکستری: یا به موضوع کلی عنوان میشه یا هنوز کد کامل نشده و داریم هی مرحله به مرحله کد رو کامل می‌کنیم.

صورتی: کد اوکیه ولی یکی دو سه جاش رو میشه تغییر داد و بهتر کرد.

آبی: تغییرات کد نسبت به مرحله قبلی و چیزایی که اضافه شدن.

سبز: کد درست و نهایی.

- جاهایی نوشتی «متوسط». آیا معنای خاصی داره؟

+ بله. یعنی مفاهیم یه خرده سطحشون از سطح ساده بالاتره و آدمایی که می‌خوان بیشتر یاد بگیرن بخوننش.

- پیشنهاد خاصی داری بهمون بگی؟

+ بله! سعی کنین زبان انگلیسیتون رو تقویت کنین. هرچی بیشتر بهتر! تأکید می‌کنم هرچی بیشتر بهتر! بهترین منابع دنیا به زبان انگلیسی هستن. شما با فارسی خوندن نمی‌تونین پیش برین! کتاب‌های فارسی ترجمشون خیلی بده و بدتر گیجتون می‌کنن.

+ تمرین تمرین تمرین! تمرین خیلی مهمه. یعنی اگر من یه مثال زدم، شما مثال رو عوض کنین و حالت دیگه‌ای ازش رو حل کنین. اگر من گفتم این کار کنین، شما هم اون کار و هم‌نوع دیگشو

- نظری دارم برای بهبود این آموزش. می‌تونم بگمش؟

+ آره حتماً! خوشحال میشم!

## رونیت

من این کار را به صورت رایگان منتشر کرده‌ام و برای دانشی که یاد گرفته‌ام و همچنین مدت زمان تایپ مطالب، زمان بسیار زیادی از وقتم را صرف کرده‌ام. به عقیده‌ی من، **متعالی‌ترین و عالی‌ترین** شیوه‌ی تولید دانش، اینه که دانش باید **رایگان** و **آزاد** در دسترس همه باشه؛ که همه‌ی افراد فارغ از جنسیت و درآمد و هرچیز دیگری، به آن دسترسی داشته باشن. درواقع گسترش علم باید رایگان باشد، اما تولیدش خیر! همانطور که شما رایگان به مدرسه می‌روید ولی معلم از پول مالیات و فروش نفت حقوق می‌گیرد، این گسترش علم هم همینطور است!

هرکسی که فکر کرد این دانش کمک‌کننده است و دوست داشت به پیشرفت دانش آزاد و رایگان کمک کند و همچنین تمایل داشت بابت وقتی که من برای آموزش دانش خودم به صورت رایگان و آزاد به دیگران صرف کردم، کمک کند، می‌تونه از طریق لینک زیر، به من دونیت (کمک/حمایت مالی) کنه:

<https://zarinp.al/mkamal>

چیزهایی که باید اضافه شوند (اگر بلد هستین، لطفاً بنویسین و بگین تا در نسخه بعدی نوشته شما رو با کردیت خودتون منتشر کنم):

- شروع برنامه‌نویسی با نصب IDE و Code editor و قابلیت‌های موجود در <sup>۳</sup>شون
- تفاوت IDE و Code editor
- تفاوت compiler و interpreter به صورت فوب و قابل فهم و درعین حال عمیق
- کمی بیشتر درباره Dictionary
- try and except and raise
- کار با فایل
- کمی بیشتر درباره سفت‌افزار با دیر برنامه‌نویسی

## حق نشر

این کتاب به وسیله لایسنس زیر عرضه شده:

<https://creativecommons.org/licenses/by/4.0/>

استفاده از مطالب این کتاب به شرط ذکر منبع و دادن منبع، بلامانع است.

---

۳ مثل Format on save، رنگا، تما، افزونه‌ها چی هستن و چیکار می‌کنن، فعال‌سازی گیت برای دیدن تغییرات

## برنامه‌نویسی (Programming) چیست؟

استفاده از کامپیوتر و برنامه‌ریزی کردن کامپیوتر برای انجام یک سری کار. مثلاً من می‌تونم یک ربات بسازم که بگم روی GPS نگاه کن و ببین اگر من نزدیک خونه هستم، برو به سمت ماکروفر و دکمه ۵ دقیقه رو بزن تا غذا گرم شه. درواقع کامپیوتر مثل ما آدم‌ها نیست. باید به صورت خیلی دقیق و ریز بهش دستور بدیم. یعنی درواقع برنامه‌ریزی کنیم که اگر فلان شد، فلان کار کن. اگر  $2 + 3$  دادن بهت، جمع بزن و ....

## چرا اصلاً برنامه‌نویسی؟

برنامه‌نویسی دیگه داره میشه شبیه دونستن ضرب و تقسیم. درواقع ایده بعضی کشورای خارجی این شده که ما باید به بچه‌ها برنامه‌نویسی یاد بدیم. نه لزوماً برای رشته کامپیوتر. بلکه برای همه رشته‌ها! مثلاً یک مهندس فیزیک، محاسبه نیروی فیزیک رو دستی انجام بده ساده‌تره یا بده کامپیوتر واسش انجام بده؟ خب مطمئناً کامپیوتر! پس درواقع برنامه‌نویسی رو به صورت ابزاری استفاده کنه. یعنی افراد بتونن از کامپیوتر کمک بگیرن که کارهای مختلف رو ساده‌تر و سریع‌تر انجام بدن.

همونطور که یک چیزی به نام ماشین حساب داریم، شما می‌تونین برنامه بنویسین که محاسبات سخت ریاضی رو حساب کنه. می‌تونین برنامه بنویسین که ترافیک شهر رو بهتون بگه. می‌تونین به عنوان یک مَنجِم، برنامه‌ای بنویسین که روند چرخش ستاره‌ها رو توی کامپیوتر شبیه‌سازی کنه که بدونین کی فلان شهاب‌سنگ میخوره به زمین. پس هر رشته‌ای برین، کامپیوتر و برنامه‌نویسی ابزار خیلی خیلی مفیدیه که خوبه یادش بگیرین. درواقع برنامه‌نویس‌ها، حل‌کننده‌های مسأله با کمک کامپیوتر هستن. چون همیشه در حال حل مسأله هستن، ذهن خلاق‌تر و راهگشاتری دارن.

خب حالا برگردیم به اینکه برنامه‌نویسی کجای رشته کامپیوتر قرار داره؟ برنامه‌نویسی درواقع ابزاریه. شما مثلاً می‌خوای یک سیستم عامل طراحی کنی، میری برنامه‌نویسی سطح پایین انجام میدی. زبونایی مثل Rust و C. اما یک سوال! تا وقتی ندونی کامپیوتر چه جور کار می‌کنه، می‌تونی ویندوز و سیستم عامل بنویسی؟ نه! پس درواقع شما صرفاً داری علمی که از نحوه کار کامپیوتر داری رو به زبونی برنامه‌نویسی پیاده‌سازی می‌کنی! درواقع دانش اصلی، اون علمی هست که کامپیوتر چه جور کار می‌کنه؟

مثلاً یک نمونه (این رو بعد یادگرفتن if بخونین):

فرض کنیم من به برنامه‌ای می‌خواهم بنویسم که به سری داده (ages) رو دونه‌دونه بهش بدم (مثلاً سن افراد)، بعد بیاد تعداد سن‌های بالای ۱۸ رو بهم بده.

```
count = 0
```

```
if ages > 18:
```

```
    count = count + 1
```

توضیح: قاعدتاً اول تعداد برابر صفره. بعد سن‌ها رو میدیم بهش. (اینکه چه‌جوری بهش میدیم رو فعلاً کاری نداشته باشیم!) بعدش اگر هر دونه بزرگ‌تر از ۱۸ بود، می‌گیم count جدید ما برابر count قبلی بعلاوه یک هست. (یکی رو اضافه می‌کنیم بهش) یعنی درواقع هر دفعه یکی از سنا میاد و اگر بیشتر از ۱۸ بود، یکی به count اضافه میشه.

خب به نظرتون این کد در دو حالت زیر، چه‌زمانی سریع‌تره؟

```
1 - 3 - 4 - 6 - 8 - 10 - 20 - 21 - 24 - 25 - 29 - 30
```

```
21 - 4 - 29 - 3 - 30 - 8 - 10 - 21 - 1 - 6 - 25 - 20
```

حالت اول مرتب شده هست، حالت دوم هم نامرتب. (توجه داشته باشیم که مرتب‌کردن اعداد، خودش مقداری زمان می‌بره.)

- خب اینکه خیلی سادس! مطمئناً حالت اول سریع‌تره! چون نیاز نیست من به دور اول مرتبش کنم که زمان الکی ببره!

+ خب خب خب (: در نگاه اول آره به نظر میاد حالت اول سریع‌تر باشه، اما درواقع حالت دوم سریع‌تره! درواقع اگر من از نحوه کار CPU (مغز) کامپیوتر آشنا باشم، می‌دونم که CPU ها به سری قابلیت دارن که تأخیر رو کاهش بدن.

بذارین به مثال بزنم! فرض کنیم من منشی به دکترم. می‌بینم که پنج دفعه قبلی که وارد مطب شدم، می‌خواستی پروندتو بهت بدم که ببری پیش دکتر. خب به نظرتون کدوم منطقی‌تره؟

۱- منتظر بمونم شما پرسی کنار میز من و من تازه بگردم دنبال پروندت.

۲- تا از دور دیدمت، بگردم دنبال پروندت که تا رسیدی، پروندتو بدم بهت و زیاد منتظر نمونی.

قاعدتاً حالت دوم بهتره! چون از تأخیر جلوگیری می‌کنه.

شاید شما این دفعه کار دیگه‌ای داشته باشی، اما شانس اینکه بازم پروندتو بخوای زیاده و اگر من پروندت رو آماده داشته باشم، خیلی زود بهت میدم و کارا خیلی خیلی سریع‌تر پیش میره.

درواقع CPU هم همیشه می‌خواد از تأخیر جلوگیری کنه. یعنی می‌گه آقا من الان خط ۲ کد هستم باید چندتا چیز رو با هم جمع بزنم. حالا تا وقتی که متغیرها از حافظه میان، یکم طول میکشه. خب من بیکار نشینم! برم خط بعدی هم اجرا کنم که یکم جلو بیوفته کارا.

خب بیایم رو کد. کامپیوتر میرسه به if. خب پیش خودش می‌گه که نمیدونم که داخل if باید برم یا نه! خط چیکار کنم؟ حدس بزنم که اگر احتمالش زیاده وارد if بشم، خب برم توش. وگرنه دستورای بعد if رو جلو جلو اجرا کنم.



نگاه می‌کنه به قبل می‌گه عه! از ۳ بار قبلی که رسیدم به if، من هر ۳ بار رفتم توش! پس ایندفعه هم شانس بالایی هست که باز بخوام برم توش. برای همین میره دستورای توی if رو جلو جلو حساب می‌کنه.

حالا چرا مرتب‌شده سریع‌تره؟

چون کامپیوتر شروع می‌کنه از اول، دو سه تای اول می‌بینه وارد if نمیشه. اما از یه جایی به بعد، می‌بینه داره وارد if میشه. پس می‌گه بار بعدی که رسیدم به if، توی زمانی که شرط داره چک میشه، من بیکار نمیشیم! میرم توی if و چیزای داخلش رو حساب می‌کنم که یکم بیوفتیم جلو. درواقع به دلیل اینکه یه سری محاسبات جلو‌جلو انجام میشه، حالت مرتب‌شده سریع‌تره!

ولی توی مرتب نشده، می‌بینه یه بار میره تو if، یه بار نمیره و اصلاً نمی‌فهمه باید چیکار کنه و کدوم دستورا رو جلو جلو اجرا کنه که سرعت زیاد شه!<sup>۴</sup>

درواقع دونستن دانش درباره نحوه کار کامپیوتر و نوشتن برنامه طبق اون نحوه کار و همچنین نحوه نوشتن الگوریتم سریع، باعث میشه در داده‌های بسیار بزرگ، مثل سرچ کردن گوگل، صدها برابر کد سریع شه!

خب حالا یه سوال! فرض کنین شما مدیر گوگل هستین. کدوم رو ترجیح می‌دین؟  
یه برنامه‌نویس استخدام کنین که دانش بالایی داره و جوری با نحوه نوشتار کد و الگوریتم آشناست که کدی می‌نویسه که ۱ ثانیه طول میکشه تا جواب بده.  
یه برنامه‌نویس دیگه‌ای رو استخدام کنین که با نحوه نوشتار کد آشنا نیست؛ با سیستم هم آشنا نیست؛ کدش ۲۰۰۰ ثانیه طول میکشه.  
خب مطمئناً شما اولی رو انتخاب می‌کنین.

این ارقام خیالی نیستن! ما خودمون توی همین آموزش یه الگوریتم بررسی اینکه یه عدد اوله یا نه می‌نویسیم. بعدش سعی می‌کنیم بهینش کنیم و در آخر، بهینه‌ترین کد، می‌تونه تا بالای ۳,۰۰۰ برابر سریع‌تر از کد اولمون بشه! بله! بالای ۳,۰۰۰ برابر سریع‌تر! از همین الان می‌خوام بهتون یاد بدم که سعی کنین درست کد بنویسین و از همین الان بتونین خمیرمایه ذهنتون رو درست شکل بدین!

<sup>۴</sup> به این میگن «branch predictor». مبحث سختیه و فعلاً نمی‌تونین بفهمیدش درست! ولی اگر دانش از رجیستر و کمی زبون C و یا اسمبلی و کمی نحوه ران شدن برنامه‌ها دارین، لینکای زیر رو بخونین:

Why is processing a sorted array faster than processing an unsorted array? -Stackoverflow:  
<https://stackoverflow.com/questions/11227809/why-is-processing-a-sorted-array-faster-than-processing-an-unsorted-array>

Spectre Attack: <https://spectreattack.com/>

یکم میریم بالاتر، شما برنامه‌های مختلف رو می‌نویسین. مثلاً تلگرام و اینا. اینجا هم نیاز به اینکه چه جوری کد بنویسین که خوب باشه دارین ولی دیگه نیاز به مسائل سخت‌افزاری کمتره توشه. ولی نیازه که انواع الگوریتم‌ها رو یاد بگیرین و بدونین چه جور برنامه بنویسین.

- یعنی چی؟

فرض کنین شما یه عدد بین ۱ تا ۱۰۰ توی ذهنتون انتخاب کردین. من باید عدد رو پیدا کنم. راه اول اینطوره که من سرمو بندازم پایین و یه حدس بزنم. مثلاً ۲۰. بعد همینطوری بی‌منطق حدس بزنم و برم جلو.

راه دوم بهش می‌گن باینری سرچ (جست‌وجو دودویی)

میگه من اولین حدسم رو عدد وسطی انتخاب می‌کنم. میگم ۵۰. حالا میگم بیشتره یا کمتر؟ شما میگی بیشتر. پس همینجا ۵۰ تای اول خط خورد و نیاز نیست توش بگردم. پس حالا عدد ما از ۵۰ تا ۱۰۰ هست.

دوباره حدس رو روی نصف بازه می‌زنم. میگم ۷۵ هست؟ شما میگی نه. میگم بیشتره یا کمتر؟ شما میگی کمتر.

خب پس عدد بین ۵۰ تا ۷۵ هست. دوباره نصف. میگم مثلاً ۶۳ هست؟ شما میگی نه. میگم بیشتر یا کمتر و همینطور ادامه. به همین راحتی هی بازه رو نصف می‌کنم و به سرعت جواب رو به دست بیارم. این الگوریتم بسیار سریع‌تر از الگوریتم عادی هست که سرمو بندازم پایین و پیداش و دونه‌دونه جلو برم و حدس بزنم.

باز میانین بالا. چیزایی مثل طراحی وبسایت و اینا. مخصوصاً قسمتی که کاربر رو صفحه می‌بینن، خیلی خیلی کم نیاز به اون درک سخت‌افزاری داره. شما مثلاً خوشگلی وبسایت، اینکه این دکمه بزنم چی رخ میده و اینا رو طراحی می‌کنین و خب قشنگی هست.

برنامه‌نویسی یه ابزاره. این ابزار همه جا کاربرد داره. یعنی شما هر رشته‌ای بخواین برین، باید تا حدی برنامه‌نویسی بلد باشین. نه اینکه برنامه‌نویس باشین، بلکه بتونین ابزار بنویسین که کارهاتون رو انجام بده.

بذارین یه مثال بزنم. فرض کنین بخواین یه گاوصندوق رو باز کنین. رمزشم ندارین. چیکار می‌کنین؟ قاعدتاً دونه دونه ترکیب رمزا رو امتحان می‌کنین تا وقتی بالاخره باز شه دیگه. ترکیبیات سال دهمتون!

همین رو برای یافتن پسورد در امنیت استفاده می‌کنن. ولی امتحان کردن همش دونه‌دونه سخت و زمان‌بره. برای همین هم هست که شما مثلاً با همین پایتون، ابزار می‌نویسین که بره خودکار امتحان کنه. درواقع کارها رو خودکارسازی می‌کنه. درواقع شما توی امنیت هم به ابزارنویسی احتیاج دارین. شما رشته فیزیک هم بری، در کارهایی مثل مدلسازی حرکت ماشین، جمع‌آوری و محاسبه محاسبات نیاز به برنامه‌نویسی پیدا می‌کنین. ولی برای ابزارنویسی و خودکارسازی!

## • آشنایی کوتاه با سفت افزار با دید برنامه نویسی

کامپیوتر تشکیل شده از یه سری قطعه الکترونیکی. یکی از اونها اسمش CPU هست. CPU مغز کامپیوتره. کارا رو قطعه CPU انجام میده. کارایی مثل ضرب کردن و جمع کردن تا کارهایی مثل اینکه یه پیام رو می نویسین پردازش کنه و بفرسته یه جای دیگه، همه وظیفه CPU هست. درواقع همه چیز باید در کنترل CPU باشه. CPU مثل یه آشپز ماست که کارا رو انجام میده. اما شما مثلاً کتاب الکترونیکی دارین. این کتاب الکترونیکی که تو هوا ذخیره نمیشه که. قاعداً باید یه قطعه باشه که نگهداریشون کنه. یا مثلاً ویندوز داریم که توش یه سری برنامه هست که همه این ویندوز و برنامه هم باید یه جا نگهداری شن. ما توی کامپیوتر یه جای خیلی بزرگ داریم که همه چیز رو توش نگه می داریم. یه انباری. این انباری ما توی کامپیوتر، اسمش هارد هست. (البته می تونه SSD هم باشه. SSD هم یه نوع انباری دیگس. توجه کنین که SSD هم یه نوع درایو ذخیره سازیه و هارد نیست! پس کلمه «هارد SSD» کلمه ای اشتباهه)

خب مثلاً وقتی شما دکمه روشن شدن رو می زنین و ویندوز بالا میاد، باید ویندوز در دسترس CPU که مغز کامپیوتره قرار بگیره. اما انباری ما از CPU خیلی دوره. CPU هر بار بخواد یه چیز از انباری برداره، خیلی زمان بره و کنده!

اینجا اومدن یه مکانی نزدیک تر و کوچکتر به CPU ساختن به نام RAM. جایی که هر چی ما باهاش کار داریم بیاد اونجا بشینه که دسترسی CPU بهش ساده تر باشه. درواقع وقتی روی یه برنامه کلیک می کنین، اون برنامه پا میشه و میاد توی RAM میشینه که CPU بتونه راحت تر بهش دسترسی داشته باشه.

اطلاعات برنامه های در حال اجرا توی رم هست که نزدیک CPU باشه. رم یه حافظه کوچکتر و محدودیه. پس همیشه هزار تا برنامه با هم باز کرد. چون نسبت به هارد خیلی زودتر پر میشه.

پس درواقع یه انباری (SSD یا HDD (هارد)) داریم. هر وقت بخوایم برنامه ای رو اجرا کنیم میاد میشینه توی RAM که در دسترس مغز CPU باشه.

فرض کنین که یه برنامه دارین که هر یه ثانیه یکبار یکدونه صدای سیستم رو زیاد می کنه. دستوراتش اینه:

ا- یک ثانیه صبر کن

۲- یه دونه صدا رو زیاد کن

۳- کارهای بالا رو تا وقتی صدا ۱۰۰ نشده انجام بده.

خب گفتیم اطلاعات برنامه میاد میشینه توی رم درسته؟ پس این دستورات هم میان می‌شینن داخل رم. یعنی CPU میره داخل رم میگه خب دستور اول چیه؟ یه ثانیه صبر کن. باشه صبر می‌کنم. دستور دوم چیه؟ یه دونه صدا رو زیاد کن. باشه می‌کنم! دستور سوم چیه؟ چک کنم ببینم ۱۰۰ شده یا نشده. اگر نشده دوباره برگردم به دستور ۱. دوباره میره بالا رو بخونه. صد بار این رو می‌خونه. خب اما رم یکم از CPU دوره. درسته نسبت به هارد خیلی نزدیک‌تره. اما بازم برای کارهای تکراری مثل این، من هی باید برم سمت رم هم بخونم. کار تکراریه. کامپیوتر که عقل نداره بفهمه یه کار باید ۱۰۰ بار انجام شه و توی ذهن حفظش کنه و هی نخواد بره سمت رم. کامپیوتر صرفاً یه رباته که یه سری دستوری که بهش دادیم، انجام میده. چیزی رو حفظ نمی‌کنه. پس ۱۰۰ بار هی باید رم رو بخونه. فاصله CPU و RAM یکمی زیاده هنوز. ۱۰۰ بار هم باید بره هی سراغ رم. خب طول می‌کشه دیگه! اینجا اومدن گفتن یه حافظه بسیار بسیار کوچیک توی CPU قرار میدیم که این چیزایی که تکرارین و یا چیزایی که CPU احتمالاً در دستور بعدی بهش نیاز داره رو اونجا می‌ذاریم که نخواد بره تا رم. دست کنه کنارش و برشون داره. یعنی صرفاً یکی دو دستور جلویی و چیزایی که برای اون دستورا بهشون نیاز داریم رو اونجا می‌ذاریم که CPU نخواد تا رم بره. مثلاً کامپیوتر میگه برای دستور بعدی احتمالاً نیاز به دونستن مقدار صدای سیستم داره. پس من قبلش میرم اون رو از رم میارم که زودتر بتونه کار رو انجام بده. یه حافظه خیلی خیلی کوچیکه. در حد کیلوبایت و مگابایت!

## • تبدیل مبنا (Base Conversion)

همونطور که پول می‌تونه واحدها و مبنای مختلف مثل ریال، دلار، یورو، دینار و... داشته باشه، عدد هم می‌تونه یه سری مبنا داشته باشه. ببینین اعدادی که ما استفاده می‌کنیم، مبناشون ۱۰ هست. یعنی ۱۰ رقم داریم که اعداد رو میسازین. رقم ۰ تا ۹ (۱۰ تا) اما اعداد می‌تونن به حالت مبنای دیگه هم نمایش داده شن. مثلاً کامپیوتر با مبنای ۲ کار می‌کنه. یعنی ۰ و ۱ (۲ تا رقم). یعنی صرفاً تمام اعداد رو با ۰ و ۱ نمایش میدیم. مثلاً عدد «یازده» به مبنای ۲ (دودویی-باینری) میشه «۱۰۱۱»<sup>۵</sup> مبنای خیلی مهم که خوبه بلدش باشین:

base	اسم فارسی	range
2 (Binary)	دودویی (باینری)	0 - 1
8 (Octal)	هشت‌هشتی	0 - 7
10 (Decimal)	ده‌دهی	0 - 9

۵ به هرکدوم از این صفر یا یک‌ها، می‌گن «بیت (bit)» می‌گن.

16 (Hexadecimal)	مبنای شونزده	0 - 9, a - f
------------------	--------------	--------------

برای مبنای ۱۶، گفتن که ما خب باید ۰ تا ۱۵ بریم دیگه. اما از کجا معلوم مثلاً ۱۲ که می‌نویسیم، منظورمون یه رقم که معنای ۱۲ داره هست یا منظورمون ۱ و ۲ جدا هست؟ برای همین از اعداد انگلیسی کمک گرفتن. گفتن که:

a = 10

b = 11

c = 12

d = 13

e = 14

f = 15

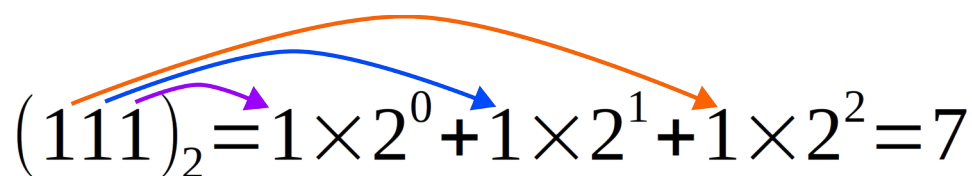
ولی خب حواسمون هست که مثلاً c معنای ۱۲ میده.

نکته: هر ۲ تا کرکتر هگزادسیمال (که هرکدوم ۱۶ حالت دارن)، یه بایت رو می‌سازن:

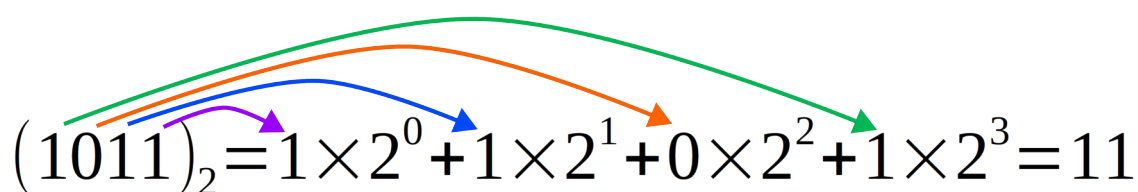
$$16 \times 16 = 2^4 \times 2^4 = 2^8 \rightarrow 8 \text{ bits} = 1 \text{ Bytes}$$

حالا شاید بگین چه‌طور میشه این اعداد رو به هم تبدیل کرد؟

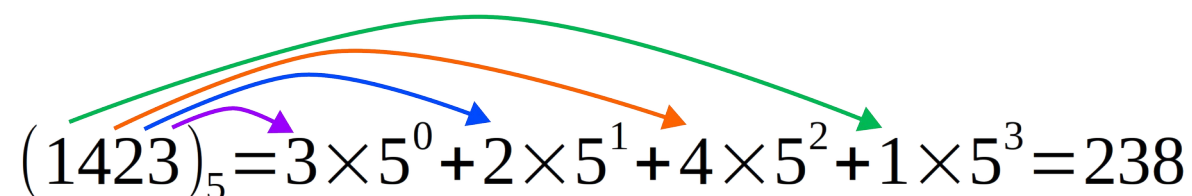
اگر بخوایم یه عددی رو از یه مبنا (مبنا رو معمولاً پایین سمت راست عدد می‌نویسن) برسونیم به یه مبنای دیگه، از راه زیر استفاده می‌کنیم:



$$(111)_2 = 1 \times 2^0 + 1 \times 2^1 + 1 \times 2^2 = 7$$



$$(1011)_2 = 1 \times 2^0 + 1 \times 2^1 + 0 \times 2^2 + 1 \times 2^3 = 11$$



$$(1423)_5 = 3 \times 5^0 + 2 \times 5^1 + 4 \times 5^2 + 1 \times 5^3 = 238$$

از سمت راست شروع می‌کنیم و اون رقم رو ضربدر مبنا به توان صفر می‌کنیم. بعدش رقم بعدی رو اینبار ضربدر مبنا به توان ۱ می‌کنیم. بعدش توان ۲ و... و همینطور ادامه میدیم.

اما اگر بخوایم یه عددی رو از دسیمال ببریم به یه مبنای دیگه، هی باید دونه دونه تقسیم کنیم و باقی‌مونده‌ها رو از آخرین تقسیم به سمت اولین تقسیم، از سمت چپ شروع کنیم بنویسیم تا عدد ساخته شه:

$$\begin{array}{r} 25 \\ \underline{-24} \\ 1 \end{array} \quad \begin{array}{r} 2 \\ \underline{-12} \\ 0 \end{array} \quad \begin{array}{r} 2 \\ \underline{-6} \\ 0 \end{array} \quad \begin{array}{r} 2 \\ \underline{-2} \\ 0 \end{array} \quad \begin{array}{r} 2 \\ \underline{-1} \\ 1 \end{array} \quad \begin{array}{r} 2 \\ \underline{-0} \\ 2 \end{array}$$

$$(25)_{10} = (11001)_2$$

$$\begin{array}{r} 87 \\ \underline{-86} \\ 1 \end{array} \quad \begin{array}{r} 2 \\ \underline{-42} \\ 1 \end{array} \quad \begin{array}{r} 2 \\ \underline{-20} \\ 0 \end{array} \quad \begin{array}{r} 2 \\ \underline{-16} \\ 0 \end{array} \quad \begin{array}{r} 2 \\ \underline{-10} \\ 1 \end{array} \quad \begin{array}{r} 2 \\ \underline{-6} \\ 0 \end{array} \quad \begin{array}{r} 2 \\ \underline{-2} \\ 0 \end{array} \quad \begin{array}{r} 2 \\ \underline{-1} \\ 1 \end{array}$$

$$87_{(10)} = (1010111)_2$$

**راه دوم:** (سادگیش موقع تبدیل به باینری هست)

فرض کنیم می‌خواهیم ۲۵ رو به باینری بنویسیم:

میگیم که بزرگترین توان ۲ در ۲۵ چیه؟ خب معلومه ۱۶ بزرگترین توانی هست که در عدد ۲۵ وجود داره. پس ۱ به نشانه ۱۶ می‌گذاریم:

1

از ۲۵، ۱۶ تا کم می‌کنیم. میشه چقدر؟ ۹. خب حالا توی ۹ آیا ۸ (بزرگترین توان بعد ۱۶) وجود داره؟ بله. پس ۱ هم به نشانه ۸ می‌گذاریم:

11

از ۹، ۸ تا کم می‌کنیم. میشه ۱. بعدش می‌گیم بعد ۸ بزرگترین توان چیه؟ ۴. آیا ۴ توی ۱ وجود داره؟ نه! پس جای ۴، صفر می‌گذاریم:

110

خب همینکارو همینطور برای توان‌های کوچکتر ۲ انجام میدیم.. آیا ۲ در ۱ وجود داره؟ نه. پس برای این هم صفر می‌گذاریم:

1100

آیا ۱ (۲ به توان صفر) درون ۱ وجود داره توش هست؟ بله پس یکدونه ۱ به نشانه وجود ۱ می‌گذاریم:

11001

تمام! به همین سادگی! یکم شاید اولش سخت بیاد ولی یکم تمرین کنین به سادگی می‌تونین سریع بگین ۱۶ داره. ۸ داره. ۴ نداره. ۲ نداره. ۱ داره. پس ساخته شد!

از سایت زیر هم می‌تونین استفاده کنین که مبناها رو به هم تبدیل کنه:

<https://www.rapidtables.com/convert/number/base-converter.html>

توی پایتون هم می‌تونین توابع `int` و `bin` و `hex` استفاده کنین. (اگر نمی‌دونین الان چی هستن، بعد خوندن `string` ها و تابع، برگردین و بخونیدشون):

```
print(int("21", 3))
```

میگه که یه استرینگی که در مبنا ۳ هست رو تبدیل به یه `integer` کن.

```
print(bin(7))
```

یه عدد دسیمال رو تبدیل به باینری کن.

```
print(hex(21))
```

یه عدد دسیمال رو تبدیل به `hex` کن.

- اعداد اعشاری چه جور ذخیره میشن؟

+ سخته نمی‌خواد بدونین. فقط بدونین ذخیره اعشار برای کامپیوتر سخته و برای همین چون اعشار رو نمی‌تونه تا بی‌نهایت ذخیره کنه، یه سری خطاها به وجود میاد. مثلاً:

```
print(3*0.1 == 0.3)
```

آیا ۳ تا ۰.۱ مساوی هست با ۰.۳؟ می‌بینین چاپ می‌کنه نه!

## • معرفی وبسایت Quera<sup>۷</sup>

ببینین یه وبسایتی هست که پر از سواله و شما می‌تونین برین توش و تمرین کنین. تمرین مهم‌ترین قسمت برنامه‌نویسی هست. اگر می‌خواین واقعاً یاد بگیرین، باید مداوم تمرین کنین. حتی هر روز! توی این سایت ثبت نام کنین. بعدش که وارد شدین، توی منوی بالا، یه قسمت هست به نام بانک سوالات؛ روی اون کلیک کنین.

۶ اگر خیلی علاقه دارین: «IEEE 754» رو بخونین.

7 <https://quera.org>

حالا شما می‌تونین از سمت راست چیزا رو فیلتر کنین که چه سؤالاتی بهتون نمایش داده شه. مثلاً دسته سؤالات «دانشگاهی»

یا حتی می‌تونین از برچسب فیلتر کنین بگین سؤال رشته (string) می‌خوام.

تعداد حل رو هم زده که مثلاً از ۵۰۰ تلاشی که افراد کردن، ۴۰۰ نفر تونستن حل کنن. ابتدا که شروع به برنامه‌نویسی می‌کنین از اونایی که تعداد حل زیاد دارن (مثلاً چند هزار نفر حل کردن) شروع کنین و هی بیاین کمتر که سخت‌تر شه.

شما حل می‌کنین و فایل حلتون رو برای وبسایت ارسال می‌کنین. زیر هر سؤال شما می‌تونین فایل رو ارسال کنین. البته زبون برنامه‌نویسی رو هم باید مشخص کنین. شما Python 3.8 رو انتخاب کنین. بعد ارسال، یه یکی دو ثانیه بعدش صفحه رو ریفرش کنین تا بهتون امتیاز بده که چه نمره‌ای رو کسب کردین. اگر کامل شدین که ایول. اگر نشدین، روی عدد امتیازتون بزنین و ببینین مشکلات چی هستن:

- **Wrong Answer**

کد شما جواب اشتباهی میده. سعی کنین کدتون رو بررسی کنین و ببینین مشکل چیه؟ نمونه تست‌هایی که خود سؤال داده رو جواب درست میدین یا نه؟

نکته! چون بررسی جواب شما به صورت خودکار صورت می‌گیره، عیناً باید شبیه چیزی که گفته مقادیر رو چاپ کنین و ورودی بگیرین. موقع ورودی گرفتن، پیامی چاپ نکنین. یعنی اینطوری ننویسین:

```
num1 = int(input("Please enter a number: "))
```

چون کدتون غلط میشه. هیچ چیزی نباید اضافه چاپ کنین. چون به صورت خودکار داره چک میشه و فکر می‌کنه که چیزی چاپ کردین و فکر می‌کنه این جوابه. بلکه جواب نیست. صرفاً باید ورودی‌ها رو به صورت عادی بگیرین. یعنی:

```
num1 = int(input())
```

هیچ چیز اضافه‌ای رو چاپ نکنین. فرمت چاپ کردن عیناً شبیه سؤال باشه. یعنی اگر سؤال نمونه ورودی رو اینطوری داده:

1 2 3 4

شما هم باید همینطوری چاپ کنین. اگر مثلاً به صورت‌های زیر چاپ کنین همش اشتباهه و Wrong Answer می‌خورین:

1  
2  
3  
4

The answer is: 1 2 3 4



حتی یک نقطه یا فاصله اضافه چاپ کردن باعث غلط شدن جواب میشه. پس عیناً شبیه نمونه خروجی چاپ کنین.

- **Runtime Error**

کدتون به ارور خورده وسط کار. شاید مثلاً ورودی‌ها رو برعکس گرفتین و ارور خورده. مثلاً اگر اول name رو میدن و بعد age، باید عیناً به ترتیب ورودی بگیرین. اگر برعکس مثل زیر ورودی بگیرین، کدتون به ارور بر می‌خوره:

```
age = int(input())
name = input()
```

چون استرینگ name رو میدن ولی شما اشتباهی توی age می‌ذارید و تبدیل به int که می‌کنی، می‌گه من استرینگ کرکتری رو که نمی‌تونم به int تبدیل کنم! پس ارور می‌خوره. خیلی وقتاً این ارور رو زمانی می‌گیرین که حالتای گوشه و کنار رو بررسی نکردین. یا مثلاً تقسیم بر صفر داشتن تو کد. خلاصه هر نوع اروری! توی این دسته قرار می‌گیره!

- **Time Limit exceeded**

یا توی حلقه بی‌نهایت گیر کردین و کدتون به جا توی while یا چیزی گیر کرده و بیرون نمیاد (احتمالاً شرط while اشتباه انتخاب کردین) یا اینقدر کدتون رو بد نوشتین که اینقدر کار اضافه و بی‌هوده داره انجام میده که از مدت زمان تعیین شده توسط سؤال عبور کرده و Time limit خورדיن. اگر مطمئنن کدتون مشکل نداره، شاید نیاز باشه کدتون رو بهینه کنین (با چیزایی مثل break) که کار اضافه انجام نده. یا کلاً راهی که به ذهنتون رسیده اونقدر بد و کند هست که باید راهتون رو کلاً عوض کنین.

- **Memory Limit exceeded**

کدتون اونقدر فضا گرفته که از فضای محدودی که سؤال تعیین کرده عبور کردین. مثلاً ۱۰۰ هزار بار ۱۰۰ هزار لیست یا string های مختلف رو ساختین. خب معلومه مموری تموم میشه دیگه! یا توی recursive به این مشکل بر می‌خورین. (اگر نمی‌دونین recursive چیه، بعداً باهاش آشنا میشین)  
- راه حل چیه؟

+ سعی کنین کدتون رو درست‌تر بنویسین و مثلاً توی while خیلی طولانی، هی متغیر اضافه با حجم خیلی خیلی بالا (مثل یه لیست با ۱۰ میلیون عضو) نسازین. ولی هیچ‌وقت به واسطه راه درست و متغیر درست تعیین کردن و چهارتا متغیر ساده و یا حتی یه لیست ۱۰ هزار تایی به این ارور بر نمی‌خورین!

برای کوئرا فرقی نداره که یه چیز رو اول یا آخر چاپ کنین. صرفاً نیازه که در آخر همه چیز عیناً درست و به ترتیب درست چاپ شده باشه.

یعنی فرقی نداره که حتی بین ورودی گرفتن‌ها چیزی رو چاپ کنین یا بعدش. چون کوئرا خروجی‌ها رو می‌ریزه داخل یه فایل جدا و در آخر چک می‌کنه همه چیز درست و با ترتیب درست هست یا نه. یعنی مثل شما که خروجیتون همونجایی که ورودی میدین نیست. پس فرقی نداره چه اول چاپ کنین چه آخر.

### • آشنایی با تاریخچه زبون‌های برنامه‌نویسی<sup>۸</sup>

ببینین اول کامپیوتر چیه؟ یه دستگاهی که با دستوراتی که بهش میدیم، عمل میکنه. مثلاً روی فایرفاکس کلیک می‌کنیم. عملاً داریم بهش میگیم که فایرفاکس رو برای من باز کن. دکمه بلندی صدا رو نمایش میدیم. عملاً بهش میگیم که صدا رو بلند کن. در کامپیوتر هم دقیقاً هم همین اتفاق میوفته. تا دکمه بلندی صدا رو زدیم، یه پیام میره به سمت کامپیوتر و کامپیوتر میگه عه دکمه‌ی بلندی صدا رو زدی؟ باشه! صدا رو برات بلند میکنم. عملاً ما دستور میدیم به کامپیوتر.

اما کامپیوتر که نمیفهمه دکمه بلندی صدا یعنی چی؟ اصلاً کامپیوتر abcd رو نمیفهمه. کامپیوتر فقط یک چیز میفهمه. بودن سیگنال یا نبودن سیگنال. صفر یا یک. برقراری جریان یا نبود جریان الکتریکی. خب اما برای ما خیلی سخته که بخوایم به صورت صفر یا یک دستور بدیم. بگیم الان کامپیوتر تو جریان عبور نده. الان بده. خیلی سخته! اینجا افرادی اومدن یه سری راهکار دادن به ما. گفتن چی؟ گفتن ما یه سری برنامه میسازیم، که این رو براتون ساده کنه. به جای اینکه تو صفر و یک دستور بدی، بیا من کارو برات ساده میکنم. تو بیا کلیک کن و صدا رو با کشیدن موس زیاد کن. من برات صدا رو زیاد میکنم. این شد که کامپیوترها پرکاربرد شدن. چون به شدت کار با صفر و یک سخته. ولی یه عده اومدن این کار رو برای ما ساده کردن. عملاً این ویندوزی که شما باهاش کار میکنین، این مرورگر مثل فایرفاکس یا کروم که کار میکنین، آفیس و ورد و پاورپوینت، همه و همه نوعی برنامه هستن که انجام کار رو براتون ساده کردن. به جای اینکه شما صفر و یک دستور بدی، برنامه میگه کلیک کن من برات بقیشو انجام میدم. پس این قصه‌ی برنامه‌ها و جایگاه برنامه‌نویسی.

در برنامه‌نویسی ما میخوام فلسفه‌ی پشت این برنامه‌ها رو انجام بدیم. یعنی ما با ساخت یه برنامه، بین یه برنامه و اون سخت‌افزار کامپیوتر قرار میگیریم و مثلاً میگیم که اگر کسی که داشت این برنامه رو اجرا میکرد، دکمه رنگی کردن متن رو زد، حالا ما با کمک خود زبون برنامه‌نویسی به سخت‌افزار دستور میدیم، که فلان کار رو برات انجام بده. یعنی ما مثل واسطه قرار می‌گیریم. یعنی ما با ساخت یه برنامه، امکان تبادل کاربر با کامپیوتر رو برقرار می‌کنیم. چیزایی میسازیم که یه کاربر بتونه استفاده کنه و کاراشو پیش ببره.

۸ ترجیحاً اینو بعد فهمیدن نحوه کار با متغیرها بخونین.

درواقع مثلاً می‌گیم اینجا یه دکمه قرار بگیره. کاربر اگر روش کلیک کرد، کامپیوتر باید یه سری مراحل رو طی کنه که منجر به عمل‌کردی که کاربر می‌خواد میشه. یکمی سخت شد نه؟ بذارین بریم توی مثال‌های برنامه‌نویسی، خیلی ساده‌تر میشه.

خب گفتیم کامپیوتر فقط ۰ و ۱ می‌فهمه درسته؟ پس اگر ما می‌خوایم بین کاربر و کامپیوتر قرار بگیریم، باید با صفر و یک یه محیطی رو بسازیم که برای کاربر قشنگ باشه و با صفر و یک به کامپیوتر دستوراتی که کاربر می‌گه رو بفهمونیم.

خیلی این کار سخته. باید به دونه دونه پیکسل‌های صفحه نمایش بگیریم تو فلان جا روشن شو، تو فلان جا فلان جریان‌ها رو بفرست که رنگ عدد قرمز شه که مانیتور عددها و شکلا رو نشون بده و از اون طرف محاسباتی که کاربر می‌خواد رو فلان جور به سخت‌افزار بفهمونی. این کار واقعاً سخته!

برای همین یه سری چیزا تولید شد که ارتباط ما با سخت‌افزار ساده شه. کامپیوتر رو لایه لایه کردن. یه سریا گفتن که ما متخصصیم و می‌فهمیم چطور باید با ۰ و ۱ و چیزای عجیب با سخت‌افزار صحبت کنیم. ما یه چیزی درست می‌کنیم که افراد دیگه که بلد نیستن بتونن ارتباط رو داشته باشن. بهتون می‌گیم که اگر فلان صفر و یک رو پشت هم بچینی، کامپیوتر برات جمع رو انجام میده. اما باز صفر و یک سخت بود. اوکی حالا من فهمیدم چطور به کامپیوتر بگم جمع انجام بده. ضرب انجام بده و چیزای دیگه. اما خیلی سخته بخوام صفر و یک بنویسیم.

بازم متخصصا گفتن ایرادی نداره. ما یکم این رو براتون خواناتر می‌کنیم. به جای اینکه شما صفر و یک بنویسی، صرفاً بنویس `add`، و ما خودمون تبدیلیش می‌کنیم به صفر و یک.

اینجا خیلی خیلی کار ساده‌تر شد. جایی بود که اسمبلی به وجود اومد و ما نیاز نبود صفر و یک کد بنویسیم و با کامپیوتر ارتباط برقرار کنیم. من می‌گفتم `sub` و دو چیز رو از هم کم می‌کرد.

اما هنوزم کار خیلی پیچیده بود. درسته کلمات ساده شده بود و انگلیسی و خوانا ولی مشکل این بود که بازم هنوز من عملاً داشتم با CPU صحبت می‌کردم. می‌گفتم CPU فلان عدد که فلان جای رم نوشته شده رو بردار و بیار توی خودت. حالا عددی که آوردی رو با فلان عدد توی فلان جای رم جمع بزن و باز پیش خودت نگه دار. حالا برو بذارش توی فلان آدرس رم.

خیلی سخته این کار. قشنگ دارم بهش دونه دونه سخت‌افزاری می‌گم توی خونه سوم رم فلان چیز نوشته شده برو بیارش. این خیلی سخته. اشتباه زیاد توش پیش میاد. ممکنه من آدرس خونه‌ها رو اشتباه بدم و کلی چیز دیگه.

برای همین یه سری زبون سطح بالاتر به وجود اومدن و گفتن آقا ما یه زبونی برات نوشتیم که کارت ساده شه. نخواستی به CPU دستور بدی. تو به ما دستور بده، ما خودمون بلدیم چطور با CPU صحبت کنیم.

زبونایی مثل C به وجود اومدن که قرار شد کار رو برای ما ساده کنن.

من نیاز نبود بگم برو از فلان جای رم یه عدد نوشته شده برش دار بیار با فلان چیز جمع کن. من صرفاً می‌گم یه خونه توی رم برام رزو کن و اسمشو بذار X و فلان عدد رو بریز توش. نه نیازه بگم آدرسش

کجاست نه هیچی! خود زبون برنامه نویسی حواسش هست. هر وقت هم با اون خونه کار داشته باشم، بهش میگم X و خود زبون برنامه نویسی حواسش هست آدرسش کجا بود. یه مثال:

- ۱- یه جایی توی رم برام در نظر بگیر اسمش بذار X و ۵ که یه عدد صحیح هست رو بریز داخلش
- ۲- یه جایی توی رم برام در نظر بگیر و اسمش رو بذار Y و ۶ که یه عدد صحیح هست رو بریز داخلش
- ۳- ۵ و ۶ رو برام جمع کن و نتیجتش که یه عدد صحیح هست رو نشون بده.

مثلاً:

```
int x = 5;
int y = 6;
printf("sum is: %d", x + y);
```

```
int x = 5;
int y = 6;
printf("%d", x + y);
```

output: **sum is: 11**

توضیح: int مخفف کلمه integer هست.

یه عدد صحیح X رو در نظر بگیر که برابر ۵ هست.

یه عدد صحیح Y در نظر بگیر که برابر ۶ هست.

پرینت کن (چاپ کن) چیزی که توی quotation هست رو.

حالا توی quotation به زبون برنامه نویسی میگم که چاپ کن روی صفحه که **sum is: %d** و **%d** یه علامت خاص هست که به زبون برنامه نویسی بگم که اینجا قراره یه عدد صحیح چاپ کنی. حالا اون عدد صحیح کجاست؟ بعد کاما نوشتمش. اون عدد صحیح جمع X و Y هست!

اینجا باز یه سری زبون سطح بالاتر مثل Python به وجود اومدن. پایتون میگه بابا تو کاریت نباشه! من خودم همه چیز حواسم هست. نیاز نیست بگی یک عدد صحیح در نظر بگیر که برابر ۵ هست. وقتی بگی یه X در نظر بگیر که ۵ هست، من خودم میفهمم عدد صحیح. همه اینا خودم حواسم هست! نگران نباش.

مثال جمع دو عدد در پایتون:

```
x = 5
y = 6
print(f"sum is: {x + y}")
```

دیگه نیاز نیست بگم اینجا یه عدد صحیح قراره چاپ شه. صرفاً توی کروش می نویسم X. خودش می فهمه.

خب بذارین یه سؤال ازتون بپرسم. چه زبون برنامه نویسی بهتره؟  
 - خب معلومه دیگه! آخری! هرچی سطح بالاتر، بهتر! کار باهش راحت تره!  
 + قبول دارم کار باهش راحت تره، اما یه مشکل داره! هرچی از سطح صحبت با سخت افزار دورتر شیم، کندتر میشه. چون باید این چیزا اول توسط زبون برنامه نویسی دونه دونه تبدیل شن به ۰ و ۱ و این تبدیل، به اندازه کدی که از اول با ۰ و ۱ ساخته شه، سریع نیست!  
 چون زبون های برنامه نویسی یه چیز جنرال و کلی هستن و مشخصاً تبدیلاشون هم یه چیز کلی هست. شما توی زبونی مثل اسمبلی، دستتون خیلی بازه. قشنگ می تونین با CPU صحبت کنین و حرف بزنین و قشنگ یه سری میون برا رو بزنین که برنامه خیلی سریع شه.  
 - خب کسی که زبون سطح بالا رو می نویسه، نمیتونه خودش حواسش به کد باشه که بتونه موقع تبدیل کد به کد ماشین (۰ و ۱)، همین چیزا رو هم میون بر کنه و سریع کنه؟!  
 + می تونه تا حدی بهینه کنه کد رو ولی نه اندازه ای که یه فرد حرفه ای همون کد رو به زبان اسمبلی بنویسیتش!

چیزی به نام بهترین نداریم. مثل اینکه کامیون بهتره یا خب حالا برگردیم به اینکه برنامه نویسی کجای رشته کامپیوتر قرار داره؟  
 برنامه نویسی درواقع ابزاره. شما مثلاً می خوای یه سیستم عامل طراحی کنی، میری برنامه نویسی سطح پایین انجام میدی. زبونایی مثل Rust و C. اما یه سوال! تا وقتی ندونی کامپیوتر چه جور کار می کنه، می تونی ویندوز بنویسی؟ نه! پس درواقع شما صرفاً داری علمی که از نحوه کار کامپیوتر داری رو به زبونی برنامه نویسی پیاده سازی می کنی! درواقع دانش اصلی، اون علمی هست که کامپیوتر چه جور کار می کنه؟

## رعایت خوانایی:<sup>۹</sup>

یکی از چیزایی که برای خوانایی مهمه رعایت کنین، رعایت نحوه نوشتار انگلیسی هست:

**Correct:** Others teach python hard, I don't. Read this document.<sup>10</sup>

**wrong:** Others teach python hard,I don't.Read this document.

**Correct:** WHO (World Health Organization)<sup>11</sup>

**wrong:** WHO(World Health Organization)

**wrong:** WHO( World Health Organization )

۹ بعداً توی نوشتن کد خیلی بهتون کمک می کنه.  
 ۱۰ کاما (و تمام کرکتهای دیگه جز پرانتز که شامل «نقطه»، «دو نقطه»، «کویشن» و... به کرکتر قبل می چسبن و از بعدی یه دونه فاصله دارن.  
 ۱۱ پرانتز از دو طرف یه فاصله داره ولی داخلش فاصله نداره.

## پایتون وارد می‌شود!

ما باید به زبون برنامه‌نویسی دستور بدیم. بگیم فلان کار کن. حالا فلان کار کن. درواقع زبون برنامه‌نویسی خودش یه برنامه هست (Interpreter پایتون) که میاد خط به خط کدهای درون فایل رو می‌خونه و دستورات رو انجام میده. پس ما باید خط به خط بهش دستور بدیم.

### 1. print()

فرض کنین ما بخوایم یه چیزی رو روی صفحه نمایش چاپ کنه. پایتون یه دستوری داره که میشه ازش استفاده کرد. اسمش رو گذاشته «print» یعنی چاپ کن.

- چیه چاپ کن؟

+ چیزی که توی پرانتز بهت می‌گم رو.

مثلاً می‌خوایم بگیم ۵ رو چاپ کن:

```
print(5)  
output:۱۲ 5
```

یا مثلاً بهش می‌گیم:

```
print(5 + 6)  
output: 11
```

Wrong:<sup>۱۳</sup>

```
print( 5 + 6 )
```

پرانتز هم به print و هم به عضو اول بعدش می‌چسبه. همچنین بعد ۶ باید پرانتز بیاد. فاصله‌ای نباید بگذارین.

ما می‌تونیم هر خط یه دستور جدید بنویسیم. درواقع یه پایتون میاد توی هر خط دستور رو می‌خونه و اجرا می‌کنه. مثلاً اگر توی خط اول بگیم ۵ رو چاپ کن و توی خط دوم بگیم ۵ + ۶ رو چاپ کن، به ترتیب ۵ و ۱۱ رو توی خروجی نشون میده:

```
print(5)  
print(5 + 6)  
output:  
5  
11
```

مثال بیشتر:

```
print(2 * 6)  
output: 12
```

<sup>۱۲</sup> مقداری که روی صفحه نمایش چاپ میشه. درواقع چیزی که کامپیوتر به ما خروجی (output) میده.  
<sup>۱۳</sup> این کد نیست! صرفاً یعنی مثال پایینی اشتباهه.

نکته: از همین حالا سعی کنین تمیز بنویسین. یعنی مثلاً:

correct:

```
print(5 + 6)
```

wrong:

```
print(5+6)
```

توضیح: بین چیزا و علامتا یه فاصله باشه که تمیز تر باشه.<sup>۱۴</sup>  
یه خرده تمرین کنین و علامتای مختلف پایتون رو یاد بگیرین:

operator	Name	توضیحات	مثال	حاصل
+	Addition (جمع)	دو چیز رو با هم جمع می کنه	2 + 3	5
-	Subtraction (تفریق)	دو چیز رو از هم کم می کنه	3 - 2	1
*	Multiplication (ضرب)	ضرب دو چیز	3 * 2	6
/	Division (تقسیم)	تقسیم دو چیز	3 / 2	1.5
//	Floor division (تقسیم صحیح)	قسمت اعشاری رو می ریزه دور و فقط قسمت صحیح رو نمایش میده	3 // 2	1
%	Modulus (باقی مونده) (mod)	تقسیم دومی بر اولی رو انجام میده و باقی مونده تقسیم هرچی بود رو نمایش میده. (باقی مونده و خارج قسمت زمان دبستان رو یادتونه؟!)	3 % 2	1
**	Exponentiation (توان)	اولی رو به توان دومی می رسونه	3 ** 2	9

مثال:

```
print(3 ** 3)
```

output: 27

مثال:

```
print(3 ** 2 - 5 * 6)
```

output: -21

چجوری؟

اولویتای ریاضی زمان دبستان رو یادتون رفته؟!

الف) اول پرانتز

ب) دوم توان

<sup>۱۴</sup> اصول تمیز نویسی رو می تونین از <https://peps.python.org/pep-0008> بخونین که البته فعلاً براتون پیشرفتن. نگران نباشین! توی مسیر یادگیری خودم بهتون میگویم!

ج) سوم ضرب و تقسیم (اگر صرفاً ضرب و تقسیم بود، از چپ به راست باید بریم. اولویت‌هاشون با هم برابره!)

د) چهارم جمع و تفریق (اگر صرفاً ضرب و تقسیم بود، از چپ به راست باید بریم. اولویت‌هاشون با هم برابره!)

مثلاً سؤال زیر خیلی معروفه که جوابش چی میشه؟

$$6 / 2 (1 + 2)$$

بیایم با هم بررسی کنیم:

اول داخل پرانتز رو حساب می‌کنیم. میشه ۳. پس ساده شد به:

$$6 / 2 * 3$$

بعدش می‌گیم خب فقط شامل تقسیم و ضربه. پس از چپ به راست پیش میریم. اول ۶ رو تقسیم بر ۲ می‌کنیم. حاصلش میشه ۳.

حالا ۳ رو ضربدر ۳ می‌کنیم. میشه ۹! به همین سادگی و خوشمزگی! اینقدر نیاز نبود توی توییترو اینستا، قانون ریاضی اختراع کنین؛

خب تمرین کردین؟ حالا بیایم غیر ریاضی یکم کارهای دیگه کنیم.

اگر بخواین یه چیزو عیناً پرینت کنین، توی علامت کوتیشن «'''» یا دبل کوتیشن «"""» قرارش بدین. (ترجیحاً کوتیشن) مثلاً من می‌خوام یه جمله رو چاپ کنم:

```
print('Hi! How are you?')
```

output:

```
Hi! How are you?
```

مثلاً بخوام یه سؤال جواب رو چاپ کنم:

```
print('Hi! How are you?')
```

```
print('Great! How about you?')
```

output:

```
Hi! How are you?
```

```
Great! How about you?
```

دیدین؟ پایتون خط به خط کدو می‌خونه و کار رو واسم انجام میده. خط اول می‌خونه می‌گه عه باید فلان جمله چاپ کنم. چاپ می‌کنه و میره خط بعد و می‌بینه عه بازم باید یه چیز دیگه چاپ کنم. چاپ می‌کنه و می‌گه خب عه دیگه خطی نیست؟ بعدش تموم میشه.

تذکر! همیشه حواستون باشه که وقتی می‌خوانین از یه متن استفاده کنین، باید اونو توی کوتیشن بذارین. وگرنه ارور میده. مثلاً:

```
print(hello)
```

output:

```
File "<string>", line 1, in <module>
```



NameError: name 'hello' is not defined

```
1 print(hello)
```

Code

```
>>> %Run -c $EDITOR_CONTENT
```

```
Traceback (most recent call last):  
  File "<string>", line 1, in <module>  
NameError: name 'hello' is not defined
```

Output

همینطور که می‌بینیم، نوشته که در خط یک به مشکل برخوردیم. نوشته که `hello` تعریف نشده است. درواقع می‌گه خب یعنی چی؟! منظور متنه؟ اگر منظور متنه خب باید می‌گذاشتیش توی کوتیشن.

**نکته!** از همین حالا سعی کنین خوندن ارور و یافتن مشکل رو یاد بگیرین. توی ارور بهتون می‌گه مشکل از کجاست. یا همیشه مشکل از اونجاست یا یه خط بالاش.

تمرین ۱-ا:

سعی کنین با استفاده از دستور پرینت، یه مستطیل بکشین!

راهنمایی:

یادتونه گفتیم که توی کوتیشن هرچی بگذاریم، همونو عیناً بدون هیچ تغییری چاپ می‌کنه؟ خب سعی کنین با علامتا و کرکترهایی که روی کیبوردتون می‌بینین، یه مستطیل بکشین.

پاسخ:

```
print('*****')  
print('*      *')  
print('*      *')  
print('*      *')  
print('*      *')  
print('*      *')  
print('*      *')  
print('*****')
```

دیدین؟! صرفاً با دستور پرینت یه مستطیل کشیدیم. گفتم اول یه سری ستاره چاپ کن. بعدش خط بعدیش بیا اول یه ستاره، بعد یه یه سری فاصله و بعد یه ستاره دیگه چاپ کن. و چندبار دیگه همینو انجام بده. آخر هم یه سری ستاره برای عرضش چاپ کن.

## تمرین ۲-۱:

یه مثلث قائم‌الزاویه بکشین!

## پاسخ ۲-۱:

```
print('*')
print('* *')
print('* * *')
print('* * * *')
print('* * * * *')
print('* * * * *')
print('* * * * *')
print('* * * * *')
print('* * * * *')
print('* * * * *')
print('*****')
```

حالا مورد زیری رو امتحان کنین:

```
print('5 + 6')
```

دیدین؟ عیناً چیزی که داخل کوتیشن نوشته بودیم رو چاپ کرد. کاری نداره اصلاً چی هست! صرفاً چاپش می‌کنه. (درواقع هرچی داخل علامت کوتیشن باشه، عیناً بدون هیچ تغییری چاپ میشه. بدون هیچ تغییری! اصلاً براش مهم نیست چیه. اصلاً نمیدونه اینا عددن! صرفاً فکر می‌کنه متن. برای همین عیناً چاپش می‌کنه)

خب حالا بریم یکم پیچیده‌تر. فرض کنین بخوایم چندتا چیز رو توی یه پرینت چاپ کنیم. اینجا میایم اون چندتا چیز رو با کاما<sup>۱۵</sup> از هم جدا می‌کنیم:

```
print(10, 11, 200)
```

output:

10 11 200

درواقع میاد می‌گه بهم گفتی اول ۱۰ رو چاپ کنم. بعدش ۱۱ رو چاپ کنم و بعدش ۲۰۰ رو چاپ کنم. (بین چیزا یه فاصله می‌ده موقع چاپ) یعنی اگر بخوایم چندتا چیز رو با هم توی یه خط چاپ کنیم از این استفاده می‌کنیم. مثلاً:

```
print('Hello', 11)
```

output: Hello 11

بهبش می‌گم اول بیا متن «Hello» رو چاپ کن و بعدش بیا عدد ۱۱ هم در کنارش چاپ کن.

---

<sup>۱۵</sup> دکمه کاما سمت راست کیبورد هست. نزدیک دکمه shift راست.

مثال:

```
print('Hello', 5 + 6)
```

output: Hello 11

اول متن «Hello» رو چاپ کن و بعدش بیا در کنارش، حاصل  $5 + 6$  (عدد ۱۱) رو چاپ کن. (خود پایتون بین چیزایی که قراره توی یه خط چاپ شن، یه فاصله قرار میده.

تمرین ۳-۱:

سعی کنین عبارت زیر رو توی خروجی چاپ کنین:

```
5 + 6 = 11
```

پاسخ ۳-۱:

```
print('5 + 6 =', 5 + 6)
```

output: 5 + 6 = 11

اینطوری میاد از چپ به راست پرینت می کنه. یعنی اول متنی که داخل کوتیشن گذاشتیم رو عیناً چاپ می کنه و بعدش حاصل جمع  $5 + 6$  رو با یه فاصله از متن چاپ می کنه.

حالا چیزای مختلف رو امتحان کنین که یاد بگیرین. برنامه نویسی همش خلاقیت و تمرینه. خودتون باید برای خودتون مسأله بسازین و تلاش کنین درکش کنین. مثلاً یه نمونه دیگه:

```
print('The sum of 5 + 6 is', 5 + 6)
```

جوابشو زیر صفحه می گم تا خودتون چک کنین.<sup>۱۶</sup>

تمیزنویسی:

کاما به عبارت قبلش می چسبه و از عبارت بعدیش یه فاصله پیدا می کنه.

correct: 

```
print('sum of 5 + 6 is', 11)
```

incorrect: 

```
print('sum of 5 + 6 is',11)
```

خب اما یه راه دیگه هم هست که حالا از نظر من ساده تره (من توی این آموزش این حالت استفاده می کنم):

```
print(f'sum of 5 + 6 is {5 + 6}')
```

output:

```
sum of 5 + 6 is 11
```

---

<sup>16</sup> sum of 5 + 6 is 11

همچنین اینو بدونین که sum مخفف summation به معنای مجموع هست.

قبل باز کردن کوتیشن، یه `f` می‌ذاریم و عبارتی که می‌خوایم حساب بشه و حاصلش چاپ بشه (یعنی به صورت عادی ببینتش که بتونه حسابش کنه (نه به صورت یه متن))، توی یه گروه می‌گذاریم.

تاحالا شده دوست داشته باشین که ایموجی‌هایی که توی تلگرم و اینا می‌فرستین رو توی کدتون هم بذارین؟! خب کاری نداره! به اینا میگن «Unicode Emojie». هرکدومشون یه سری کد دارن که می‌تونین پرینتشون کنین.

کدها رو از وبسایت خود یونیکد می‌تونین دریافت کنین.<sup>۱۷</sup>

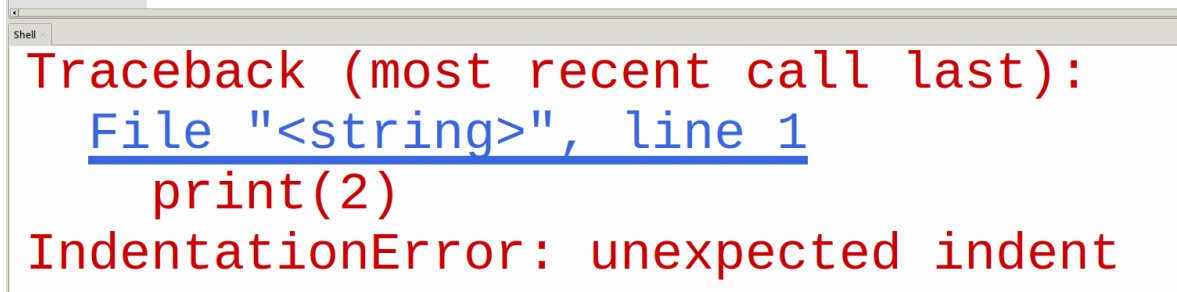
اول هر کد، «\U» قرار می‌گیره و شما کد رو قرار میدین. به جای علامت بعلاوه، اونقدر صفر می‌ذارین که هشت رقم شه. بعدش چاپش می‌کنین!<sup>۱۸</sup> مثلاً:

```
print("\U0001F600")
```

البته یکی کد رو بخونه نمی‌فهمه این عبارت عجیب و غریب یعنی چی. پس بهتره از یه مفهومی استفاده کنیم که در ادامه یادش می‌گیریم.

یه ارور رایج:

```
1 print(2)
```



```
Traceback (most recent call last):  
  File "<string>", line 1  
    print(2)  
IndentationError: unexpected indent
```

شما یه دستور نوشتی و میگی این درسته که! چرا ارور میده. ارور می‌گه خط ۱ به مشکل بر خوردم. ارور هم هست «unexpected indent». اگر دقت کنیم، یه فاصله قبل پرینت اضافه گذاشتیم. برای همین به ارور بر خورده. ما نباید فاصله‌ای رو کم یا زیاد قبل یه چیز بذاریم. دستورات باید بچسبن به خط. اگر فاصله بدیم، به ارور بر می‌خوره.

17 <https://unicode.org/emoji/charts/full-emoji-list.html>

۱۸ این رو اولین بار از «جادی» یاد گرفتیم. حتماً پیشنهاد می‌کنم رادیوگیک‌های جادی رو گوش بدین. کلی چیز به دانشتون اضافه میشه. از رادیوگیک شماره ۱ تا ۹۹ توی وبسایتش به آدرس: [jadi.net](http://jadi.net) و از ۱۰۰ به بعد توی کانال یوتیوبش.

## 2. Variable Naming & Working with variables

ببینیم ما در برنامه‌نویسی گاهی می‌خواهیم یه سری چیز رو یه جا توی کامپیوتر نگه داریم. خب کامپیوتر توی هوا که نمیتونه اونارو نگه کنه! یه قسمت بهش اختصاص میده و نگهش میداره. (درواقع توی RAM نگاهشون می‌داره) به این‌ها میگن «variable» یا «متغیر».

این‌ها مثل یه ظرف در نظرشون بگیرین که می‌تونیم چیز میز بریزیم داخلش. از اسمش معلومه. «متغیر»! یعنی تغییرپذیر. یعنی مثل یه ظرفی که محتوای رو می‌تونیم هی تغییر بدیم.

مثال:

$$x = 5$$

یعنی چی؟ مثلاً اینجا گفتیم که  $x$  برابر است با ۵. یعنی یه ظرفی به نام  $x$  برام در نظر بگیر و داخلش عدد ۵ رو قرار بده.

حالا عدد ۵ توی ظرفمون ذخیره شد. می‌تونیم با استفاده از دستور پرینت چاپش کنیم (تا وقتی بهش نگیم فلان چیزو چاپ کن، چاپ نمی‌کنه. همیشه حواستون باشه که اگر می‌خواین چیزی چاپ شه، پرینتش کنین):

$$x = 5$$

```
print(x)
```

output:

5

درواقع بهش می‌گیم برو x رو چاپ کن. نگاه می‌کنه می‌گه x چیه؟ عه عجیبا غریبا! توی کوتیشن هم نداشتی که معنانش این باشه عیناً متن x رو چاپ کنم.

آه‌ها!!!! یادم اومد! منظورت این بوده که X متغیر هست! پس مقدارشو چاپ می‌کنم.  
مثال:

$$y = 12.5$$

```
print('y is', y)
```

اینجا هم بهش می‌گیم که یه خونه برام در نظر بگیر. اسمشو بذار لا و برام ۱۲.۵ رو بریز توش. (علامت اعشار توی پایتون، «نقطه» هست)

بعدش یه عبارت و مقدار لا رو چاپ کردم.

منطوق زیوں یاتوں:

عبارت زیر رو در نظر داشته باشیم. اجراش کنین بینین که چی رخ میدہ.

```
print(age)
```

```
age = 32
```


می بینیم کہ ارور میدہ۔ خب دلیلش چیہ؟

پایتون یه برنامه هست که کدهای شما رو خطبه خط از بالا به پایین می خونه و اجرا می کنه؛ اول میگه خب خط یک چیه؟ خط یک گفته که . وایسا ببینم! من از بالا که شروع کردم به خوندن، اصلاً ندیدم که age ای تعریف کرده باشی که اصلاً بخوای پرینتش کنی! برای من اصلاً مهم نیست که بعدش تعریف کردی! باید قبلش تعریف کنی. چون من از بالا به پایین میام. برای همین حتماً حتماً، قبل استفاده از هر ظرفی (متغیری)، باید قبلش تعریفش کنی! یعنی:

```
age = 32
print(age)
```

نمی فهمه! چون از بالا به پایین خط به خط می خونه. خط اول رو می خونه میگه age رو چاپ کنم. خب عه! من چیزی به نام age بالا ندیدم. یعنی چی! من نمی فهمم چی میگی! اول برام تعریفش کن بعد بگو کار انجام بده باهاش یا چاپش کن. اول بهم بگو age چیه؟! اگر هم اجراش کنی، ارور زیر رو میده:

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

همونطور که می بینین، بعد اجرا بهمون ارور داده و گفته که age تعریف نشده است. یعنی منظورشه خب age کجاست؟! من از بالا داشتم می خوندم نبود که. اول بگو چی هست، بعدش من چاکرتم هستم. هرکاری بخوای واست انجام میدم.  (همیشه سعی کنی ارور رو بخونین و مشکل رو پیدا کنین. اگر نتونستین خودتون مشکل رو پیدا کنین، سعی کنین عیناً متن ارور رو سرچ کنین.)

### اشتباهی رایج:

سعی کنین مشکل این کد رو پیدا کنین:

```
age = 32
print(aga)
```

پاسخ:

یکی از اشتباهات رایج افراد اینه که هنگام استفاده از متغیر اشتباه تایپی پیدا می کنن و میگوین چرا ارور خورد! خب ببین تو بالا نوشتی «age» ولی پایین اشتباه تایپی داشتی! نوشتی «aga»! خب معلومه میگه چیزی به نام «aga» ندارم! کجاست!؟

همونطور که گفتیم، اسم اینا «متغیر» هست. یعنی تغییرپذیر. پس من می تونم مثلاً بگم از این به بعد، توی خونه y برام عدد ۲۰ رو بریز:

```
y = 12.5
y = 20
```

هیچ ایرادی نداره! میره خونه y مقدار جدید رو می ذاره داخلش و مقدار قبلی دیگه وجود نخواهد داشت:

```
y = 12.5
y = 20
```

```
print(y)
```

output:

20

به این کار می‌گن «assign» کردن یا همون «assignment». یعنی به  $y$  مقدار ۲۰ رو انتساب دادم. تازه می‌تونم پا رو از این هم که هست فراتر بگذارم و بگم از این به بعد توی این خونه  $y$ ، یه متن باشه! بله می‌تونم متن رو بهش assign کنم! مثلاً:

```
y = 12.5
```

```
y = 20
```

```
y = 'Hello!'
```

```
print(y)
```

output:

Hello

یعنی می‌تونم بگم از این به بعد توی خونه  $y$  برام متن Hello! رو بریز! می‌گه باشه! متغیر شبیه یه ظرفه؛ شبیه یه خونس! اوکیه باشه! من از این به بعد برات توی این خونه،  $y$  قرار میدم. به خط پرینت که برسه، میاد به آخرین تعریف  $y$  نگاه می‌کنه. می‌گه آها! توش Hello هست؟ باشه خب Hello رو چاپ می‌کنم.

نکته: به این متنا می‌گن «string». از این به بعد به جای متن بهشون می‌گم string.

به اعداد صحیح می‌گن «integer».<sup>۱۹</sup>

به اعداد اعشاری می‌گن «float».<sup>۲۰</sup>

اینا رو به خاطر داشته باشین.

## نکات و قوانین تعریف متغیر:

- ۱- باید با حرف یا «underscore» یا همون «\_» شروع بشه. (مثلاً با عدد نمی‌تونه شروع شه!)
- ۲- صرفاً از «0-9»، «A-Z»، «a-z»، و «\_» میشه استفاده کرد. (مثلاً خط فاصله «-» و علامت‌های عجیب و غریب مثل «&^%\$» نمیشه استفاده کرد!)
- ۳- بین کرکترهایی که تعیین می‌کنین، فاصله مجاز نیست. اگر اسم ظرفتون چند کلمه‌ای بود، بینشون «\_» بذارین.
- ۴- به بزرگی و کوچیکی حرف حساسه و براش متفاوت<sup>۲۱</sup> یعنی `amir = 5` و `Amir = 20` متفاوتن!
- ۵- سعی کنین متغیرها تون حروف بزرگ نداشته باشه. صرفاً با حروف کوچک بنویسین. (هرچند مانعی نیست ولی اینطوری تمیزتره!)

19 ..., -3, -2, -1, 0, 1, 2, 3, ...

20 e.g. 2.1, 2.3, 2.0, 153.7

21 Case sensitive

۶- از کلماتی که از قبل برای خود زبان برنامه‌نویسی رزروشدن و مال خود زبونن، همیشه استفاده کرد. حالا بعداً مثلاً شو می‌بینیم. کلماتی مثل if و while و ... (اگر شروع کنین به نوشتن، خود IDE یا Text editor تون، کلماتی که رزروشدن رو پیشنهاد میده. خب مطمئناً نباید عین اون باشه! قاطی میشه خب!)

مثال:

`1variable`

+ عدد نمیتونه اولین کرکتر باشه.

`Variable-2`

+ سمبل‌ها رو نمی‌تونین استفاده کنین. صرفاً «underscore» رو میشه استفاده کرد و نه علامت منهای!

`var@var`

+ سمبل (مثل هشتگ و @) رو همیشه استفاده کرد!!!

`variable 2`

+ توی اسم، فاصله همیشه گذاشت!

`sum`

```
number = 5
sum = 5
print(number)
```

اگر بنویسینش می‌بینین رنگش با متغیرهای عادی متفاوت میشه (شبهه print شده) و از همینجا هم می‌فهمین که یه مشکلی داره!

درواقع یه سری کلمات هستن که برای پایتون معنای خاصی دارن. درواقع برای زبان پایتون رزروشدن. مثلاً یادتونه که دستور پرینت معنای خاص داشت برای پایتون؟ درواقع کلی از این کلماتی که مثل پرینت داریم. یکی از اونها کلمه «sum» هست. برای همین پایتون می‌گه برای اینکه با اون کلمه خاص قاطی نشه، اسمت رو عوض کن خب! یه چیز دیگه بذار.

- آیا همیشه چیزی مثل زیر به کار برد؟

`x = 'hello' 5`

+ نه! همیشه! صرفاً یه نوع چیز می‌تونین توی یه خونه بگذارین. همیشه ده تا چیز بدین بهش! یا عدد بده یا متن! همیشه هردوش!



- همیشه پس هم عدد صحیح بدم هم اعشاری؟ یه چیزی مثل:

```
x = 5 10.6 3.3
```

نه نه! صرفاً یک چیز. فقط یک چیز! صرفاً فقط یک چیز می‌توانین در متغیر بگذارین!

### مثال‌هایی درست از تعریف متغیر:

```
x = 10
```

```
amir = 20.5
```

```
amir = 'Hello'
```

```
hello_this_is_a_variable2 = -20 * 2
```

الف) متغیر می‌تونه طولانی باشه و شامل چند کلمه. برای خوانایی بیشتر بینشون «\_» می‌ذارن. اما خب سعی کنین تا حد امکان طولانی نباشه که خوندنش برای خودتونم سخت میشه بعداً. مثلاً توی موارد بالا، آخریش خیلی طولانیه. مشخصاً کد رو زشت و کثیف کرده. پس سعی کنین تا حد امکان کوتاه، ولی در عین حال خوانا باشه.

ب) عدد هم می‌تونه توی اسم متغیر باشه. صرفاً نباید اولین کرکترش باشه. مثلاً خیلی رایجه که چندتا عدد بخوایم اینطوری نامگذاری می‌کنن:

```
number1 = 4
```

```
number2 = 7
```

```
number3 = -2
```

ج) قسمت سمت راست می‌تونه یه عبارت ریاضی باشه! موقعی که پایتون می‌رسه به علامت مساوی، میاد اول سمت راست مساوی رو حساب می‌کنه و بعدش می‌ریزه توی سمت چپ. پایتون می‌گه که خب ۲۰- ضربدر ۲ میشه ۴۰- . حالا میاد ۴۰- رو می‌گذاره توی متغیرمون.

تمرین!

سعی کنین چیزمیز مختلف برای خودتون تعریف کنین و پرینت کنین تا ببینین چی میشه. برای خودتون تمرین کنین.

حل!

```
name = 'kamal'
```

```
print(name)
```

output:

```
kamal
```

```
-----
```

```
name = 'kamal hastam'
```

```
print(name)
```

output:

```
kamal hastam  
first_name = 'kamal'  
print(first_name)
```

output:

```
kamal  
-----
```

```
age = 30  
print(age)
```

output:

```
30  
-----
```

```
name = 'Amir'  
age = 32  
print(f'Hello. I am {name} and I am {age} years old.')  
print('Hello. I am', name, 'and I am', age, 'years old.')
```

output:

```
Hello. I am Amir and I am 32 years old.  
Hello. I am Amir and I am 32 years old.
```

توضیح: خب حتماً یادتونه که گفتیم می‌تونیم چندتا چیز رو توی یه دستور پرینت چاپ کنیم؟ کافیه صرفاً بین چیزا یه علامت کاما بگذاریم. اینجا همینکارو کردیم. گفتیم که اول برام متن «Hello. I am» و بعد متغیر `name` و بعدش عیناً متن «and I am» و بعدش متغیر `age` و بعدش عیناً متن «years old» رو چاپ کن.

همونطور که دیدین، هر دو نوع پرینت رو براتون نوشتم. مورد اول رو اگر دقت کنین خیلی ساده‌تره. حداقل به نظر من (🤖\_🧐)

مثلاً می‌تونیم جمع دو عدد رو حساب کنیم و چاپش کنیم:

```
number1 = 5  
number2 = 10  
summation = number1 + number2  
print(summation)
```

output:

```
15
```

توضیح:

۱- متغیر number1 برابر ۵ قرار بده.

۲- متغیر number2 رو برابر ۱۰ قرار بده.

۳- متغیر summation رو برابر جمع number1 و number2 قرار بده.<sup>۲۲</sup> (درواقع حواستون باشه که همیشه سمت راست حساب میشه و ریخته میشه توی سمت چپ. یعنی اینجا میاد اول حاصل جمع رو حساب می‌کنه. حاصلش هرچی شد میریزه توی summation)

۴- مقداری که توی summation هست رو چاپ کن.

یکم می‌تونین بازی کنین باهاش. مثلاً خط پرینت رو اینطوری بنویسین:

```
print(f'sum is {summation}')
```

output:

```
sum is 15
```

یا حتی اینطوری بنویسین:

```
number1 = 5
number1 = 5
number2 = 10
summation = number1 + number2
output_text = 'sum is'
print(f'{output_text} {summation}')
```

output:

```
sum is 15
sum is 15
```

درواقع حتی متن چاپی هم اضافه کردم ولی ریختمش توی یه متغیر. پرینت اولی همون روش f رو رفتم که یه متنی هست که داخلش می‌تونیم حاصل چیز میز چاپ کنیم. یکی از اون چیز میزا می‌تونه حاصل درون متغیر باشه!

پرینت دومی هم گفتم اول متغیر output\_text رو چاپ کن و بعدش متغیر summation رو چاپ کن. که متغیر اولی رو چاپ می‌کنه. بعدش یه فاصله می‌ده و دومی رو چاپ می‌کنه.

نکته! همیشه بدونین که سمت راست حساب میشه و ریخته میشه تو سمت چپ. پس می‌تونیم بنویسیم:

```
age = 18
age = age + 2
```

<sup>۲۲</sup> توضیح فنی‌تر: برو توی خونه number1 مقدارشو بیار و با مقداری که توی خونه number2 هست جمع بزن و بریز توی خونه‌ای که اسمشو sum گذاشتی.

```
print(age)
```

output:

```
20
```

یعنی میره سمت راست میگه خب age چند بود؟ آها برابر با ۱۸ بود. خب با ۲ جمعش می‌کنم میشه ۲۰. حالا ۲۰ رو دوباره می‌ریزم توی ظرف age. (گفتیم متغیر مثل یه ظرفه! میشه چیز جدید ریخت توش و از این به بعد فقط اون چیز جدید توش خواهد بود) یا حتی می‌تونیم بنویسیم:

```
num1 = num2 = 2
```

درواقع اول میاد ۲ رو میریزه توی num2 و بعد مقدار num2 که ۲ هست رو می‌ریزه توی num1. درواقع هر هم num1 و هم num2 برابر ۲ میشن.

تذکر! حواستون باشه که مطمئناً نمی‌تونین یه متن رو با یه عدد جمع کنین! مثلاً:

```
name = 'Bruce'
```

```
age = 34
```

```
x = name + age
```

فلاصه یکم چیز میز تمرین کنین. مثل اینبا که من کلی تمرین کردم، شما هم تمرین کنین.

### متوسط:

دراقع در ابتدا که زبون‌های برنامه‌نویسی اینقدر ساده نبودن، ما می‌گفتیم برو خونه ۱۲۰۰ ام رم، برام عدد ۵ رو قرار بده.

حالا هر وقت نیاز به محتوای اون قسمت حافظه داشتم، می‌گفتم برو خونه ۱۲۰۰ ببین توش چیه. اما زبون‌های برنامه‌نویسی اومدن و کار ما رو راحت کردن. به جای اینکه من بگم برو خونه ۱۲۰۰ حافظه، اسم اون خونه رو یه چیز با معنی انگلیسی می‌ذارم. حالا هر وقت خواستم به اون قسمت دسترسی پیدا کنم و محتوایش رو بخونم، اسم رو می‌نویسم. پایتون میره میگه خب اسم x چی بود؟ آهاااا یادام اومد! خونه ۱۲۰۰ حافظه رو اسم گذاشته بودم براش. اسمش رو گذاشته بودم x.

درواقع من هروقت اسم x رو اوردم، میره سراغ اون آدرس رم که ببینه چی اونجاست و بخونتش. درواقع x یه اسمی هست که من هر وقت بخوام با زبان برنامه‌نویسیم صحبت کنم، با استفاده از x صحبت کنم. نخوام بگم برو فلان آدرس مموری رو بخون. صرفاً بهش میگم x و خودش می‌فهمه منظورم کجاست.

**توجه!** نامگذاری یکی از مهمترین چیزایی هست که باید بهش دقت کنین. وگرنه در آینده به مشکل می‌خورین.

بذارین یه سؤال بپرسم. فرض کنین شما توی کدتون به یه مشکل برخوردین. حالا می‌خوان از یکی بپرسین که مشکلتش چیه. کد زیر رو می‌بین به طرف نشون میدین و مثلاً می‌پرسین مشکلتش چیه:

```
x = 2024
```

```
j = 7
```

```
h = 21
```

به نظرتون طرف شروع کنه به خوندن گیج نمیشه؟ x چیه؟ j چیه؟ h چیه؟! یا حتی ممکنه خودتون بعد دو ماه برگردین و به کدتون یه نگاه بندازین. گیج نمی‌شین که هرکدوم چی هست؟! یادتون می‌مونه که x چی بود؟ نه!

پس سعی کنین اسم متغیرها تون رو درست انتخاب کنین. مورد بالا رو می‌تونیم اینطور بنویسیم:

```
year = 2024
```

```
month = 7
```

```
day = 21
```

خیلی خواناتر و بهتر نشد؟ هرکی کد رو بخونه می‌فهمه چی نوشتین! همکارتون توی شرکت می‌گه آها تاریخن پس! پس سعی کنین اسما توضیح دهنده کاربرد باشن. نه خیلی طولانی نه خیلی کوتاه. بلکه یه مقدار متناسب. مثلاً:

```
country_name = 'Spain'
```

```
student_id = '24'
```

```
birth_year = 1992
```

```
student_count = 156
```

```
final_result = 56.4
```

این خیلی کمک می‌کنه که شما بفهمین دارین چیکار می‌کنین. اسمایی مثل x و اینا واقعاً گیج‌کنندن و آدم نمی‌فهمه داره چیکار می‌کنه!

نکته! فرض کنین بنا به کاری، می‌خوانین یه عددی رو به شکل string (انگار متنه و نه عدد!) داخل یه متغیر بگذارین، خوبه توی اسمش جوری بهش اشاره کنین که طرف بفهمه. مثلاً:

```
salary_string = '56000'
```

چون اگر من صرفاً salary<sup>۲۳</sup> رو ببینم، حس می‌گه داخلش یه عدد صحیح (integer) هست. ولی اگر ببینم توی اسمش نوشته شده string، حواسم هست که salary ما به شکل متنی (string) توی متغیر گذاشته شده. (و نه به صورت عددی!)

کلاً چیزای غیرمعمول مثل ذخیره یه عدد به شکل string و... رو یه جوری بهش اشاره کنین خیلی خوانایی کدتون رو بالا می‌بره.

تمرین:

---

۲۳ اسم متغیر رو بامعنی تعریف کردم. حقوق سالانه یه فرد به انگلیسی میشه «salary».

۱- برنامه‌ای بنویسین که مساحت یه دایره به شعاع ۴ رو حساب کنه. و در نهایت عبارت زیر رو نشون بده:

The area of the circle is 50.24

۲- برنامه‌ای بنویسین که اول سه تا متغیر رو تعریف کنه و بعد میانگین اون‌ها رو حساب کنه.  
۳- برنامه‌ای بنویسین که جای دو متغیر رو عوض کنه. یعنی مقداری که متغیر اول هست با مقداری که توی متغیر دوم هست عوض شه. یعنی:

مقدار نهایی متغیر اول = مقدار اولیه متغیر دوم  
مقدار نهایی متغیر دوم = مقدار اولیه متغیر اول  
یعنی فرض کنین اگر

a = 2

b = 5

باشه، در نهایت حاصل درون متغیر a بشه ۵ و در b مقدار ۲ وجود داشته باشه.

پاسخ ۱:

```
pi = 3.14
```

```
radius = 4
```

```
area = radius * radius * pi
```

```
print(f'The area of the circle is {area}')
```

اول عدد ۳.۱۴ رو گذاشتم توی متغیری به نام pi. بعدش شعاع رو گذاشتم توی متغیری به نام radius که تو انگلیسی به معنای شعاع هست.<sup>۲۴</sup>  
بعدش هی متغیر دیگه قرار دادم به اسم area و حاصل مساحت که شعاع \* شعاع \* عدد پی هست رو ریختم توش.

در آخر هم طبق فرمتی که خواسته بودم، چاپش کردم.

به همین سادگی و خوشمزگی! ٢٠٢٠

پاسخ ۲:

```
num1 = 2
```

```
num2 = 10
```

```
num3 = 7
```

```
count = 3
```

```
average = (num1 + num2 + num3) / count
```

```
print(f'average is {average}')
```

خب سه متغیر رو تعریف کردم. تعدادشونم ریختم توی یه متغیر. بعدش گفتم جمع سه عدد تقسیم بر تعداد رو بریز توی مقدار میانگین (average).

---

<sup>۲۴</sup> دقت می‌کنین که اسامیم با معناس؟ یادتونه درباره اهمیت نام‌گذاری صحبت کردم؟ راستی انگلیسی‌تون هم تقویت کنین. نه صرفاً برای نام‌گذاری (: بلکه برای آینده خودتون (:

- خب یه سوال؟ میشه count رو هم ننوشت و تعریف نکرد و قسمت میانگین، تقسیم بر ۳ کرد. یعنی اینطوری:

$$\text{average} = (\text{num1} + \text{num2} + \text{num3}) / 3$$

+ بله میشه؛ ولی خب حالت اولی قشنگ تر و خواناتر هست. همچنین اگر زمانی بخواین کد رو تغییر بدین و یه متغیر دیگه اضافه کنین، اونوقت باید کل کدتون بگردین و بینین کجاها نوشته بودین ۳ و تغییرش بدین و بکنینش ۴. اما حالت اول، بعد اضافه کردن یه متغیر جدید، صرفاً مقدار count رو عوض می کنین و می گذارینش ۴ و خیالتون راحت دیگه جایی از کد نیاز به تغییر نداره و از بسیاری از مشکلاتی جلوگیری می کنه.

الآن برنامهتون کوچیکه و چند خط بیشتر نیست ولی برنامهتون بزرگ شه، چند هزار خط شه، با یه تغییر کوچیک، کل برنامهتون به هم میریزه! حالا بیا درستش کن و کل کد رو چک کن که آیا نیازی هست عددا تغییر کنن یا نه؟

پاسخ ۳:

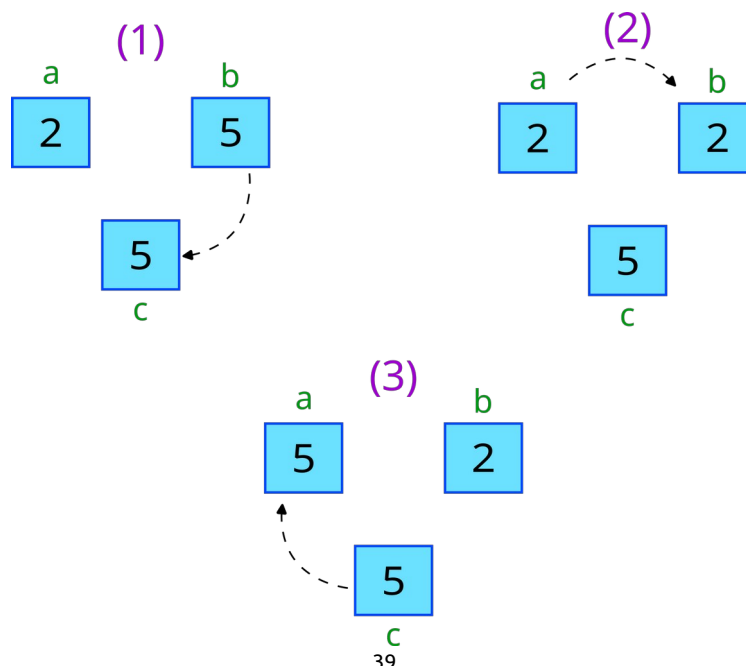
یکم راهنماییتون می کنم. فرض کنین دو جعبه دارین که توی هر جعبه یه توپ هست. می خواین توپ جعبه اول بره جعبه دوم و توپ جعبه دوم بره جعبه اول. همچنین نمیشه دو توپ توی یه جعبه باشن. این زمان چیکار می کنین؟

توپ اول رو در میارین.

توپ دوم رو می گذارین توی جعبه اولی.

توپ اولی رو می گذارین توی جعبه دوم.

خب اما توی دنیای واقعی شما توپ اولی رو که در آوردین یه جا توی دستتون نگهش می دارین. یا روی میز می گذاریدش. توی کامپیوتر هم باید مقدار رو یه جا به صورت موقت نگه داری کنین. محل نگهداری چیزا کجاس؟ آفرین! متغیر!



۱- من از یه متغیر سوم کمک می‌گیرم. میام  $b$  رو می‌ریزم توی یه متغیر سوم.  
 ۲- پس حالا قبول دارین که انگار یه کاپی از  $b$  گرفتم؟ پس می‌تونم  $b$  رو خراب کنم و  $a$  رو بریزم توش.

۳- مقدار  $b$  (همون  $c$ ) باید بره توی  $a$ . خب می‌فرستیمش!  
 به همین سادگی!  
 بریم روی کد!

```
a = 2
b = 5
c = b
b = a
a = c
```

درواقع من می‌گم متغیر اولی رو توی یه متغیر دیگه به صورت موقت بذار. اسمش با توجه به کاربردش بامعنا انتخاب می‌کنم و می‌گم اسمش مخفف temporary هست.  
 بعدش متغیر ۲ رو می‌ریزم توی ۱. چون خیالم راحت‌تره یه کاپی از ۱ توی temp دارم. حالا temp رو می‌ریزم توی var2 (یعنی انگار عیناً مقدار var1 رو ریختن تو var2)

## برای خوانایی:

گاهی نیازه یه عدد بزرگ رو توی یه متغیر بریزین. اما عدد بزرگه و ممکنه یکی دو صفر اشتباه کنین. برای خوانایی بیشتر، می‌تونین بین اعداد «\_» یا همون «underscore» اضافه کنین. مثلاً یک میلیون و نه صد هزار:

```
x = 1_900_000
```

الآن خیلی مشخص‌تره تا اینکه بینش هیچی نباشه. پایتون خودش به صورت پیشفرض، underscore های توی یه عدد رو نادیده می‌گیره و صرفاً برای خواناییه نویسنده کد هست.

با نماد علمی آشنایی دارین؟  
 نماد علمی می‌گه که اعداد رو به صورتی می‌نویسیم که خواناییشون راحت‌تره باشه. یعنی مثلاً ۲۰۰۰ رو اینطور می‌نویسیم:

$$2000 = 2.0 \times 10^3$$

یه رقم رو می‌بریم قبل اعشار و بقیه سمت راست اعشار و بعدش بر طبق اون، توان ۱۰ رو می‌نویسیم.

$$0.9 \rightarrow 9.0 \times 10^{-1}$$

$$50 \rightarrow 5.0 \times 10^1$$



$$561.43 = 5.6143 \times 10^2$$

$$0.932 = 9.32 \times 10^{-1}$$

درواقع هر چیزی رو به صورت یه عدد که یه رقم اعشار داره، ضربدر ۱۰ به توان یه چیزی می نویسیم. (اینطوری خواناتره. توان مثبت یعنی مثلاً ۲ تا اعشار رو ببر راست. توان منفی یعنی اعشار رو بیار سمت چپ)

این در پایتون هم هست که بهتر بتونیم اعداد رو تعیین کنیم. نخوایم صفر بشمریم. ولی توی پایتون به جای ۱۰، علامت «e» هست. یعنی:

$$561.43 = 5.6143 \times 10^2 = 5.6143e2$$

مثلاً اگر خواستین 0.0001 رو تعریف کنین، می نویسین:

```
num = 1e-4
```

این از شمردن صفر و اشتباه کردن توی دیدار جلوگیری می کنه.<sup>۲۵</sup>  
یا مثلاً ۰.۰۷ رو می تونیم اینطوری نشون بدیم:

```
num = 7e-2
```

اگر هم چاپش کنیم دقیقاً بهمون همین رو می گه:

```
print(num)
```

output:

```
0.07
```

مرور!

لطفاً قسمتای قبل رو به دور مرور کنین و با مثالا بازی کنین. که می خوایم بریم سر چیزای جدیدتری که پیش زمینش موارد قبلی هست. تا وقتی به این موارد مسلط نشدین، به هیچ وجه ادامه رو نخونین!

## 4. input()

دیدین وقتی وارد یه وبسایتی میشین، مثلاً ایمیلتون رو وارد می کنین؟  
حالا می خوایم همین چیزا رو توی پایتون انجام بدیم. یعنی یه چیزی از کاربر بگیریم. به این میگن «input گرفتن یا «ورودی گرفتن».

خب بیایم منطقی فکر کنیم. ورودی رو که گرفتیم، توی هوا که نمی تونیم نگهش داریم! باید بریزیم توی یه جایی که نگهش داره. یا همون متغیر!  
خب روش نوشتاری (که بهش می گن syntax) اش اینه:

```
name = input()
print(f'name is {name}')
```

<sup>۲۵</sup> بعد خوندن قسمت «تابع» و «لایبرری»، برگردین و این ویدیو رو ببینین:

Readable large numbers (1000000 -> 1M):  
<https://www.youtube.com/shorts/fhaSL6ucEzk>

خب اجراش کنیم.

- عه! من اجراش کردم ولی چیزی نمایش داده نمیشه!  
+ دقیقاً! پایین نگاه کن. یه چشمکزن وجود داره نه؟! درواقع منتظره ورودی بگیره ازت. یه چیزی بنویس.

- نوشتم. ولی بازم اتفاق نیوفتاد!  
+ خب کامپیوتر باید یه جوری بفهمه که نوشتن تو تموم شه. این تموم شدن رو با زدن دکمه «enter» بهش میگی.  
- عه راست میگی! دکمه اینتر زدم و بعدش دقیقاً چیزی که نوشته بودم رو توی متن نوشت.

خب بیایم یکم مکانیزم ورودی گرفتن رو بهتر کنیم. پایتون خودش گفته وقتی می‌خواهی ورودی بگیری، توی پرانتز می‌تونی متنی که می‌خواهی رو بنویسی که طرف بفهمه باید یه ورودی بهت بده. مثلاً:

```
name = input("Enter your name: ")  
print(f'name is {name}')
```

حالا اجراش کنین.

بهتر نشد؟ دیدین؟ داره کم کم شبیه یه برنامه واقعی میشه. کم کم یه سیخونکایی دارن بهمون میزنن که واقعاً داریم برنامه می‌نویسیم.

به صورت پیشفرض input قصه ما همه چیز رو به صورت string (متنی) میبینه. اصلاً براش مهم نیست چی بهش میدیم. هرچی بدیم می‌ذاره توی کوتیشن و به صورت متنی می‌بینتش. پس اگر بخوایم بهش عدد صحیح یا اعشاری بدیم چی؟

اینجا موضوعی پیش میاد به نام type casting. نترسین! اسمش فقط یه خرده عجیبه!

فرض کنین من متن ۱۸ رو داخل یه متغیر ذخیره کردم:

```
age = '18'
```

یادتونه دستوراتی مثل پرینت داشتیم؟ یه دستور هم داریم به نام «int» که میاد یه چیزو تبدیل می‌کنه به عدد صحیح:

```
age = '18'  
age = int(age)
```

اینجا ما باید بگیریم اون چیزی بود که به صورت string دیدیا، اون یه عدد صحیحه. تبدیلهش کن به عدد صحیح.

یادتونه گفتیم پایتون هر وقت مساوی رو دید میره طرف راست تساوی؟ اینجا هم میره سمت راست می‌گه باید تبدیل کنم به عدد صحیح (int). تبدیل می‌کنه و بعدش عدد ۱۸ رو درون age قرار میده. (یادتونه گفتیم متغیر قابلیت تغییرپذیری داشت؟ اینجا هم تغییرش میدیم و عدد ۱۸ رو توش می‌ذاریم)

درواقع به این کار می‌گن cast<sup>۲۶</sup> کردن:

---

<sup>۲۶</sup> کلمه cast یعنی به شکل یه چیز در آوردن. (درواقع ما می‌گیم به شکل int در بیار)

`int()` → تبدیل به عدد صحیح

`float()` → تبدیل به عدد اعشاری

`str()` → تبدیل به استرینگ

پندر تا چیز رو با هم تست کنیم (پاسخ هر قسمت پایین صفحه نوشتم که فکر کنین روشن):

```
age = 18.9
```

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

```
print(age)
```

output<sup>27</sup>

```
age = 18
```

```
age = float(age)
```

```
print(age)
```

output<sup>28</sup>

```
age = '18'
```

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

```
age = age + 2
```

```
print(age)
```

output<sup>29</sup>

```
age = '18'
```

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

```
age = age + 2
```

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

output<sup>30</sup>

```
age = int('18')
```

```
age = float(age)
```

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

output<sup>31</sup>

---

27 Output: 18

28 Output: 18.0

29 Output: 20

30 Output: 40

31 Output: 20.0

از عدد اعشاری تبدیلیش کرد به عدد صحیح. (اعشار رو کامل حذف می‌کنه).

تبدیل کرد به اعشاری. (چون اعشار نداشت، اعشار اضافه کرد).

چون قبلیش اعشاری شد دیگه!  $22.0 = 2 + 20.0$

## تمرین!

حالا می‌خوایم به سن ورودی بگیریم. سن رو ۲ تا زیاد کنیم و بعدش چاپش کنیم. (بپه کوشولو! اول فوت بنویس و بعدش پاسفو بفون!)

## پاسخ:

اول باید به ورودی بگیریم:

```
age = input("Enter your age: ")
```

خب بعدش مگه نگفتیم ورودی به صورت string گرفته میشه؟ خب باید تبدیلش کنیم به int (عدد صحیح):

```
age = input("Enter your age: ")
```

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

حالا عدد شد! حالا می‌تونیم ۲ تا بهش اضافه کنیم و در نهایت چاپش کنیم:

```
age = input("Enter your age: ")
```

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

```
age = age + 2
```

```
print(age)
```

راه ساده‌تر می‌خواهین؟ بریم به راه ساده‌تر:

```
age = int(input("Enter your age: "))
```

```
print(age)
```

همیشه پرانتز رو از داخلی‌ترین بخونین:

یه ورودی بگیر؛ ورودی رو که گرفتی، تبدیلش کن به عدد صحیح و بعدش بریز داخل **age**.<sup>۳۲</sup>

**متوسط:** یادتونه که قبلاً صحبت کردیم که زبونی سطح بالا خیلی چیزها رو براتون هندل می‌کنن و نیازی نیست که شما بخواین انجامشون بدین. خیلی کمکتون می‌کنن.

مثلاً این تبدیل تو زبونی مثل C خیلی سخت‌تره. ولی اینجا به سری کد برای اون زبان سطح بالا تعریف شده که من هر وقت زدم `int()` بره اون کدا رو اجرا کنه و حاصل رو خودش حساب کنه. به این چیزایی که پرانتز باز و بسته دارن میگن تابع. کدهایی که نوشته شدن که به ما کمک کنن. نیاز نباشه من کدی بنویسم که تبدیل رو انجام بده. صرفاً ازش استفاده می‌کنم. با مفهوم بعداً بیشتر آشنا میشیم.

حالا سعی کنین کدای قبلی رو به جای در هنگام تعریف متغیر، خودتون به متغیر مقدار بدین، مقدار رو از ورودی گرفته باشین.

یکیشو خودم واستون حل می‌کنم:

**سوال:** برنامه‌ای بنویسین که شعاع یک دایره را گرفته و مساحت و محیط آن را حساب کند.

## پاسخ:

<sup>۳۲</sup> یکی از رایج‌ترین مشکلات افراد هنگام استفاده از چند پرانتز اینه که مثلاً ۲ تا پرانتز رو باز می‌کنن ولی صرفاً به پرانتز رو می‌بندن. مثل این:

```
age = int(input("Enter your age: "))
```

```
pi = 3.14
radius = float(input("Enter radius: "))
area = radius * radius * pi
print(f'The area of the circle is {area}')
```

به صورت float یا اعشاری گرفتیم. چون شعاع لزوماً عدد صحیح نیست! ممکنه طرف عدد اعشاری وارد کنه.

طبیعتاً اگر هنگام تبدیل (cast) کردن، چیز اشتباهی بدیم، ارور میده. مثلاً:

```
s = 'a'
print(int(s))
```

ValueError: invalid literal for int() with base 10: 'a'

میگه گفتی int کن ولی خب «a» یه حرف الفباس! من عددی نمی بینم که تبدیل به int کنم!

لطفاً تا وقتی به این قسمت مسلط نشدین، ادامه رو نخوانین!

## 5. if

خب این قسمت از اون قسمتای مورد علاقه من هست. چون کم کم حس واقعی برنامه نویسی رو درک می کنین!

قبلش یه خرده با علامتای ریاضی بیشتر آشنا بشیم:

اپراتور	اسم	مثال	
>	بزرگتر	$6 > 5$	بزرگتر از ۵ ۶
<	کوچکتر	$-2 < 3$	منفی ۲ کوچکتر از ۳
>=	بزرگتر یا مساوی	$2 \geq 2$ <sup>33</sup>	بزرگتر یا مساوی ۲ (دقت ۲ کنین. علامت بزرگتر یا مساوی هست! پس عبارت درستیه که بنویسیم ۲ بزرگتر یا مساوی ۲ است)
<=	کوچکتر یا مساوی	$3 \leq 6$	سه کوچکتر یا مساوی ۶
==	مساوی بودن	$3 == 3$	مساوی هست با ۳ ۳
!=	نامساوی بودن	$5 != 10$	نامساوی ۱۰ ۵

۳۳ حواستون باشه که همونطور که اسمش «بزرگتر یا مساوی» هست، اول باید علامت بزرگتر و بعد علامت مساوی رو بگذارین.

فرض کنین شما می‌خواین پرداخت بانکی انجام بدین. شما مبلغ رو وارد می‌کنین؛ رمز عبور رو می‌زنین و منتظر پرداخت شدن می‌مونین.  
در پشت صحنه این اتفاق می‌وفته:

اگر موجودی حساب بیشتر یا مساوی مقدار تراکنش است، پرداخت رو انجام بده.  
درواقع ما توی زبون‌های برنامه‌نویسی، یه ساختاری داریم که اگر یه شرطی برقرار بود (مثلاً اینجا اگر موجودی  $\geq$  مقدار تراکنش، هست)، یه کاری رو انجام بده. به این ساختار می‌گن «اگر» یا «if»:

اگر {شرط}:

...

...

یا درواقع:

```
if {condition}:
```

```
...
```

```
...
```

بیایم مثال بزنین:

```
card_balance = 500
```

```
transaction_amount = int(input("Enter transaction amount: "))
```

```
if card_balance >= transaction_amount:
```

```
    print("It's OK")
```

می‌گیم اگر `card_balance` بزرگ‌تر یا مساوی `transaction_amount` بود، حالا دو نقطه (به معنای اینکه کارهای زیر رو انجام بده).<sup>۳۴</sup>  
- چه کارهایی؟

+ برای اینکه نشون بدیم کدوم کارها رو باید انجام بده، چهارتا فاصله می‌ایم جلو. (چهارتا دونه `space`)

مثلاً اینجا گفتیم که اگر مقدار موجودی کارت بزرگ‌تر یا مساوی مقدار تراکنش بود (کارت به اندازه کافی موجودی داره)، چاپ کن `it's OK`

**تذکره!** حواستون به دو نقطه و چهارتا فاصله باشه! این خیلی مهمه. اگر رعایتش نکنین، به ارور بر می‌خورین. امتحانش کنین تا یاد بگیرین:

```
card_balance = 500
```

```
transaction_amount = int(input("Enter transaction amount: "))
```

```
if card_balance >= transaction_amount:
```

```
print("It's OK")
```

```
IndentationError: expected an indented block after 'if' statement on line 3
```

<sup>۳۴</sup> حواستون باشه که دو نقطه بچسبه به کرکتر قبلیش و فاصله نداشته باشه!

دیدین؟ ارور داده می‌گه indentation رو رعایت نکردی! این یعنی حواست نبوده و اون چهارتا فاصله رو رعایت نکردی. به اون می‌گن indentation.

```
card_balance = 500
transaction_amount = int(input("Enter transaction amount: "))
if card_balance >= transaction_amount
    print("It's OK")
```

SyntaxError: expected ':'

syntax یعنی همون نحوه نوشتار به زبون. می‌گه نحوه نوشتاری که پایتون می‌فهمه رو رعایت نکردی. می‌گه من انتظار داشتم (expect) به دو نقطه ببینم ولی نیووردیش. یاد بگیرین ارور بخونین و درستش کنین. اگر نفهمیدینش، توی اینترنت سرچ کنین. آدمای زیادی مثل شما به ارور خوردن و دربارش پرسیدن توی وبسایتای مختلف و شما با سرچ، به اون سؤال و جواب‌ها می‌رسید. عیناً متن ارور رو کاپی پیست کنین و سرچ کنین!

**نکته!** شاید متوجه شده باشین که من به جای کوتیشن، دبل کوتیشن به کار بردم. فرقی نداشت. ولی به نظرتون چرا اینجا دبل کوتیشن به کار بردم؟ یکم فکر کنین! چون اگر کوتیشن عادی به کار می‌بردم، با کوتیشن توی «it's» قاطی میشد! پایتون می‌گه اگر می‌خوای کوتیشن یا دبل کوتیشن داخل string ات به کار ببری، کوتیشن دو طرف متفاوت باشه که من قاطی نکنم! یا قبلش به بک‌اسلش بذار که بفهمم منظورت خودشه و با کوتیشن‌های طرفین اشتباهش نگیرم. بک‌اسلش به کرکتر ویژه هست برای نشون دادن چیزها به پایتون

```
print('It\'s OK!')
```

- عه! خب اگر بخوام توی متنم (string ام) بک‌اسلش به کار ببرم چیکار کنم؟ + دوتا بک‌اسلش پشت هم بذارین. اینطوری می‌فهمه منظورتون بک‌اسلش بوده و نه کرکتر ویژه.  

```
print('\\')
```

یه سری کرکتر ویژه داریم توی پایتون. می‌تونین از وبسایت زیر بخونین:

[https://www.w3schools.com/python/gloss\\_python\\_escape\\_characters.asp](https://www.w3schools.com/python/gloss_python_escape_characters.asp)

مثلاً \n مثل enter عمل می‌کنه و new line هست.

یکی از چیزایی که می‌خواستم توی این آموزش رعایت کنم، یادگیری با کاربرد بود. یعنی نیومدم کل قواعد string رو موقع معرفی بگم. چون واقعاً شیوه درستی نیست! حداقل اینطوری من یاد نمی‌گیرم. چون به دفعه با کوله‌باری از نکته و تذکر مواجه میشم و نمی‌فهمم چی شد! بلکه اگر دونه‌دونه در طی زمان یاد بگیرم، خیلی بهتره و بهتر متوجه میشم.

منم نیومدم مثلاً کوتیشن رو توی قسمت string بگم. بلکه با هم کم کم میریم جلو و نکات و تذکرات رو کاربردی یاد می‌گیریم. ۱(۱۰)

دو ویدیو یک دقیقه‌ای کاربردی (استفاده از ۲ برای فرمت استرینگ):

[https://www.youtube.com/shorts/ZwdA\\_0Tjzc0](https://www.youtube.com/shorts/ZwdA_0Tjzc0)

<https://www.youtube.com/shorts/3yyNvsLnHE8>

**مثال:** برنامه‌ای بنویسین که یه رمز عبور از کاربر بگیره و اگر رمز عبور برابر استرینگ ۱۲۳۴ بود،

بنویسه OK.

**پاسخ:**

```
password = input('Enter password: ')
if password == '1234':
    print("OK")
```

حواستون باشه که توی if، برای چک کردن مساوی بودن، دوتا مساوی می‌ذاریم. یه اشتباه رایج هست که ممکنه همینطوری اشتباهی یک فاصله بذارین.

یکم بریم جلوتر باز.

من شاید بخوام بگم اگر شرط برقرار نبود، فلان کار کن.

```
password = input('Enter password: ')
if password == '1234':
    print("OK")
else:
    print("Not OK")
```

کلمه else یعنی در غیر این صورت.

میگیم اگر if برقرار نبود، در غیر این صورت، بیا کارهای زیر رو انجام بده که با دو نقطه گفتم کدوم کارها.

دقت کنین که دو نقطه رو یادتون نره.

دقت کنین کارهایی که باید انجام بشه، چهارتا فاصله دارن.

دقت کنین else دقیقاً زیر if نوشته شده و indentation اضافی نداره!

بیایم یکم دیگه پیچیده‌ترش کنیم.

فرض کنین همینو نگه داریم ولی بگیم اگر password برابر ۱۲۳۴ نبود، چک کن ببین برابر

استرینگ admin هست یا نه؟ اگر بود. بنویس OK. و خط بعدیش هم بنویس. خوش آمدید.

```
password = input('Enter password: ')
if password == '1234':
    print('OK')
elif password == 'admin':
```



```

print('OK')
print('Welcome!')
else:
print('Not OK')

```

این کار رو با elif که مخفف else if هست انجام میدیم. یعنی «در غیر این صورت، اگر»  
یعنی اول میاد if رو چک می‌کنه، اگر if برقرار بود که هیچ و کارهایی که با چهارتا فاصله زیر if نشون داده شدن رو انجام میده.

اگر نه، حالا یه شرط دیگه! اگر password برابر استرینگ admin بود، کارهای زیر elif که زیرش و با چهارتا فاصله از اون مشخص شدن رو انجام بده. یعنی: OK رو چاپ کنه و یه بعدش یه Welcome هم چاپ کنه.

اگر هیچ‌کدوم از اون شرط‌ها برقرار نبود یا همون در غیر این صورت، بدون هیچ شرطی چک کردن، چاپ کنه Not OK.

دقت کنین else هیچ شرطی چک نمی‌کنه و تنها زمانی اجرا میشه که هیچ if و elif ای اجرا نشه.  
در واقع elif هم صرفاً زمانی اجرا میشه که شرطی قبل خودش انجام نشده باشن.

توجه! ما می‌تونیم بعد یک if، بی‌نهایت elif داشته باشیم، ولی در نهایت حداکثر یه else خواهیم داشت. (می‌تونیم else رو هم اصلاً نذاریم!)

### تمرین!

- ۱- برنامه‌ای بنویسین که یه سن از کاربر بگیره. اگر بالاتر از ۱۸ بود، چاپ کنه You're over 18  
اگر برابر ۱۸ بود، چاپ کنه You're 18  
اگر کوچکتر از ۱۸ بود، چاپ کنه You're below 18
- ۲- برنامه‌ای بنویسین که سه عدد از کاربر بگیره و بزرگترینشون رو نمایش بده.
- ۳- برنامه‌ای بنویسین که یه نام و یه سن از کاربر بگیره و چک کنه ببینه اگر اسم کاربر kourosh بود و یا سنش برابر ۳۵ نبود، چاپ کنه Hello وگرنه چاپ کنه Bye.

۴- توی خارج نمره‌ها اینطورین:

[20, 16] → A  
[14, 16] → B<sup>35</sup>  
[12, 14] → C  
[10, 12] → D  
[0, 10] → F

---

۳۵ یعنی درواقع ۱۴ تا ۱۵.۹۹. پرانتز یعنی خود ۱۶ حساب نشه. کروش یعنی بشه.

خب سؤال به شما نمره می‌ده و شما باید بگین کدوم هست؟ A هست؟ یا B؟ یا ...؟

۵- فرض کنید که سیستمی داریم که رمزش رشته (string) «۱۲۳۴» هست. کاربر حداکثر ۳ بار فرصت داره که رمز رو درست بزنه. اگر تا اون مدت زد، بهش بگین «Welcome»، اگر نه، بهش بگین «Blocked». (اگر مفهوم حلقه رو بلدین، از حلقه استفاده نکنید!)

## ورودی

حداکثر شامل ۳ خط. (ورودی رمز)

## خروجی

در صورت درست بودن رمز در مدت معلوم، «Welcome» در غیر این صورت، «Blocked».

input:

1234

output:

Welcome

-----

input:

hi

Hello

admin

output:<sup>36</sup>

Blocked

-----

input:

hi

Hello

1234

output:<sup>37</sup>

Welcome

-----

توجه! اگر سه بار اینتر زد (۳ بار استرینگ خالی داد)، پس خب ۳ بار تلاششو کرده و بلاک باید بشه!

۶- ۴ تا عدد بهتون داده میشه. شما باید ماکزیمم و تعداد تکرار ماکزیموم رو به دست بیارین.

input:

\_\_\_\_\_

۳۶ ۳ بار تلاششو کرد و همش اشتباه! بگین Blocked.  
۳۷ ۳ سوم درست زد و اوکیه.

1

4

9

3

output:

9 1

-----

input:

9

-1

10

10

output:

10 2

-----

input:

3

3

4

1

output:

4 1

-----

input:

3

3

3

3

output:

3 4

۷- دو کد زیر رو در نظر بگیرین. تفاوتشون رو پیدا کنین و فکر کنین که در چه صورتی، هر کدوم از

if، elif، else ها اجرا میشن؟

code1:

```
char = input()
```

```
if char == 'a':
```

```
    print('a')
```

```
    print('a')
```

```
elif char == 'b':
```

```
    print('b')
```

```
else:
    print('else block')
```

code2:

```
char = input()
if char == 'a':
    print('a')
    print('a')
if char == 'b':
    print('b')
else:
    print('else block')
```

۸- دولت نورلند (Neverland) به تازگی طرح جدید سهمیه بنزین را با افزایش قیمت غیرمنتظره اعلام کرده است. طبق برنامه جدید، هر نفر سهمیه‌ای معادل ۶۰ لیتر در ماه در کارت سوخت دریافت می‌کند. هر لیتر بنزین در صورت سهمیه‌بندی ۱۵۰۰ اسلوب هزینه دارد. هرگونه سوخت اضافی (آزاد) ۳۰۰۰ اسلوب به ازای هر لیتر هزینه دارد. همچنین سهمیه‌های اضافی هر ماه، برای ماه بعد هم محفوظ است.

به شما سهمیه اضافی از ماه قبل و مصرف این ماه داده می‌شود. شما باید هزینه‌ای که بابت بنزین داده می‌شود را حساب کنید.

#### ورودی:

ورودی شامل دو خط است.

خط اول مقدار مصرف ماه.

خط دوم مقدار سهمیه اضافی از ماه قبل.

#### خروجی:

شامل یک خط و م میزان هزینه کل بنزین مصرفی به اسلوب.

input:

41

0

output:<sup>38</sup>

61500

-----

input:

125

40

---

۳۸ مصرف ۴۱ لیتر. مقدار اضافه‌مونده از ماه قبل ۰ لیتر. و ۴۱ لیتر رو میشه از سهمیه‌ای ۱۵۰۰ اسلوبی که هر ماه برای ۶۰ لیتر داریم، داد. پس:  
 $41 * 1500 = 61500$

output:<sup>39</sup>

225000

-----

پاسفنامه:

پاسخ ا:

```
age = int(input("Enter your age:\n"))
if age > 18:
    print("You're over 18")
elif age == 18:
    print("You're 18")
else:
    print("You're below 18")
```

دقت کنین که برای چک کردن شرط مساوی، دوتا علامت مساوی می گذاریم. این یه اشتباه رایجه که افراد موقع شرط چک کردن، به اشتباه یه علامت مساوی می ذارن! این اشتباه رو انجام ندین! شرط، دوتا مساوی. انتساب دادن، یک مساوی!

خط یکی مونده به آخر (به جای else) می تونستیم اینطوری هم بنویسیم:

```
elif age < 18:
```

البته فرق هم نداشت! چون حالت دیگه ای نبود، نیاز به چک کردن شرط نبود. چون یا بزرگ تر هست یا مساوی یا آخرین گزینه که اگر بقیه نبود، کوچکتتر. پس اگر دوتای قبلی نبود، قطعاً کوچکتتر بود. پس نیازی هم نبود به چک کردن!

**متوسط:** کامپیوتر برای انجام هر کار یه مقداری زمان صرف می کنه و چه بهتر که کارهایی که می کنیم و ارزش می خوایم کمتر باشه. نگیم در غیر این صورت بیا شرط چک کن. وقتی نیاز به چک کردن شرط نیست، صرفاً انجامش بدیم! الکی شرط چک نکنیم که کامپیوتر بخواد یه زمانی هم برای اون شرط چک کردن سپری کنه.

درواقع ما سعی کردیم برنامه رو بهینه تر و سریع تر کنیم. به روایتی performance برنامه رو بالا بردیم!

نکته!

شاید دیدین که من توی ورودی گرفتیم، نوشتیم \n یا همون کرکتر new line. خب یه خرده قشنگ ترش کردم. یعنی حالت عادی ورودی رو عیناً جلوی متن input می گرفتیم ولی الان یه اینتر میزنه و ورودی رو در خط بعد می گیره. شاید گاهی این چیزا تمیزتر به نظر بیاد!

<sup>39</sup> مصرف ۱۲۵ لیتر. مقدار اضافه مونده از ماه قبل، ۴۰ لیتر. پس ما درواقع ۶۰ + ۴۰ = ۱۰۰ لیتر این ماه سهمیه داریم. پس ۱۰۰ لیتر رو با سهمیه ۱۵۰۰ اشلوبی و ۲۵ لیتر دیگر رو با سهمیه ۳۰۰۰ اشلوبی می دیم. یعنی:

$$100 * 1500 + 25 * 3000 = 225000$$

راستی! علامت دونقطه به کلمه قبل می‌چسبه و از کلمه بعدیش فاصله می‌گیره. (درست مثل کاما! یادتونه دیگه!) مثلاً:

Correct: `age = int(input("Enter your age: "))`

Wrong: `age = int(input("Enter your age:"))`

چون دومی هنگام ورودی گرفتن، ورودی کاملاً می‌چسبه به دو نقطه و این زبا نیست!

## پاسخ ۲:

این سؤال رو با عمد آوردم که شما رو با عملیات‌های منطقی (logical) آشنا کنم. خط فکری:

من باید چک کنم و بگم مثلاً اگر عدد اول بزرگ‌تر از عدد دوم بود، بعدش چک کن ببین عدد اول بزرگ‌تر از سومی هست؟ اگر هست یعنی بزرگ‌ترین و چاپش کن. ولی خب اگر عدد اول بزرگ‌تر از عدد دوم نبود چی؟ یعنی عدد دوم بزرگ‌تر یا مساوی اولیه. پس یه شرط دیگه می‌گذارم که اگر شرط اولی برقرار نبود، برنامه چک کنه و ببینه عدد دومی که به واسطه بزرگ‌تر بودن از اولی، اومده توی این قسمت، آیا از عدد سوم بزرگ‌تره یا نه؟ اگر بزرگ‌تره، خب پس یعنی بزرگ‌ترین. پس چاپش کن. اگر هیچکدوم از شرطای اصلی و کلی بالا درست نبود (درغیر این صورت)، یعنی سومی بزرگ‌ترین و خب سومی رو چاپ کنه.

سعی کنین خودتون با راهنمایی و خط فکری که گفتم انجامش بدین و بعد نگاه پایین کنین.

```
if num1 > num2:
    if num1 > num3:
        print(num1)
    elif num2 > num3:
        print(num2)
else:
    print(num3)
```

توضیح: اول چک می‌کنه ببینه عدد اول آیا از دومی بزرگ‌تره یا نه. اگر آره بیاد پایین و یه سری کارها رو انجام بده. چه کارهایی؟ کارهایی که با چهارتا فاصله و دو نقطه بالاش مشخص شدن. یعنی چک کنه آیا عدد اول از سومی بزرگ‌تره یا نه. اگر هست، عدد اولی رو چاپ کنه. خب اگر نبود، می‌بینه هیچی زیرش نیست و هیچ `else` و یا چیزی نیست. میره خط بعدیش که کلاً از چهارتا فاصله زیر `if` اصلی و بیرونی ما (یعنی همون `num1 > num2`) خارج شده و به یه `elif` بر می‌خوره.

نکته! این زمانا می‌گن از بلاک (block) `if` بیرونی (خارجی‌تر) ما بیرون اومده. درواقع از اون چهارتا فاصله‌های زیرش که متعلق بهش بودن یا اصطلاحاً `block` اش بودن، بیرون میاد!

بعدش میره سراغ elif. گفتیم elif زمانی اجرا میشه که if بالاییش که دقیقاً بالاشه (یعنی if بیرونی ما)، اجرا نشده باشه. یعنی شرط چک شده ولی بینه عه شرط برقرار نیست. معنی elif هم همین بود دیگه! «در غیر این صورت، اگر»

حالا اگر if برقرار نشده بوده، یعنی چی؟ یعنی  $num2 > num1$  بوده! پس حالا صرفاً نیازه چک کنیم ببینیم که آیا  $num2 < num3$  هست یا نه؟ اگر بود یعنی بزرگترین و چاپش کن.

اگر هیچکدوم از if های بیرونی ما و elif ها برقرار نبود، یعنی قاعدتاً عدد سومی بزرگترین و بدون هیچ شرط چک کردنی، عدد سومی رو چاپ می کنیم.

اگر متوجه نشدین، روی کد فکر کنید و تا صد درصد مطمئن نشدین که فهمیدین، ادامه رو نخونین! گاهی اوقات صرفاً فکرتون میگه اینکه سادس بابا فهمیدم. ولی واقعاً درکش نکردین! اگر می خواین مطمئن شین فهمیدین، دو روز بعد سعی کنید همین الگوریتم رو خودتون بنویسین. اگر نتونستین بنویسین، یعنی درسته فهمیدین ولی اگر نتونستین، یعنی نفهمیدین!

برنامه نویسی مثل ریاضیات! نمیتونین با خوندنش بگین عه فهمیدم. صرفاً فکرتون که فهمیدین. ولی در اصل نفهمیدینش.

از من به شما نکته!

اگر می خواین برنامه نویسیتون قوی **نشو**، صرفاً بخونین و رد شین و تلاش برای حل **نکنین**! پاسخ رو خودتون بخونین و تلاشی برای حل موضوع نکنین!

یا اگر خواستین تلاش کنین، دو دقیقه شد و نتونستین، دست از تلاش بکشین و پاسخ نامه رو بخونین! این کار تضمین می کنه که شما برنامه نویسیتون ضعیف باقی بمونه (:

## روش دوم:

ما به جای اینکه تعداد if هامون رو زیاد کنیم و بگیم اگر فلان برقرار بود، بیاد دوباره زیرش if چک کن (کاری که توی if اولی کردیم و زیرش باز if چک کردیم)، می تونیم دو یا چند تا شرط رو همزمان توی یک if چک کنیم.

می تونیم بگیم اگر فلان چیز و (and) فلان چیز برقرار بود، فلان کار کن.

پایتون گفته من کارتون رو راحت می کنم! مثل زبون گفتاری که می گی اگر فلان چیز و فلان چیز برقرار بود، اگر فلان چیز یا فلان چیز حداقل یکیشون برقرار بود و... رو همش برات آوردم (: کارت سادس!

اپراتور	مثال	توضیح
and	$\text{if } x > y \text{ and } x > z$	اگر $x$ بزرگتر از $y$ بود و همچنین $x$ بزرگتر از $z$ بود (باید هر دو شرط برقرار باشه تا بره زیرش و کارهایی که توی بلاکش هست رو انجام بده).
or	$\text{if } x > y \text{ or } x > z$	اگر $x$ بزرگتر از $y$ بود یا $x$ بزرگتر از $z$ بود. (هرکدوم از این دو شرط برقرار باشه، اوکیه و میره تو بلاکش)

not	if not (x > y)	اگر X بزرگتر از Y بود، حالا برعکسش کن ببین اگر برعکس درست بود (یعنی X کوچکتر از Y بود)
-----	----------------	--

خب سعی کنین با چیزایی که بهتون یاد دادم، پاسخ قبلی رو با استفاده از اپراتورها بنویسین.  
پاسخ:

```
if num1 > num2 and num1 > num3:
    print(num1)
elif num2 > num3:
    print(num2)
else:
    print(num3)
```

خیلی قشنگ تر نشد؟  
یا حتی حالت زیر هم درسته:

```
if 4 <= num < 6:
```

یعنی اگر num بزرگتر یا مساوی ۴ و کوچکتر از ۶ بود...  
- من الگوریتم متفاوتی رو نوشتم. از کجا بدونم درسته یا نه؟  
+ اول به صورت منطقی بررسیش کنین ببینین درسته یا نه. خط به خط از بالا تا پایین بخونین و ببینین کار درست رو انجام میده.

درواقع حالتای مختلف رو بررسی کنین. بگین اگر اینجا شرط برقرار شه چی میشه؟  
اگر برقرار نشه چی میشه؟!  
عین یه درخت توی ذهنتون پیش برین. بگین خب اگر برقرار شه میاد اینجا. اگر برقرار نشه میره فلان جا. حالا اون جای دوم اگر برقرار شه شرطش فلان میشه و همینطور فکرتون رو گسترش بدین.

دوم: برنامه رو تست کنین!  
الکی تست نکنین! بلکه هدفمند تست کنین. یعنی چی؟  
یعنی حالتای تست (test case ها) رو گروه بندی کنین. یعنی بگین ممکنه یه حالت num1 بزرگترین باشه. یه حالت num2 بزرگترین باشه. یه حالت هم num3. یه حالت هم حالتی که مساوی ممکنه باشن با هم.

درواقع حالت بندی کنین ببینین چه حالتایی ممکنه پیش بیاد و بر اون اساس test case بسازین!  
یعنی یه بار بزرگترین عدد رو بدین به num1. یه بار به num2 و یه بار به num3. یه بار num1 و num2 مساوی باشن با هم. یه بار num1 و num3 و یه بار num2 و num3 و یه بار هم همش با هم مساوی. از این گروه حالتا که خارج نیست!  
پس صرفاً نیازه که برای هر گروه یه مثال بزنین که تقریباً مطمئن شیم که کدمون درسته.  
- چرا تقریباً؟



+ چون ممکنه حواسمون نباشه یه حالت رو جا انداخته باشیم. بالاخره ذهن آدمی هست دیگه. ممکنه یه حالت رو یادش بره! یا ممکنه برنامه یه باگ بخوره که حالا عجیبه. همیشه برنامه ممکنه همچین مشکلاتی پیش بیاد و وظیفه ما به عنوان یه برنامه‌نویس، تهیه برنامه‌ای هست که حداقل منطقی مشکلی نداشته باشه!

بذارین یه مثال بزنم. سه مثال زیر واقعاً فرقی ندارن و الکی سه تا مثال زدین و وقتتون هدر دادین! بلکه می‌تونستین با یه مثال مطمئن شین این حالت خاص درسته.

num1: 3	num1: 10	num1: 100
num2: 2	num2: 7	num2: 90
num3: 1	num3: 5	num3: 10

چون همش مربوط به یه گروه خاصن. همون گروهی که num1 بزرگ‌ترین. num2 متوسط و num3 کوچک‌ترین. پس سعی کنین که هدف‌مند انتخاب کنین که از گروه test case های متفاوت باشن. گاهی اوقات هم حالتا اونقدر زیاده که نمیشه تستشون کرد. ولی خب حداقل حالتای معروف رو تست کنین.

پاسخ ۳:

```
if name == 'Kourosh' or age != 35:
    print("Hello")
else:
    print("Bye")
```

روش دوم:

سعی کنین با کمک not پیاده‌سازی کنین!

پرانته‌گذاری کار خیلی خوبیه که کدتون رو قشنگ‌تر کنه. مثلاً دو شرط اصلی که بینشون and بود، توی پرانته‌زار قرار گرفتن.

بعدش برای not، خواستم بگه نات عبارت داخل پرانته‌زار. که خواننده کد براش ملموس‌تر باشه که نات چی هست.

- من نام رو کوروش وارد می‌کنم و سن رو یه چیزی به جز ۳۵ وارد می‌کنم ولی بهم Hello رو نشون نمیده! چرا؟

+ شاید هنگام اجرای برنامه، «K» توی «Kourosh» رو کوچیک وارد کردی! پایتون به بزرگی و کوچیکی حروف حساسه! یعنی «Kourosh» با «kourosh» براش متفاوت.

- خب اینکه خیلی بده! شاید کاربر حواسش نباشه و اسمشو اشتباهی با حروف کوچیک یا بزرگی که با چیزی که من نوشتم متفاوت وارد کنه. اینطوری برنامه درست کار نمی‌کنه!

+ درست می‌گین! اما بعداً راهکار میدم بهتون که تمام چیزها رو کوچیک در نظر بگیره و حروف بزرگ و کوچیکی نباشه! همش کوچیک باشه و این اشتباهات پیش نیاد! فعلاً فرض کنین کاربر همه چیز رو درست وارد می‌کنه.

وگرنه اگر کاربر می‌خواست اشتباه وارد کنه، موقع وارد کردن سن، شاید دستش می‌خورد به جای عدد، یه حرف وارد می‌کرد و اون موقع برنامه باگ می‌خورد! اما فعلاً فکرمون اینه که کاربر همه چیز رو درست و عیناً چیزی که می‌خواهیم وارد می‌کنه!

پاسخ ۴:

روش اول:

```
grade = float(input())
if grade >= 16 and grade <= 20:
    print("A")
elif grade >= 14 and grade < 16:
    print("B")
elif grade >= 12 and grade < 14:
    print("C")
elif grade >= 10 and grade < 12:
    print("D")
elif grade >= 0 and grade < 10:
    print("F")
```

روش دوم:

مثل ریاضیات، اینجا هم می‌تونیم بگیریم اگر متغیر بین دو عدد فلان و فلان بود، فلان کار کن.

```
grade = float(input())
if 16 <= grade <= 20:
    print("A")
elif 14 <= grade < 16:
    print("B")
elif 12 <= grade < 14:
    print("C")
elif 10 <= grade < 12:
    print("D")
elif 0 <= grade < 10:
    print("F")
```

پاسخ ۵:

```
inp_pass = input()
if inp_pass != '1234':
    inp_pass = input()
```

```

if inp_pass != '1234':
    inp_pass = input()

if (inp_pass != '1234'):
    print('Blocked')
else:
    print('Welcome')

```

یه ورودی می‌گیرم. چک می‌کنم ببینم درسته یا نه. اگر نبود یه دونه به تعداد تلاش‌ها باید اضافه کنم دیگه. بعدش دوباره توی همین if ورودی می‌گیرم. چرا توش؟ چون در صورتی که اشتباه بود باید ورودی بگیرم. وگرنه اگر درست بود که مشکلی نداریم!

بعدش دوباره یه if باید بذارم که ببینم درست شد یا نه؟ اگر نشد باز توش ورودی می‌گیرم. این ورودی بار آخره! حالا با یه if نهایی، چک می‌کنم اگر ایندفعه هم درست نبود، خب پس بلاکه. وگرنه می‌گیم Welcome.

کد ما توی ۳ تلاشی که فرصت داره، اولین ۱۲۳۴-ای که ببینیم! می‌گه Welcome. یه بار تلاش کنین کد رو از بالا به پایین بخونین و چکش کنین. مثلاً بگیرم ورودی اینه:

```

hi
1234

```

خب if اولی می‌بینیم که درست نیست و دوباره ورودی می‌گیره. اما توی if دومی، می‌بینیم درسته! پس واردش نمیشه. همچنین وارد if سومی هم نمیشه. چون inp\_pass برابر ۱۲۳۴ هست. پس میره if آخر. بازم درست نیست. اما اینجا else داریم. پس میره توی else و پرینت می‌کنه Welcome.

پاسخ ۶؛

```

inp_num = int(input())
maximum = inp_num
count = 1

inp_num = int(input())
if inp_num > maximum:
    maximum = inp_num
    count = 1
elif inp_num == maximum:
    count = count + 1

inp_num = int(input())
if inp_num > maximum:
    maximum = inp_num

```

```

count = 1
elif inp_num == maximum:
    count = count + 1

inp_num = int(input())
if inp_num > maximum:
    maximum = inp_num
    count = 1
elif inp_num == maximum:
    count = count + 1

```

```
print(maximum, count)
```

همیشه بدونین که وقتی می‌خوایم ماکزیمم یا مینیمم رو به دست بیاریم، شما عدد اول رو بگیرین و بذاریدش داخل max و min. دقیقاً کاری که اینجا کردم. همون اول که گرفتمش، گذاشتمش توی maximum.

می‌خوایم تعداد تکرار رو به دست بیاریم. برای این کار، نیازه که یه متغیر داشته باشیم که هر بار ماکزیمم تکراری دیدیم (عدد جدیدی که با ماکزیمم قبلی برابره)، یکی بهش اضافه کنیم. اسم این متغیر رو می‌ذارم count.

کار اصلی ما از بعد این شروع میشه. باید هر بار عدد بگیریم، ببینیم اگر بزرگ‌تر از ماکزیمم قبلی ماست، خب از این به بعد، عدد جدید رو می‌ذاریم توی مکس که عملاً مکسیمم برابر عدد جدید شه. حتماً حواسمون هست که تعداد تکرار رو باید دوباره ریست کنیم. وگرنه قاطی میشه. چون می‌گیم برای این مکس جدید، تکرار ریست شه و برابر ۱ شه.

حالا اگر بزرگ‌تر نبود، چک کن بین یکسانه؟ اگر یکسان (تکراری) بود، یکی به count اضافه کن. این روند رو تکرار می‌کنیم تا ۴ عدد گرفته شن. در آخر هم maximum و count رو چاپ می‌کنیم.

پاسخ ۷:

کد ۱: کد اولی یه if داره. می‌گه یه استرینگ بگیر. اگر استرینگ برابر کرکتر «a» بود، بیا دو کار زیر رو انجام بده. دو تا پرینت کن.

بعدش می‌گه elif (یعنی اگر if برقرار نبود، بیا این شرط رو چک کن)، اگر کرکتر «b» بود، پرینت کن «b».

بعدش می‌گه else (وگرنه اگر هیچ کدوم از if و elif هام اجرا نشدن)، پرینت کن «else».

کد ۲: فرقی اینه که به جای elif، عملاً یه شرط جدید رو داره شروع می‌کنه. این شرط جدید دیگه مثل elif (که مخفف if else هست)، دیگه هیچگونه وابستگی به if بالایی نداره. پس فارغ از اینکه if

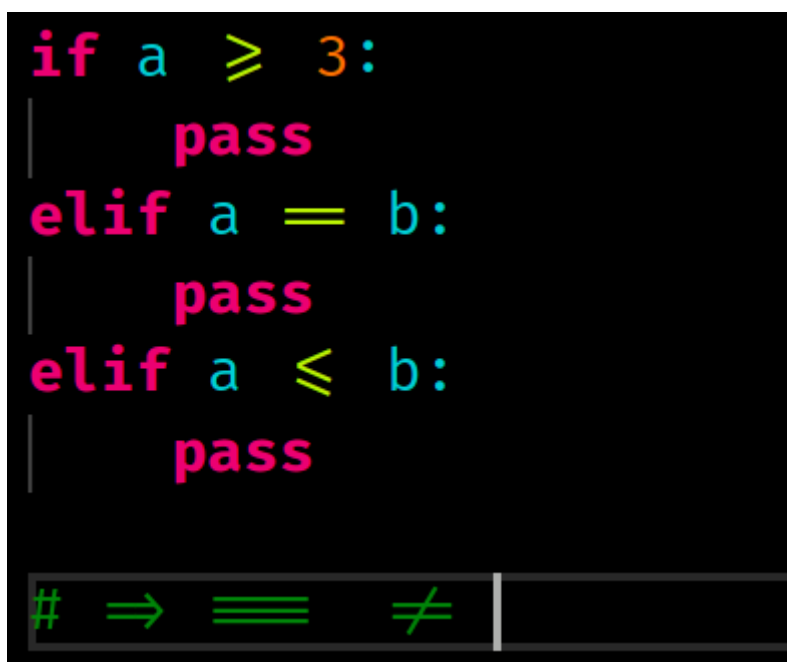
بالایی اجرا شه یا نشه، این اجرا میشه و `else` پایانی هم فقط و فقط مختلف `if` دومی هست. یعنی `if` ای که دقیقاً بالا سرشه!

### پاسخ ۸:

صرفاً نیازه یه شرط بذاریم بگیریم که اگر جمع سهمیه اضافی ماه قبل و سهمیه این ماه (۶۰ تا)، کمتر از مقدار مصرفی بود، خب تمامش با ۱۵۰۰ اشلوب حساب شه. اگر نه، بیشتر بود، به میزان جمع سهمیه اضافی ماه قبل و این ماه (۶۰ لیتر)، با ۱۵۰۰ اشلوب و بقیش با ۳۰۰۰ اشلوب حساب شه:

```
usage = int(input())
add_quota = int(input()) # additional quota from the last month
monthly_quota = 60
total_quota = add_quota + monthly_quota
if usage <= total_quota:
    price = usage * 1500
else:
    price = (usage - total_quota) * 3000 + (total_quota) * 1500
print(price)
```

علامتای `==` و `!=` قشنگ نیستن؟ دوست دارین شکلشون اینطوری شن؟



```
if a ≥ 3:
    pass
elif a = b:
    pass
elif a ≤ b:
    pass
```

# ⇒ ≡ ≠

از قابلیت Ligature استفاده کنین.

توی هر IDE یا text editor ای فرق داره ولی برای VS Code اول نیازه یه افزونه به نام Fira Code نصب کنین و بعد یه سری فونت میاره باید دستی نصبشون کنین و بعدش توی تنظیمات `Json`،

```
"editor.fontFamily": "Fira Code",
```

```
"editor.fontLigatures": true
```

رو وارد کنیم.

برای رفتن به فایل json (تنظیمات code editor)، روی کنترل کاما بزنیم و بعد بالا سمت راست دومین دکمه کنار سه نقطه، به گزینه هست که اگر روش بگیریم نوشته: Open Settings (JSON)

## • Nested if

ما می‌تونیم توی یه if، یه if دیگه داشته باشیم! یعنی بگیریم اگر اومدی توی این if، می‌تونی یه if دیگه هم داشته باشی! مثلاً:

```
num = int(input())
if num > 20:
    print('num is bigger than 20')
    if num % 2 == 0:
        print('even')
    print('end of the outer if')
```

یعنی می‌گیریم اول چک کن ببین عدد بزرگ‌تر از ۲۰ هست، اگر بود بیا داخل if اولی. یه پرینت انجام بده. بعدش باز همینجا داخل if، ببین آیا زوج؟ اگر زوج بود، پرینت کن even. بعدش بیا بیرون از if دومی. اما حواستون باشه که هنوز توی if اولی هستیم و باید پرینت «end of the outer if» رو انجام بدیم.

حتی می‌تونیم همون elif و else هم برای این if ها پیاده‌سازی کنیم. هر if، می‌تونه else مخصوص به خودش رو داشته باشه. فقط حواستون باشه که عیناً زیرش باشه و فاصله‌ای اضافه یا کم نذاریم:

```
num = int(input())
if num > 20:
    print('num is bigger than 20')
    if num % 2 == 0:
        print('even')
    else:
        print('odd')
    print('end of the outer if')
else:
    print('num is smaller than 20')
```

حتی می‌تونیم توی else هم یه if قرار بدیم!

```
num = int(input())
if num > 20: # the outer if
    print('num is bigger than 20')
```

```

if num % 2 == 0:
    print('even')
else:
    print('odd')
print('end of the outer if')
else:
    print('num is smaller than 20')
    if num % 2 == 0:
        print('even')
    else:
        print('odd')
    print('end of the outer else')

```

معمولاً به خارجی‌ترین if می‌گیم if لایه اول. و به موازات اون، هر لایه اضافه شه، میشه مثلاً if لایه دوم. مثلاً اینجا ما دو لایه داریم. خارجی‌ترین if لایه اول هست و داخلی‌ترین، لایه دوم.

### تمرین:

۱- برنامه ای بنویسید که سه عدد ورودی بگیرد. اگر حداقل یکی از این سه عدد حداقل سه رقمی بود، در خط اول بزرگترین و در خط بعد از آن کوچکترین عدد چاپ شود. در صورتی که هیچ کدام از اعداد سه رقمی نبودند، بر اساس جدول زیر نشان دهد که جمع رقم یکان (رقم آخر از سمت راست) هر سه عدد با هم در کدام بازه قرار می‌گیرد.

خروجی	بازه
A	کمتر از ۳
B	از خود ۳ تا خود عدد ۶
C	بزرگ‌تر از ۶

### ورودی:

در این قسمت سه عدد وارد می‌شوند. (تضمین می‌شود اعداد ورودی مثبت هستند).

### خروجی:

با توجه به توضیحات، نتیجه چاپ می‌شود.

### راهنمایی:

برای اینکه ببینین یه عدد حداقل ۲ رقمی هست، باید تقسیم صحیحش بر ۱۰ بزرگ‌تر از ۰ شه. برای اینکه ببینین یه عدد حداقل ۳ رقمی هست، باید تقسیم صحیحش بر ۱۰۰ بزرگ‌تر از ۰ شه.

```

num1 = int(input())
num2 = int(input())
num3 = int(input())
if (num1//100 > 0) or (num2//100 > 0) or (num3//100 > 0):
    if (num1 >= num2) and (num1 >= num3):
        print(num1)
    if num2 < num3:
        print(num2)
    else:
        print(num3)
    elif (num2 >= num1) and (num2 >= num3):
        print(num2)
    if num1 < num3:
        print(num1)
    else:
        print(num3)
    elif (num3 >= num1) and (num3 >= num2):
        print(num3)
    if num1 < num2:
        print(num1)
    else:
        print(num2)
else:
    lastdigit_sum = (num1 % 10) + (num2 % 10) + (num3 % 10)
    if (lastdigit_sum >= 0) and (lastdigit_sum < 3):
        print('A')
    elif (lastdigit_sum >= 3) and (lastdigit_sum <= 6):
        print('B')
    else:
        print('C')

```

## • While



خب خب! فرض کنیم شما می‌خواهید وارد حساب کاربریتون بشین. شما یه نام کاربری (username) و یه رمز رو می‌زنین. اگر درست بود، به شما میگه درسته وارد شو. اگر نبود، یه متنی رو چاپ می‌کنه که بگه اشتباه وارد کردی. کدشو هم توی if زدیم. یه همچنین چیزی بود:

```
inp_pass = input('Enter password: ')
system_pass = '1234'
if inp_pass == system_pass:
    print('Welcome')
else:
    print('Wrong password')
```

اما همونطور که می‌دونیم، این برنامه ما، بعد از یه بار اشتباه‌زدن، به پایان می‌رسه. یعنی انگار برنامه تموم شده. مثل برنامه‌ای هست که شما دکمه بسته‌شدنش رو زدن. دیگه کار نمی‌کنه! اما همونطور که ما می‌دونیم، توی وبسایتی که اینطور نیست! یعنی اگر پسورد رو اشتباه بزنیم، بازم از ما پسوردی گیره و تا جایی که اشتباه بزنیم، هر بار میگه اشتباه زدی و پسورد دیگه‌ای رو امتحان کن. اینجا ما نیاز به ساختار if ای که تکرارپذیره داریم! یعنی هم شرطی باشه و هم مثلاً بتونیم بگیریم که تا وقتی که پسورد درست رو نزده بود، نذار از این مرحله عبور کنه. اینجا یه ساختاری به نام while برای ما محیا شده.

while به معنای «تا وقتی که» هست. از اسمش معلومه یعنی تا وقتی که یه چیزی می‌خواهیم برقرار باشه، یه سری کارها رو باید واسمون انجام بده:

تا وقتی که {شرط}؛

...

...

یا درواقع:

```
while {condition}:
```

```
...
```

```
...
```

مثلاً من می‌خوام بگم اول یه عدد از کاربر بگیر، بعدش تا وقتی که عدد بزرگ‌تر از صفر هست، عدد رو منهای یک کن. بعد چاپ کن Hello. در پایان برنامه هم چاپ کنه End. درواقع به این میگن حلقه (loop). یعنی برنامه رو توی یه حلقه می‌ندازم که هی یه سری کارها رو انجام بده. تا کجا؟ تا وقتی که دیگه شرط برقرار نباشه و اصطلاحاً از حلقه بپره بیرون. این رو اینطور می‌نویسن:

```
number = int(input("Enter a number: "))
while number > 0:
    number = number - 1
    print("Hello")
print("End")
```

خب بیایم تحلیل کنیم. بعد گرفتن عدد، میگم تا وقتی که عدد ما بزرگ‌تر از ۰ هست، بیا یه سری کارها رو انجام بده. - چه کارهایی؟

+ کارهایی که با دو نقطه و چهارتا فاصله جلوتر مشخص کردم.  
یعنی اول بیاد یکی از عدد کم کنه. (این رو اینطوری انجام میدم که عدد رو منهای یک کنه و دوباره بگذاره توی خود عدد)

بعدش چاپ کنه Hello

مثلاً بیایم یه مثال بزنیم:

عدد ۳ رو میدیم.

آیا ۳ کوچکتر از ۰ هست؟ بله! پس باید کارها رو انجام بده. یعنی اول میاد number رو منهای یک می کنه و بعد می گذاره توی number. (یعنی درواقع number ما برابر ۲ میشه. بعدش چاپ می کنه Hello.

دوباره میره بالا و دوباره شرط رو چک می کنه.

- چرا میره بالا؟ چرا خارج نمیشه؟

+ گفتم چون while یه حلقه هست. یه حلقه و loop که تا وقتی که شرط برقرار باشه، هی کارها رو انجام میده.

حالا چک می کنه که آیا ۲ بزرگتر از ۰ هست؟ بله! پس دوباره کارها رو انجام میده. یعنی number رو یک می کنه و بعد چاپ می کنه Hello.

حالا چک می کنه که آیا ۱ بزرگتر از ۰ هست؟ بله! پس دوباره کارها رو انجام میده. یعنی number رو صفر می کنه و بعد چاپ می کنه Hello. یعنی number رو یک می کنه و بعد چاپ می کنه Hello.  
حالا چک می کنه که آیا ۰ بزرگتر از ۰ هست یا نه؟ خیر نیست! پس حالا از حلقه خارج میشه.  
از حلقه خارج میشه و میره خط بعدی که نوشته چاپ کن End رو.

توجه! End جزء کارهایی نیست که درون حلقه بخواد انجام بشه. پس نباید با چهارتا فاصله بعد while قرار بگیره. چون خطهایی که با چهارتا فاصله بعد while قرار می گیرن، توی while اجرا میشن. ولی ما می خوایم بعد تموم شدن while، کلمه End چاپ شه.

درواقع دیدین؟ ما می تونیم یه عدد بگیریم و به تعداد اون عدد، یه سری کار رو انجام بدیم. این یکی از معروف ترین ساختارهاست. اینکه یه عدد بگیریم و بگیریم تا وقتی که صفر نشده، یه سری کار کن.  
هر بار هم یکی از عدد کم می کنین. می تونیم خود عدد رو هر بار توی while چاپ کنیم که روند تغییرات رو ببینیم:

```
count = int(input())
while count > 0:
    print(count)
    count = count - 1
print('end')
```

یه بار حتماً خودتون اجراش کنین و روند رو دنبال کنین که ببینین چطوری شد که حلقه به تعداد i بار اجرا شد.

**نکته کلیدی:** مهم‌ترین نکته در `while`، نوشتن شرط درست و مناسب هست. مثلاً بیایم این کد رو

بینیم:

```
age = int(input())
while age > 20:
    age += 1
```

ما ورودی ۳۰ رو میدیم. می‌بینیم هیچی چاپ نمیشه و برنامه هم تموم نمیشه! به نظرتون دلیلش چی می‌تونه باشه؟

+ دلیلش هم اینه که گفتیم که تا وقتی که سن بزرگ‌تر از ۲۰ هست، سن رو با یک جمع کن. خب یه نگاه بندازیم بهش. ما ۳۰ رو داده بودیم. ۳۰ بزرگ‌تر از ۲۰ هست. یکی بهش اضافه میشه. میشه ۳۱. بعدش دوباره میاد بالا شرط رو چک می‌کنه. می‌بینه عه هنوزم ۳۱ بزرگ‌تر از ۲۰ هست. دوباره یکی بهش اضافه می‌کنه. میشه ۳۲. درواقع هی سن زیاد میشه و همیشه بزرگ‌تر از ۲۰ هست و تا ابد ادامه پیدا می‌کنه. به این مشکل که خیلی در کدها رایج می‌گن «حلقه بی‌نهایت» یا «infinite loop». اگر برنامه‌تون خیلی طول کشید و یا صدای فن لپ‌تاپتون بلند شد، احتمالاً توی لوپ بی‌نهایت افتادین! با `ctrl + c` می‌تونین متوقفش کنین. یا خود IDE ها دکمه توقف یا `stop` توی منوی بالا دارن.

*یه راه کاربردی برای تشخیص اینکه برنامه کجای لُگ‌ها افتاده:*

شما می‌تونین از `print` استفاده کنین که ببینین کد آیا به یه جایی رسیده یا نه. من معمولاً از پرینت کردن یه سری خط فاصله یا مساوی استفاده می‌کنم که متوجه شم آیا رسیده یا نه:

```
age = int(input())
while age > 20:
    age += 1
```

```
print('-----')
```

همونطور که می‌بینین، اجراش کنیم می‌بینیم که پرینت ما چاپ نشده و عملاً نرسیده به اون خط. پس می‌فهمیم که گیر کرده.

حتی می‌تونیم پرینت رو ببریم داخل `while` که قشنگ‌تر متوجه شیم کجا گیر کرده:

```
age = int(input())
while age > 20:
    age += 1
    print('-----')
```

۴۰ شاید برنامه‌تون رو خیلی بد نوشتین و برای اعداد بزرگ خیلی طول میکشه. به این بعداً می‌رسیم. ولی فعلاً همون چیزی که گفتیم رو بدونین.

می‌بینیم هی داره خط فاصله چاپ میشه. یعنی اینجا گیر کرده توی `while`. (به این میگن لوپ بی‌نهایت).

این تکنیک پرینت خیلی کاربردی. مثلاً می‌خواین بفهمین کدوم `if` یا `elif` اجرا شده. از این می‌تونین استفاده کنین:

```
age = int(input())
if age > 20:
    print('if 1')
if age < 20:
    print('if 2')
if age == 20:
    print('if 3')
```

خلاصه خلاقیت. مثلاً می‌تونین متغیر رو چاپ کنین که ببینین مقدارش فلان جای کد چیه.

مثال:

هرکدوم از `while` های زیر چند بار اجرا میشن؟

code 1:

```
i = -1
while i > 0:
    print(i)
    i = i - 1
print('end')
```

code 2:

```
i = 5
while i > 0:
    print(i)
    i = i - 2
print('end')
```

code 3:

```
i = 1
while i <= 10:
    print(i)
    i = i * 2
print('end')
```

پاسخ:

کد ۱:

اصلاً وارد while نمیشه! چون گفتیم تا وقتی که  $i$  بزرگ‌تر از صفر هست، یه کاری کن. اما  $i$  ما که برابر منفی ۱ هست که! پس اصلاً شرط برقرار نیست و وارد while نمیشه و مستقیم end رو چاپ می‌کنه.

کد ۲:

خب بشمارین! بار اول  $i = 5$ . پس وارد میشه. کار رو انجام میده و  $i = 3$  میشه. دوباره میره بالا چک کنه. هنوز بزرگ‌تر از صفره. پس دوباره میاد کارها انجام بده. ایندفعه  $i = 1$  میشه. دوباره میره بالا. چک می‌کنه میبینه هنوز بزرگ‌تر از صفره. میاد تو. ایندفعه  $i = -1$  میشه. دوباره میره بالا. می‌بینه برقرار نیست. پس میاد بیرون. مجموعاً ۳ بار اجرا میشه.

کد ۳:

خب همینطور حساب کنین که به ازای چه مقادیری از  $i$  وارد while میشه:

1)  $i = 1$

2)  $i = 2$

3)  $i = 4$

4)  $i = 8$

وقتی هم که ۱۶ شد، وارد while نمیشه. پس مجموعاً ۴ بار داخل while اجرا شد.

ما تونستیم یه حلقه بنویسیم که مثلاً به تعداد  $n$  بار اجرا شه. هر بار  $n$  رو کم می‌کردیم. اما این یه راه دیگه‌ای هم داره. یه متغیر با مقدار اولیه صفر تعریف کنیم و هر بار یکی بهش اضافه کنیم. شرط هم بذاریم که «تا وقتی که به  $n$  نرسیده»:

```
n = int(input())
counter = 0
while counter < n:
    print(counter)
    counter = counter + 1
print("Done!")
```

این هم یه راه دیگه. پس تا الان ۲ راه یاد گرفتین برای اینکه یه حلقه رو  $n$  بار تکرار کنین.

+ نکته! از اون and و or و not که توی if یاد گرفتیم، اینجا هم می‌تونین استفاده کنین! چون while عین یه شرطه که کارها رو تا وقتی که شرط برقرار باشه انجام میده.

## تمرین:

لازمه تذکر برم که کم کم سؤالات دارن سفت میشن. پس اگر چند دقیقه فکر کردین و راهی پیدا نکردین، نگران نشین! بلکه فکر کنین! اول باید خودتون تلاش کنین جواب رو پیدا کنین! بدون تلاش، نگاه کردن به پاسخ صرفاً ضرر زدن به خودتونه!

۱- برنامه‌ای بنویسین که یه عدد بگیره و فاکتوریلشو نشون بده.

۲- برنامه‌ای بنویسین که یک عدد به عنوان تعداد دانشجوها بگیرد و سپس به تعداد آن عدد، نمره از ورودی دریافت کند و در نهایت میانگین نمرات را نمایش دهد.

راهنمایی:

برنامه‌ای بنویسین که سه عدد بگیره و بعد گرفتن هر عدد، جمع اعدادی که تا حالا بهش دادیم رو بهمون بده.

یعنی:

input:

1  
2  
3

output:<sup>۴۱</sup>

1  
3  
6

کدش:

```
summation = 0
```

```
num1 = int(input())  
summation = summation + num1  
print(summation)
```

```
num2 = int(input())  
summation = summation + num2  
print(summation)
```

```
num3 = int(input())  
summation = summation + num3  
print(summation)
```

یه متغیر به نام `summation` گرفتیم که پیام جمع رو تا اون لحظه داخلش ذخیره کنم. اینطوری می‌تونم دنبال کنم که تا این لحظه جمع چیه.

---

۴۱ اول ۱ رو دادیم. مجموع اعداد تا حالا ۱ هست. بعدش ۲ رو دادیم. حالا مجموع اعداد تا حالا،  $3 = 1 + 2$  هست. بعدش ۳ رو دادیم. مجموع اعداد،  $6 = 3 + 3$  هست.

حالا فکر کنین که چطور می‌تونم از این تکنیک که هی دونه‌دونه یه عدد بگیرم و مجموع هم یه جا نگه‌دارم، استفاده کنم برای حل مسئله؟

۳- برنامه‌ای بنویسین که همینطور نمره بگیره. تا کجا؟ تا وقتی که کاربر در ورودی عدد ۱- را نزده. در نهایت میانگین نمراتی که گرفته را حساب کند. مثال:

input:

20

15

-1

output:

17.5

-----

input:

20

20

18

9

-1

output:

16.75

-----

input:

-1

output:<sup>42</sup>

-----

۴- برنامه‌ای بنویسین هر بار که کاربر در ورودی عدد ۱- را می‌زند، یک نمره بگیرد. در نهایت میانگین نمراتی که گرفته را حساب کند. شرط پایان while چیست؟ کاربر در ورودی اول، عددی غیر منفی ۱ را بزند. مثال:

input:

-1

20

-1

15

100

output:

17.5

۵- برنامه‌ای بنویسین که اول یه عدد به عنوان  $n$  بگیره و بعد به تعداد  $n$ ، عدد بگیره و بعدش اختلاف عدد ماکزیموم و مینیموم رو به دست بیاره. (  $-1 \leq n \leq 1000$  )

توجه! اگر  $n$  عددی کمتر از ۱ بود، چیزی چاپ نشود!  
برای اینکه ببینین کدتون درست کار می‌کنه یا نه، همیشه حواستون به نقاط مرزی و ورودی‌هایی که ممکنه کد و جوابتون رو خراب کنن باشه. مثلاً:

- اگر همشو صفر بده چی؟
  - اگر اعداد نزولی باشن چی؟
  - اگر صعودی باشن چی؟
  - اگر درهم باشن چی؟
  - اگر ورودی شامل اعداد مثبت و منفی و صفر باشه چی؟
  - اگر صرفاً یه دونه ورودی بده چی؟
  - اگر همه ورودی‌ها یکسان باشن چی؟
- سعی کردم یه سریاشو اینجا بیارم براتون:

input:

4  
100  
20  
1  
-20

output:

120

input:<sup>43</sup>

4  
0  
0  
0  
0

output:

0

input:<sup>44</sup>

2  
1

۴۳ ورودی تماماً یک‌شکل  
۴۴ صرفاً دوتا ورودی و یک‌شکل.



```

1
output:
0
-----
input:45
2
100
20
output:
80
-----
input:46
5
-1
2
3
-100
200
output:
300
-----

```

## تست کیس خودساخته!!!!

- ۶- برنامه‌ای بنویسین که تعداد ارقام فرد یک عدد را حساب کند.
- ۷- برنامه‌ای بنویسین که عددی به عنوان ورودی بگیرد و سپس به تعداد آن عدد، عددهایی ورود بگیرد و مینیموم آن اعداد را نمایش دهد.
- ۸- یادتونه گفتیم برای خوانایی بهتر اعداد، بینشون میتونین «\_» قرار بدین؟ پایتون «\_» ها رو ایگنور می‌کرد.
- وظیفه شما اینه یه برنامه بنویسین که عددی که بهش می‌دیم رو خوانا کنه:

```

input:
123456
output:
123_456
-----
input:
1234
output:

```

۴۵ دو ورودی و متفاوت  
۴۶ ورودی درهم

1\_234

-----

input:

123

output:

123

-----

input:

12

output:

12

-----

پاسفنامه:

پاسخ ۱:

Example:  $5! = 5 * 4 * 3 * 2 * 1$

یعنی هی باید از خودش تا ۱ در هم ضرب بشن.

اول یه متغیر فاکتوریل در نظر می گیریم و همچنین یه ورودی عدد می گیریم:

```
number = int(input('Enter number: '))
```

```
factorial = 1
```

بعدش چون هی باید ضربدر یکی کمترش شه، می تونیم یه حلقه شرطی بگذاریم. بگیریم هی از عدد

یکی کم کن و هی ضرب کن در متغیر فاکتوریل تا حاصل ساخته شه:

```
number = int(input('Enter number: '))
```

```
factorial = 1
```

```
while number > 0:
```

```
    factorial = factorial * number
```

```
    number -= 1
```

```
print(factorial)
```

وقتی عدد ۰ شد، از حلقه می پره بیرون و نتیجه فاکتوریل که توی متغیر factorial ساخته شده رو

چاپ می کنه.

پاسخ ۲:

روش اول:

خب قبول داریم اول باید به عدد به عنوان تعداد دانشجو از کاربر بگیریم؟ پس:

```
student_count = int(input('Enter student count: '))
```

قبول داریم که قاعدتاً باید به سری عدد به عنوان نمره بگیریم. جمعشون رو به جا نگه داریم و در آخر تقسیم بر این تعداد کنیم که میانگین به دست بیاد؟  
پس به متغیر به عنوان نگه دارنده جمع نمرات تعریف می کنیم:

```
total_score = 0
```

مقدار اولیش هم صفره چون هنوز هیچ نمره ای درونش نرفته و خب مجموع ابتدایی صفره دیگه!

خب مرحله بعد چیه؟ باید به تعداد student\_count ها به کاری رو انجام بدیم. این با چی ممکن بود؟ با while.

- خب چطور while رو بسازیم؟ یکم فکر کنیم!

+ می تونیم بگیم مثلاً هر بار که طی میشه، توی while، یکی از تعداد دانشجوها کمتر شه. پس شرط while رو چطور می گذاریم؟

خب تا وقتی که تعداد دانشجو صفر بشه. (چون هر بار یکی یکی از تعداد کم می کردیم) یعنی تا کی while ادامه پیدا کنه؟ تا وقتی که تعداد دانشجو بزرگتر از صفر باشه باید while انجام شه.

نکته! لزوماً نیاز نیست همون لحظه که while رو می نویسیم، شرط رو هم بنویسیم! بلکه یکم فکر کنیم به روند و روند رو طی کنیم ببینیم شرط چی می تونه باشه! مثلاً اینجا من گفتم چطور طی کنم؟ به تعداد دانشجو. پس هر بار می تونم یکی از تعداد دانشجو کم کنم و وقتی به صفر رسید یعنی تموم. پس شرط میشه:

```
while student_count > 0:
```

خب توی while باید به نمره از کاربر بگیرم. یعنی:

```
while student_count > 0:
```

```
    score = int(input('Enter score: '))
```

خب بعدش باید اضافه اش کنیم به مجموعها:

```
while student_count > 0:
```

```
    score = int(input('Enter score: '))
```

```
    total_score = total_score + score
```

خب بعدش باید یکی از تعداد کم کنیم:

```
while student_count > 0:
```

```
    score = int(input('Enter score: '))
```

```
    total_score = total_score + score
```

```
    student_count = student_count - 1
```

خب قسمت while ما تموم شد. قبول داریم در آخر باید مجموع رو تقسیم بر تعداد کنیم؟ پس:

```
student_count = int(input('Enter student count: '))
```

```
total_score = 0
```

```
while student_count > 0:
    score = int(input('Enter score: '))
    total_score = total_score + score
    student_count = student_count - 1
average_score = total_score / student_count
```

خب همین جا وایسین! به نظرتون این کد چه مشکلی داره؟ یکمی فکر کنین.  
 راهنمایی ۱: به خط آخر دقت کنین.  
 راهنمایی ۲: به student\_count دقت کنین!  
 راهنمایی ۳: کد رو اجرا کنین و ببینین که ارورش چیه:

```
ZeroDivisionError: division by zero
```

پاسخ: ببینین ما توی while اومدیم از student\_count هی یدونه یدونه کم کردیم که در آخر صفر شد! پس خط آخر تقسیم بر صفر داره. یه عدد تقسیم بر صفر توی ریاضی تعریف نشده هست و باگ این قسمت اینه! فکر کنین ببینین چطور می تونین این مشکل رو برطرف کنین؟!

پاسخ:

کافیه که یه کاپی از student\_count بگیریم و عملیات کم کردن رو روی اون کاپی انجام بدیم. مهم نیست که اون کاپی صفر شه یا نه! چون یه کاپی اضافه هست. درواقع یه متغیر کاپی می سازیم. یعنی مقدار student\_count رو می ریزیم توش:

```
student_count = int(input('Enter student count: '))
student_count_copy = student_count
total_score = 0
while student_count_copy > 0:
    score = int(input('Enter score: '))
    total_score = total_score + score
    student_count_copy = student_count_copy - 1
average_score = total_score / student_count
print(average_score)
```

حالا کد درست کار می کنه!

ببینین! دقیقاً تفاوت این آموزش با آموزش های دیگه اینه که سعی می کنم قشنگ فکری که توی ذهن رخ میدره رو پیش ببرم و مثل یه فردی که داره تازه یاد می گیره فکر و حل مسأله رو پیش ببرم و همراه با شما کد رو باگ یابی (دیباگ) کنیم و قدم به قدم با فکر و استدلال پیش بریم. نه اینکه یه دفعه بگم خب باید یه کاپی بگیریم. به جاش مثل فکر عادی انسان بدون کاپی گرفتن پیش میرم و نشون میدم که به باگ می خوریم. پس حالا باید به این دلیل یه کاپی بگیریم!

نکته! تقسیم بر صفر یکی از رایج‌ترین باگ‌هایی هست که بهش بر می‌خورین. هر وقت عملیات تقسیم داریم، فکر کنین که آیا شرایطی ممکنه پیش بیاد که تقسیم بر صفر رخ بده؟!

### روش دوم:

می‌تونیم یه شمارشگر (counter) بذاریم و هر بار که while اجرا شد، یکی بهش اضافه شه و در آخر خب مشخصاً تعداد نمرات رو بهمون نشون خواهد داد:

```
student_count = int(input('Enter student count: '))
counter = 0
total_score = 0

while student_count > 0:
    score = int(input('Enter score: '))
    counter = counter + 1
    total_score = total_score + score
    student_count = student_count - 1

average_score = total_score / counter
print(average_score)
```

### پاسخ ۳:

### روش اول:

خب اول نیازه که یه متغیر برای جمع نمرات داشته باشیم. اسمشو می‌ذارم score\_sum. همچنین چون تعداد رو بهم نداده، نیازه که خودم تعداد رو به دست بیارم (چون برای میانگین گرفتن، نیاز به تعداد داریم دیگه!). برای همین نیازه به یه متغیر دارم که هر بار که ورودی گرفتیم، یکی به تعداد اضافه کنه. پس تا اینجا داریم:

```
score_sum = 0
count = 0
```

خب حالا ما باید برنامه رو بندازیم توی یه while که همینطور ورودی بگیره. اما شرط پایان چی باید باشه؟ خب من اینجا برای شرط پایان از یه متغیر به نام is\_end استفاده می‌کنم. می‌گم مقدار اولیه متغیر ۰ هست. بعد می‌گم وقتی که عدد ۱- وارد شد، این مقدار رو بکن ۱ و شرط while رو روی همین می‌ذارم. یعنی می‌گم تا وقتی که این متغیرم صفر هست، بمون توی while:

```
score_sum = 0
count = 0
is_end = 0
while is_end == 0:
    inp_grade = int(input())
    if inp_grade == -1:
```

```
is_end = 1
```

خب تا اینجا کار، ما `while` مون رو ساختیم. یعنی گفتیم تا وقتی که هنوز متغیر ما صفر هست، یه سری کار انجام بده.

ورودی بگیر. چک کن بین ورودی اگر منفی ۱ بود، `is_end` رو بکن ۱ که عملاً `while` ما تموم شه. حالا اگر نمره ورودی ما -۱ نبود، خب باید به جمع اضافه کنیم و به تعداد نمرات دریافتی هم یکی اضافه کنیم.

در خارج `while` هم میانگین بگیریم. چرا خارج `while`؟ چون باید تموم ورودی‌ها رو گرفته باشیم و دیگه نمره‌ای نمونه باشه.

```
score_sum = 0
count = 0
is_end = 0
while is_end == 0:
    inp_grade = int(input())
    if inp_grade == -1:
        is_end = 1
    else:
        score_sum = score_sum + inp_grade
        count = count + 1
```

```
if count != 0:
    print(score_sum / count)
```

- به نظرتون چرا آخرش نوشتیم اگر `count` مخالف صفر بود، میانگین بگیره؟  
+ چون اگر نمی‌نوشتیم، اگر یه وقت کاربر همون اولین ورودی، -۱ میداد، تعداد نمرات صفر بود و توی پرینت، به ارور تقسیم بر صفر یا همون «ZeroDivisionError: division by zero» می‌خوردیم.

### روش دوم:

خب می‌گم بیا اولین ورودی رو در حالت عادی بگیر. بعد چک کن اگر ورودی منفی ۱ نبود، بیا اول جمع رو حساب کن، یکی هم به `count` اضافه کن. بعد ورودی‌های بعدی رو باز بگیر.

```
score_sum = 0
count = 0
inp_grade = int(input())
while inp_grade != -1:
    score_sum = score_sum + inp_grade
    count = count + 1
    inp_grade = int(input())
```

```
if count != 0:  
    print(score_sum / count)
```

پاسخ ۴:

```
average = total / count
```

یه متغیر مجموع باید بسازم که هر بار نمرات رو جمع کنم و بریزم توش

```
total_score = 0
```

باید یه متغیر هم بذارم که تعداد نمرات رو نگه‌داره و هر بار نمره‌ای بهش دادن، یکی بهش اضافه‌تر شه.

```
score_count = 0
```

تمیزنویسی: سعی کنین اسم‌تون بامعنا باشه. یعنی score\_count اسم بهتری از count هست. چون اطلاعات بیشتری میده و کسی که کد رو می‌خونه می‌فهمه که منظور ما چیه. همچنین زیاد هم طولانی نیست.

علامت مساوی از کرکترای قبل و بعدش یکی فاصله می‌گیره. یعنی:

```
correct: score_count = 0
```

```
incorrect: score_count=0
```

خب بعدش باید یه ورودی بگیریم ببینیم اگر ۱- بود، یه عدد بگیره.

```
user_input = int(input('Enter choice: '))
```

خب قبول دارین که هی باید ورودی بگیرم و هی باید چک کنم که آیا ۱- هست یا نه و اگر بود یه نمره بگیرم؟

درواقع هی تکراریه که به یه شرط وابستس. تکرار وابسته به شرط چی بود؟ while.

خب شرط while چی می‌تونه باشه؟ شرط باید این باشه که هر وقت که ورودی که گرفتم ۱- هست یا نه؟ پس:

```
total_score = 0
```

```
score_count = 0
```

```
user_input = int(input('Enter choice: '))
```

```
while user_input == -1:
```

خب گفتیم که تا وقتی که user\_input ما برابر ۱- هست، یه سری کارها کن. چه کارهایی؟ یه ورودی به عنوان نمره بگیر و بعد به total اضافه کن (توجه کنین که یه خط فاصله بین سه خط اولی و while گذاشتم که خواناتر شه براتون. خط اضافه سفید هیچ تأثیری توی برنامه نداره! پایتون خط‌های سفید رو ایگنور می‌کنه. صرفاً گذاشتم که بخشای برنامه براتون بهتر مشخص شن!):

```
total_score = 0
```

```
score_count = 0
```

```
user_input = int(input('Enter choice: '))
```

```
while user_input == -1:
```

```
student_score = int(input('Enter score: '))
```

```
total_score += student_score
```

```
score_count += 1
```

حواسمون هم هست که یکی به تعداد نمرات یا همون score\_count اضافه کنیم. یادتون نره! اگر یادتون بره، score\_count صفر میمونه و در آخر که بخوایم میانگین بگیریم، تعداد صفره و تقسیم بر صفر می خوریم!

- وایسا وایسا! این چیه نوشتی؟! `score_count += 1` دیگه یعنی چی؟ همینطوری سریع نرو جلو! توضیح بده!

+ باشه باشه! ببینین پایتون اومده گفته که من می خوام نوشتار شما رو کمتر کنم. نخواین خیلی زیاد بنویسین. راحت تر شه کارتتون. برای همین اگر می خواین یه چیزی رو به مثلاً score\_count اضافه کنین و بریزین توی خودش، اینطوری بنویسین:

```
score_count = score_count + 1 → score_count += 1
```

درواقع گفته به جای اینکه خودشو بنویسی، += بذار ساده تره. مثالی دیگه:

```
number = number + 1
```

```
number += 1
```

```
number = number - 7
```

```
number -= 7
```

```
number = number / 2
```

```
number /= 2
```

```
number = number * 6
```

```
number *= 6
```

البته دقت کنین که اولویت بندی باعث باگ برنامهتون نشه. مثلاً مورد زیر:

```
number *= 6 + 2
```

اینطوری تفسیر میشه:

```
number = number * 6 + 2
```

یعنی ضربدر ۶ میشه و بعد بعلاوه ۲ میشه. طبیعی هم هست! درواقع داریم می گیم خودش ضربدر ۲ + ۶ و خب اولویت ریاضی اینطور حکم می کنه که ضربدر ۶ شه و بعد بعلاوه ۲ شه. پس اگر خواستین جور دیگه باشه، می تونین پرانتز بگذارین:

```
number *= (6+2)
```

```
number = number * (6 + 2) = number * 8
```

زمانایی که تقسیم و ضرب و توان و اینا هست که اولویتشون با جمع و تفریق فرق داره، حواستون بهش باشه.



خب برگردیم به کد. باید قبل برگشتن به بالا و چک کردن شرط `while`، یه ورودی دیگه به عنوان انتخاب کاربر بگیریم که ببینیم بازم ۱- میده که بخواد عدد دیگه وارد کنه و همچنان توی `while` بمونیم یا نه؟

```
total_score = 0
score_count = 0
user_input = int(input('Enter choice: '))

while user_input == -1:
    student_score = int(input('Enter score: '))
    score_count += 1
    total_score += student_score
    user_input = int(input('Enter choice: '))
```

خب حالا کارمون این قسمت تموم شد. در آخر باید میانگین رو حساب کنیم و چاپش کنیم:

```
total_score = 0
score_count = 0
user_input = int(input('Enter choice: '))

while user_input == -1:
    student_score = int(input('Enter score: '))
    score_count += 1
    total_score += student_score
    user_input = int(input('Enter choice: '))

average_score = total_score / score_count
```

```
print(average_score)
```

خب همینجا وایسین! به نظرتون این برنامه چه مشکلی ممکنه به وجود بیاره؟ یادتونه توی سؤال قبلی یه باگ رایج رو توضیح دادم؟ خب حالا همون باگ ممکنه اینجا هم ظاهر شه. چه زمانی؟ فکر کنید! + زمانی که کاربر برای بار اول ۱- رو نمی‌زنه و عملاً `score_count` صفر باقی می‌مونه. حالا سعی کنید یه راهی پیدا کنید که این رخ نده! راهنمایی: سعی کنید تقسیم رو صرفاً زمانی انجام بدین که `score_count` بزرگ‌تر از صفره.

پاسخ:

```
total_score = 0
score_count = 0
user_input = int(input('Enter choice: '))
```

```

while user_input == -1:
    student_score = int(input('Enter score: '))
    score_count += 1
    total_score += student_score
    user_input = int(input('Enter choice: '))

if score_count > 0:
    average_score = total_score / score_count
    print(average_score)
else:
    print('No score entered')

```

درواقع چک می‌کنم اگر بزرگ‌تر از صفر بود، حساب می‌کنم و بهش `average_score` رو میدم. اگر نه (درغیر این صورت)، میام به چیز بامعنا چاپ میکنم که کاربر متوجه شه که هیچ نمره‌ای وارد نکرده و نمره‌ای هم در کار نیست!

فرق به برنامه‌نویس عادی با به برنامه‌نویس خوب اینه که حواسش هست نقاط حساس برنامه کجاست! حواسش هست که فکر کنه که اگر برنامه طبقی که من می‌خواستم پیش نره چی؟ وگرنه همه برنامه‌نویسا جووری برنامه می‌نویسن که اتفاقی که انتظار داریم رخ بده. برنامه‌نویسای خوب به این فکر می‌کنن که اگر یکوقت کاربر اشتباه کرد و یا روند برنامه جووری پیش رفت که انتظار نداشتیم، حالا باید چیکار کرد؟ چیکار کنیم برنامه باگ نخوره!

پاسخ ۵:

اول به عدد به عنوان  $n$  می‌گیریم:

```
n = int(input())
```

همیشه حواستون باشه وقتی می‌خوان مینیمم یا ماکزیمم به سری عدد رو به دست بیارین، همیشه اول عدد اولی که بهتون میدن رو بذارین هم توی `minimum` و هم توی `maximum`.  
- چرا؟ خب می‌تونم مثلاً مینیمم رو همیشه صفر قرار بدن از اول.  
+ خب اگر اول کد بگین مینیمم برابر صفر، این مشکل به وجود میاد که اگر اعداد اینا بودن:

```
10 20 16 9
```

مینیمم باید میشد ۹ ولی شما اشتباه گفتی صفر! برای همین همیشه سعی کنین که عدد اولی که میدن رو برابر مینیمم قرار بدین:

```

n = int(input())
inp_num = int(input())
maximum = inp_num
minimum = inp_num

```

خب حالا که عدد اول رو گرفتیم، باید به تعداد یکی کمتر از  $n$ ، عدد بگیریم (چون یکیشو قبلاً گرفتیم!):

```
n = int(input())
```

```
inp_num = int(input())
maximum = inp_num
minimum = inp_num
for i in range(n-1):
    inp_num = int(input())
```

خب حالا می‌گیم که عدد ما اگر از ماکزیمم بزرگ‌تر بود، یعنی ماکزیمم جدید ما اینه! پس بزارش توی ماکزیمم، اگر نه، اگر کوچکتر بود، مینیمم جدید ما اینه و پس بزارش توی مینیمم. اگر م که نه کوچکتر از مینیمم بود و نه بزرگتر، خب کاری نکن!

```
n = int(input())
inp_num = int(input())
maximum = inp_num
minimum = inp_num
for i in range(n-1):
    inp_num = int(input())
    if inp_num > maximum:
        maximum = inp_num
    elif inp_num < minimum:
        minimum = inp_num
```

خب حالا می‌ایم اختلاف مینیمم و ماکزیمم رو چاپ می‌کنیم:

```
n = int(input())
inp_num = int(input())
maximum = inp_num
minimum = inp_num
for i in range(n-1):
    inp_num = int(input())
    if inp_num > maximum:
        maximum = inp_num
    elif inp_num < minimum:
        minimum = inp_num
```

```
print(maximum - minimum)
```

اما یه مشکلی اینجا پیش میاد! برگردین به صورت سؤال ببینین مشکل کجا ممکنه پیش بیاد؟! تست کیس‌های لب مرزی و خاص و حالتای خاص.

مشکل ممکنه زمانی پیش بیاد که  $n$  ما صفر یا عددی کوچکتر باشه. برنامه ما منتظره `inp_num` رو بگیره ولی قرار نیست بهش داده شه! پس می‌ایم با شرط بهش می‌گیم که اگر  $n$  بزرگ‌تر یا مساوی ۱ بود، بیا حالا منتظر ورودی دومی هم بمون. خروجی هم صرفاً زمانی باید چاپ شه که  $n$  بزرگ‌تر یا مساوی ۱ باشه.

```
n = int(input())
```

```

if n >= 1:
    inp_num = int(input())
    maximum = inp_num
    minimum = inp_num

for i in range(n-1):
    inp_num = int(input())
    if inp_num > maximum:
        maximum = inp_num
    elif inp_num < minimum:
        minimum = inp_num

if n >= 1:
    print(maximum - minimum)

```

پاسخ %:

۱. من باید رقم رقم برم جلو و ببینم که آیا رقم فرد هست یا نه؟ عدد فرد چی بود؟ باقی‌موندش بر ۲ بشه.

خب یکم فکر کنین چجور باید این کار رو انجام بدیم.

عملیاتمون چی بود؟ باقی‌مونده گیری. پس من هر بار باید از رقم به رقم باقی‌مونده بگیرم. اما این خیلی سخته. نمیشه. پس چطوره اول از عدد باقی‌مونده بگیرم. (یعنی درواقع مشخص می‌کنه رقم سمت راست زوجه یا فرد)

بعد رقم راستی رو بریزم دور.

- چطور بریزیم دور؟

+ با یه بار shift دادن به سمت راست.

اینطوری رقم دوم میاد جای سمت راستی و عملاً سمت راست جدید ما رقم دومی هست. حالا باز باقی‌مانده می‌گیرم.

مثلاً:

123 → 12 → 1

خب درواقع بخوام یه رقم از عدد کم کنم یعنی باید چیکار کنم؟

باید تقسیم صحیح بگیرم بر ۱۰. یعنی هر بار عدد رو تقسیم صحیح کنم بر ۱۰.

پس کد اینطوری میشه:

```

input_number = int(input('Enter number: '))
odd_digit_count = 0

while input_number != 0:
    if input_number % 2 == 1:

```

```
odd_digit_count += 1
input_number //= 10
```

```
print(odd_digit_count)
```

یعنی توی `while` اول چک میشه که آیا باقی‌مونده بر ۲ برابر ۱ هست یا نه؟ اگر بود یکی به تعداد ارقام فرد اضافه می‌کنه. بعدش **خارج** `if` میاد عدد رو تقسیم صحیح می‌کنه بر ۱۰ و خارج `while` هم تعداد ارقام رو چاپ می‌کنه.

پاسخ ۷:

اول یه عدد به عنوان تعداد می‌گیرم:

```
number_count = int(input('Enter number count: '))
```

بعدش خب می‌تونم یه شمارنده بذارم که بشمره و تا وقتی که به `number_count` نرسیده، یه سری کارها رو پیش ببره:

```
counter = 0
```

بعدش خب من باید `minimum` رو پیدا کنم دیگه. خب می‌گم یه متغیر به نام `minimum` تعیین می‌کنم و بهش مقدار اولیه صفر میدم. `While` رو هم می‌گم تا وقتی پیش بره که `counter` کمتر از `number_count` هست. (این یعنی به تعداد `number_count` دارم `while` رو تکرار می‌کنم. اگر براتون واضح نیست، مثال بزنین تا بدونین `while` دقیقاً همون قدر بار تکرار میشه):

```
number_count = int(input('Enter number count: '))
counter = 0
minimum = 0
```

```
while counter < number_count:
```

خب توی `while` باید هر بار یه عدد بگیرم. یه دونه به `count` اضافه کنم و ببینم آیا عدد جدیدی که گرفتم از `minimum` بیشتر هست یا نه؟ اگر بود یعنی `minimum` از این به بعد باید نمایانگر عدد جدید باشه. یا درواقع مقدار عدد جدید رو توی خودش نگه داره:

```
number_count = int(input('Enter number count: '))
counter = 0
minimum = 0
```

```
while counter < number_count:
```

```
    number = int(input('Enter number: '))
```

```
    counter += 1
```

```
    if number < minimum:
```

```
        minimum = number
```

```
print(f'minimum is {minimum}')
```

و در آخر هم minimum رو چاپ کردم.  
اما میشد یه راه دیگه هم رفت:

```
number_count = int(input('Enter number count: '))
minimum = 0

while number_count > 0:
    number = int(input('Enter number: '))
    number_count -= 1
    if number < minimum:
        minimum = number

print(f'Minimum is {minimum}')
```

خب اما همینجا وایسین! به نظرتون هر دو راه بالا مشکلشون چیه؟  
حالتای مختلف ورودی رو دقت کنین ببینین آیا همیشه کد درست جواب میده؟  
ببینین مشکل اینجااست که من مقدار اولیه minimum رو دادم ۰. خب این یه مشکله! شاید کاربر  
عدد منفی وارد کنه و خب عدد منفی از صفر کوچکتره و کد من درست جواب نمیده. چون عدد منفی از  
عدد صفر کوچکتره!  
پس باید چیکار کرد؟  
همیشه توی مینیموم و مکزیموم، مقدار اولیه رو برابر عدد اول قرار بدین. یعنی من اولین ورودی رو  
بیرون while می‌گیرم و برابر minimum قرار میدم:

```
number_count = int(input('Enter number count: '))
counter = 0

number = int(input('Enter number: '))
minimum = number
counter += 1

while counter < number_count:
    number = int(input('Enter number: '))
    counter += 1
    if number < minimum:
        minimum = number

print(f'Minimum is {minimum}')
```

خب به نظرتون مشکل این کد کجا میتونه باشه؟

زمانی که کاربر تعداد اعداد رو صفر وارد می‌کنه. من نباید هیچ ورودی بگیرم. ولی بدون چک کردن شرطی، بیرون `while` یه ورودی گرفتم! حالا سعی کنین کد رو بهبود بخشین! یعنی بیرون `while` اول یه شرط چک کنین که کاربر عدد صفر یا حتی عدد منفی وارد نکرده باشه!

```
number_count = int(input('Enter number count: '))
counter = 0
```

```
if number_count > 0:
    number = int(input('Enter number: '))
    minimum = number
    counter += 1
```

```
while counter < number_count:
    number = int(input('Enter number: '))
    counter += 1
    if number < minimum:
        minimum = number
```

```
print(f'Minimum is {minimum}')
```

قسمت `if` با عمد نوشتن `number_count != 0` چون گفتیم شاید کاربر یه وقت `number_count` رو عددی منفی وارد کنه. اینطوری بازم وارد `if` میشد ولی اشتباه بود! برای همین گفتیم اگر بزرگ‌تر از صفر بود...

اما هنوزم این کد مشکل داره!

- ای بابا! ولمون کن بابا!

+ صفر رو بدین به عنوان تعداد و ببینین مشکلتش چیه؟

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

خب می‌گه `minimum` رو تعریف نکردی! وقتی تعریف نکردی چه‌جوری بهم می‌گی خط آخر چاپش کنم واست؟

یکم برگردیم کد رو بخونیم. عه آره راست می‌گه! وقتی صفر رو بدیم، اصلاً وارد `if` و `while` نمیشه و اصلاً `minimum` تعریف نمیشه و ساخته نمیشه! وقتی ساخته نمیشه، چه‌جور ازش می‌خوایم که پرینتش کنه؟!

پس باید خط آخر چک کنیم که اگر تعداد بزرگ‌تر از صفر بود، مینیموم داریم و مینیموم چاپ کنه. اگر نه، به کاربر بگین عددی وارد نکردی!

```
number_count = int(input('Enter number count: '))
counter = 0
```

```

if number_count > 0:
    number = int(input('Enter number: '))
    minimum = number
    counter += 1

while counter < number_count:
    number = int(input('Enter number: '))
    counter += 1
    if number < minimum:
        minimum = number

if number_count > 0:
    print(f'Minimum is {minimum}')
else:
    print('No numbers entered')

```

وایسین این کد یه مشکل دیگه هم داره!  
 - ای بابا بیخیال! دیگه چه مشکلی؟! دیگه بخوای هم من حل نمی‌کنم.  
 + هیچی شوخی کردم! مشکلی نداره!<sup>۴۷</sup> خواستم یکم اذیتتون کنم [ل(]] حالا کد درست کار می‌کنه و  
 ما خوشحالیم :)  
 - می‌زنمتا o\_O

برنامه‌نویسی سراسر تمرین، بررسی و رفع مشکل و دیباگ‌کردنه. پس خسته نشین! برنامه‌نویسی  
 همینه! هی می‌گین خب اینجاش ممکنه مشکل پیش بیاد. پس فلان جور طراحیش می‌کنم. عه اونجاش  
 مشکل پیش اومد پس درستش می‌کنم و کلی چیز دیگه!  
 برای همینم هست که برنامه‌ها هی آپدیت میدن و شما بعد آپدیت کردن میگی این برنامه که تغییری  
 نکرده! چرا همش داره آپدیت می‌ده؟!  
 موضوع اینه که تغییراتش مثل اینجاست! لزوماً اضافه کردن شکل و چیز میز قشنگ نیست! بلکه  
 اروریابی و رفع مشکل هم می‌تونه آپدیت باشه. چیزایی که شما حسش نمی‌کنین! پس برنامه‌ها تون رو  
 آپدیت کنین که برنامه‌نویس‌ها سخت مشغول حل مشکلن (:

### پاسخ ۸:

اینجا من یه حرکت جالب می‌خوام بزنم. می‌خوام با تقسیم و باقی‌مونده‌گیری این کار رو انجام بدم!  
 خب چطور؟ اول اینکه میام میگم خب اول باید حداکثر ۳ رقم اول سمت راست رو ببرم توی پاسخ. این  
 مرحله برای تمام ورودی‌ها یکسانه. (چه ورودی یک رقم باشه، چه دو رقم، چه ده رقم)

<sup>۴۷</sup> درواقع داره‌ها (: ولی شما فعلاً بلدش نیستین!



خب این کار رو با چی انجام بدم؟ با باقی‌مونده‌گیری بر ۱۰۰۰ (یکم با باقی‌مونده‌گیری بازی کنین، دستتون میاد)! حالا که انجامش دادم، میام این حداکثر سه رقم رو می‌ریزم دور. این کار رو با تقسیم صحیح بر ۱۰۰۰ انجام میدم:

```
inp_num = int(input())
readable = ''
readable = str(inp_num % 1000) + readable
inp_num //= 1000
```

بعدش می‌گم خب برای اونایی که طولشون بیشتر از ۳ بود، باید هی سه‌تا سه‌تا، «\_» بذارم براشون. به طوری که ترتیب زیر رعایت شه:

```
new_digits + '_' + readable
```

از یه `while` کمک می‌گیرم. می‌گم که تا وقتی که عدد بزرگ‌تر از صفره، بیا هی اضافه کن و حداکثر سه رقمش رو بریز دور. (با تقسیم صحیح)

```
inp_num = int(input())
readable = ''
readable = str(inp_num % 1000) + readable
inp_num //= 1000
while inp_num > 0:
    readable = str(inp_num % 1000) + '_' + readable
    inp_num //= 1000

print(readable)
```

کوییز:

پاسخ: دو عدد بهتون میدن و شما باید تلاش کنین مقسوم‌علیه‌های مشترک اون دو رو از بزرگ به کوچک پرینت کنین.  
مثال:

input:

20

12

output:

4

2

1

-----

input:

20

20

output:

20

10

5

4

2

1

پاسخ ا:

خب به راحتی می‌تونیم از عدد کوچکتر بیایم پایین و دونه دونه تست کنیم هرکدوم بخش پذیر بود پرینتش می‌کنیم.

```
num1 = int(input())
```

```
num2 = int(input())
```

```
if num1 < num2:
```

```
    divisor = num1
```

```
else:
```

```
    divisor = num2
```

```
while divisor > 0:
```

```
    if num1 % divisor == 0 and num2 % divisor == 0:
```

```
        print(divisor)
```

```
    divisor -= 1
```

یه متغیر به نام divisor قرار میدیم که عدد کوچکتر رو می‌ریزیم توش. بعدش یکی یکی میایم پایین و هر جا بخش پذیر بود، تموم می‌کنم.

خب تا بعد که بریم سراغ یه مبحث خیلی طولانی، یکم استراحت کنیم به نظرم (:

## 7. for

خب یادتونه از counter استفاده می‌کردیم تا بگیم while تا کجا پیش بره؟ مثلاً:

```
counter = 0
```

```
while counter < total:
```

خب اینجا به چیز دیگه هم داریم که کارمون رو ساده تر کنه. نخواستیم اول به متغیر تعریف کنیم، بعد توی while اضافه یا کمش کنیم. به نوع دیگه از حلقه داریم به نام «for».

```
for i in range(0, 3, 1):  
    print("Hi")
```

خب بیایم ببینیم اصلاً چی هست!

میگه برای متغیر i که از ۰ شروع میشه و تا ۳ (دقت کنین که به ۳ نمیرسه! تا قبل ۳)، پیش میره. چه جوری پیش میره؟ آخرین عدد میگه چجوری. میگه یکی یکی بهش اضافه میشه. دو نقطه بیا کارهای زیر رو انجام بده. کارهایی که باید انجام بده توی indentation اش هستن. (یعنی همون چهارتا فاصله جلوتر از for).

خب اول i برابر ۰ هست. شرط رو چک می کنه میگه کوچکتر از ۳ هستم پس میاد توی بلاکش و Hi رو چاپ می کنه. بعد یکی به i اضافه میشه. i ما میشه ۱. دوباره شرط رو چک می کنه میگه هنوز کوچکتر از ۳ هستم، میره پایین و Hi رو چاپ می کنه. بعدش باز میاد بالا. یکی به i اضافه میشه و برابر ۲ میشه. میره پایین Hi رو حساب می کنه. بعدش میاد بالا. یکی به i اضافه میشه و برابر ۳ میشه. شرط رو چک می کنه میگه عه! رسیدم به ۳. خب قرار بود تا قبل ۳ پیش برم. پس دیگه نمیره تو بلاک و از for می پره بیرون. درواقع ۳ بار اجرا شد.

یکم بازی کنین با پارامترها. مثلاً بیایم مقدار افزایش رو عوض کنیم. یعنی دوتا دوتا زیاد شه:

```
for i in range(0, 3, 2):  
    print("Hi")
```

اول متغیر for ما یعنی همون i برابر ۰ هست. کوچکتر از ۳ پس میاد Hi رو چاپ می کنه. بعد متغیر for ما میشه ۲. هنوزم کوچکتر از ۳ پس میاد و Hi رو چاپ می کنه. بعدش i میشه ۴. حالا کوچکتر از ۳ نیست! پس از for می پره بیرون. دو بار Hi چاپ میشه.

تمیز نویسی: کما به عدد قبلی می چسبه و از عدد بعدی به فاصله داره. یعنی:

```
correct: for i in range(0, 3, 2):
```

```
incorrect: for i in range(0,3,2):
```

- چه جوری توی این چیزا تعداد بار اجرا رو پیدا کنیم؟  
فرمول تعداد که توی دبستان خونديم:

$$\text{count} = [(last - first) / distance] + 1$$

آخری منهای اولی تقسیم بر فاصله بینشون (مثلاً اگر دو تا دوتا دارن زیاد/کم میشن، تقسیم بر دو)، در آخر هم بعلاوه ۱.  
مثلاً برای

```
for i in range(3, 9, 2)
```

از ۳ شروع میشه و تا قبل ۹ پیش میره. دو تا دوتا پیش میره. یعنی آخرین چیز، ۷ هست. ۳ ۵ ۷. تعداد هم به دست میشه آورد:

$$3 = [(7-3)/2] + 1$$

فرمول جمع اعداد هم براتون بنویسم شاید به کارتون اومد:

```
summation = [(last+first)*count]/2
```

(آخری بعلاوه اولی) ضربدر تعداد. در آخر هم تقسیم بر ۲

خب بازم یکم باهاش بازی کنیم:

```
for i in range(3, 0, -1):  
    print(i)
```

برای  $i$  هایی که از ۳ شروع میشن و تا قبل ۰ پیش میرن. روند پیش رفتنشون چطوره؟ بعلاوه ۱- میشن. یعنی منهای ۱ میشن درواقع. و توی بلاک for اومدم  $i$  رو چاپ کردم که روند رو متوجه شین که هر بار ۱ چه مقداری داره. اجراش کنین که بهتر درکش کنین.

خب کمی ساده‌تر!

چون ما خیلی وقتا for هامون شبیه for اولی یعنی `for i in range(0, 3, 1)` هست، پایتون گفته بیا من کارو واسه می‌کنم. نیاز نیست بنویسی یکی یکی برو بالا. ننویسش. اگر ننویسیش، من به صورت پیشفرض فکر می‌کنم که منظورت یکی یکی برم جلو هست. یعنی دو تای زیر عیناً یکسان:

```
for i in range(0, 3, 1):  
    print(i)
```

```
print('-----')
```

```
for i in range(0, 3):  
    print(i)
```

بینش یه دونه پرینت چندتا خط فاصله گذاشتم که بهتر بتونین از هم تفکیکشون کنین. خیلی وقتا برای تفکیک کدتون و دیدن نتایج اجرا، خوبه یه پرینت بگذارین که خروجی‌های مختلف از هم جدا باشن و بهتر بفهمین هرچیزی که `print` شده مال کجا بوده. من خیلی از این استفاده می‌کنم. اما خب یادتون نباید بره که بعداً پاکش کنین. وگرنه یه دفعه وسط برنامه‌تون چندتا خط فاصله میبینین پرینت میشه!

خب بازم پایتون گفته وقتی می‌نویسی `for i in range(0, 3)` یعنی درواقع می‌خواه ۳ بار یه چیزی انجام شه. و  $i$  تو از ۰ شروع شده. این نوع for خیلی رایجه و توی روز ده‌ها بار ممکنه نوشته شه. خب یکم باز من کارو براتون ساده‌تر می‌کنم که سریع‌تر بخواین for بنویسین. بالاخره وقت طلاست دیگه! دو ثانیه زودتر هم دو‌ثانیس!

وقتی `i` شما از ۰ شروع میشه و یکی یکی زیاد میشه، اینطوری بنویسینش (هایلات صورتی):

```
for i in range(0, 3):
```

```
    print(i)
```

```
print('-----')
```

```
for i in range(3):
```

```
    print(i)
```

هر دو مورد (بالایی که حالت اول بود و پایینی که حالت جدید)، عیناً یکی هستن و هر دو ۳ بار اجرا میشن.

حالا به من بگین یه `for` که بخوایم صدبار اجرا شه چجوریه؟

```
for i in range(100)
```

متغیر `for` هر چیزی می تونه باشه. لزوماً نیاز نیست `i` باشه. صرفاً کلمه `i` خیلی رایجه و می گذارنش. مثلاً:

```
for kourosh in range(5):
```

```
    print(kourosh)
```

هیچ فرقی با

```
for i in range(5):
```

```
    print(i)
```

نداره.

توجه! این متغیر در بیرون از `for` هم قابل استفاده هست. یعنی:

```
for i in range(4):
```

```
    print(i)
```

```
print(f'i outside of for is: {i}')
```

پس حواستون باشه که قاطی نشه با متغیرهای دیگه. یکی از اشتباهات رایج اینه که قبلش یه `i` داشتن و اینجا هم `i` تعریف می کنین و فکر می کنین با قبلی فرق داره ولی فرق نداره و مقدار قبلی عملاً از بین میره و مقدار جدید `for` جاش قرار می گیره:

```
i = 70
```

```
for i in range(4):
```

```
    print(i)
```

```
print(f'i outside of for is: {i}')
```

عملاً اون ۷۰ از بین میره و جاش رو `i` درون `for` میگیره.

حالا for رو یاد گرفتین! یکم باهاش بازی کنین و چیز میز مختلف باهاش انجام بدین. پارامترهای مختلفشو تست کنین که کامل مسلط شین.

کمی بیشتر درباره print کردن (حتمأً ببینین):

<https://youtube.com/shorts/uHbrCFwU2iY>

## 8. Boolean

بریم سراغ یه نوع از متغیر به نام boolean.

این نوع متغیر فقط دو نوع می‌تونه داشته باشه. «درست» یا «True» و «نادرست» یا «False». بذارین یه سؤال بپرسم. حاصل متغیرای زیر چی میشن؟

```
bool_var1 = 5 > 3
bool_var2 = 5 < 3
bool_var3 = 5 == 3
bool_var4 = 5 != 3
```

- عه سخت شد نمی‌دونم!

+ بذارین یه راهنمایی کنم. یادتونه می‌گفتیم سمت راست تساوی حساب میشه و ریخته میشه توی سمت چپ؟ پس برای اولی سمت راست رو حساب کنین. نگاه می‌کنه آیا ۵ بزرگ‌تر از ۳ هست؟ بله هست. پس حاصل میشه True. یعنی:

```
bool_var1 = True
```

برای بقیه هم چاپشون کنین و نتیجشو ببینین:

```
print(bool_var1)
print(bool_var2)
print(bool_var3)
print(bool_var4)
```

خب این به چه کاری میاد؟

مواقعی که ما اصطلاحاً می‌خوایم یه متغیر داشته باشیم که اگر توی روند برنامه یه تغییری رخ داد، بگیم بله تغییر رخ داد. یعنی فرض کنین می‌خوایم بفهمیم یه عدد زوج هست یا نه؟ می‌تونیم کدش رو اینطور بنویسیم:

```
num = int(input("Enter a number: "))

is_even = False
if num % 2 == 0:
    is_even = True
```

```

if is_even == True:
    print("Even")
else:
    print("Odd")

```

یه عدد گرفتیم. به صورت پیش فرض یه متغیر به نام `is_even` تعریف می کنیم. (توی چیزای `boolean` همیشه سعی کنین اسامی بامعنا باشن و نشون دهنده `boolean` بودنشون باشن. مثلاً اینجا مشخصه `is_even` یعنی «آیا زوج است؟» «بله» یا «خیر».

بعدش میام چک می کنم که اگر بر ۲ بخش پذیر بود، یعنی اون پرچم یا `flag` ام که `is_even` بود رو تغییر میدم. میگم عه یافتم که زوجه. پس پرچم رو میکنم «بله» یا «درست». بعدش حالا در انتهای برنامه چک می کنم که اگر پرچم یا `flag` ام `True` بود، چاپ کنه `Even` (زوج) وگرنه، چاپ کنه `Odd` (فرد).

یه نکته! پایتون بازم خواسته کار ما رو ساده تر کنه. یعنی گفته که دو تا `if` زیر عیناً یکی هستن:

اولی اینطوری ترجمه میشه: «اگر `is_even` برابر `True` بود»

دومی اینطوری ترجمه میشه «اگر `is_even` برقرار بود»

دومی هم همون معنا میده. برقرار بودن یعنی درست بودن. یعنی `True` بودن.

پس برای سادگی کار، ما اینطوری می نویسیم:

```

if is_even == True:
    print("Even")
if is_even:
    print("Even")

```

یا سه مورد زیر عیناً یکی هستن:

```

if is_even != True:
    print("Odd")

```

اگر `is_even` مخالف `True` بود.

```

if is_even == False:
    print("Odd")

```

اگر `is_even` برابر `False` بود. (یعنی مخالف `True`)

```

if not is_even:
    print("Odd")

```

اگر مخالف `is_even` برقرار بود.

قراره `if` زمانی اجرا شه که مقدار جلوش `True` شه درسته؟

خب وقتی میگیم `if not is_even`، یعنی زمانی که مخالف `is_even` برقرار باشه. یعنی `is_even`

باید `False` باشه که مخالفش بشه `True` که وارد `if` شه.

یکم پیچیده شد. چند بار بخونیدش و تحلیلش کنین و تا وقتی کامل مسلط نشدین، ادامه رو نخونین!

به چه دردی می‌خوره؟  
مثلاً روند برنامه من طولانی هست. مثلاً ده تا مرحله هست که اگر حتی یه مرحله هم درست بود، پرچم درست بشه و اگر پرچم درست شد، در نهایت یه چیزی رو چاپ کنم. وگرنه یه چیز دیگه‌ای رو چاپ کنم.

یعنی مثلاً ده تا if و اگر هرکدومشون درست شد، توشون flag رو True کنم.

خب حالا به من بگین حاصل is\_even چی میشه؟

```
num = 5  
is_even = num % 2 == 0
```

بینین همیشه اول سمت راست تساوی حساب میشه و ریخته میشه توی سمت چپ تساوی. پس اول

```
num % 2 == 0
```

حساب میشه. خب حاصل این چی میشه؟

```
5 % 2 == 0
```

یا در واقع:

```
1 == 0
```

خیر مساوی نیست! گفتیم که این علامتای مقایسه‌ای مثل == و != < > و... میان boolean بر می‌گردونن. پس حاصل اینجا میشه False. چون ۱ مساوی ۰ هست؟ خیر نیست! پس False. پس توی is\_even، مقدار False ریخته میشه!

خب یه سوال! به نظرتون کد زیر ارور می‌خوره؟

```
num = 0  
if (2 > 3 and 5 / num):  
    print("Hello")
```

- بله به نظرم ارور می‌خوره! چون تقسیم بر صفره! ما یادمونه که بهمون گفتی حواسمون باشه هر وقت تقسیم دیدیم، حواسمون رو جمع کنیم که تقسیم بر صفر به کار نبریم چون تعریف نشدس!  
+ خوبه یادتون مونده. اما این کد به ارور بر نمی‌خوره!  
- وات؟! چرا؟  
+ چون پایتون از بالا به پایین و از چپ به راست پیش میره. یعنی می‌گه خب اگر ۲ بزرگ‌تر از ۳ بود و همچنین...

نگاه می‌کنه میبینه عه! ۲ که بزرگ‌تر از ۳ نیست! پس می‌گه خب صد درصد این عبارت جلوی if به خاطر شرط اولیش False هست. پس الکی چرا قدرت CPU رو درگیر کنم و الکی بشینم قسمت دوم رو هم حساب کنم؟ خب می‌دونم and یعنی هر دو باید برقرار باشن. چون اولی برقرار نیست، خب چه نیازی به چک کردن دومی هست؟ پس دومی رو چک نمی‌کنه که اصلاً بدونه تقسیم بر صفر هست یا نه!



- من کد رو اجرا کردم چیزی چاپ نکرد. چرا؟  
+ خب چون چیزی پرینت نمیشه در کل برنامه! برای همین چیزی چاپ نشده!

یه نمونه دیگه:

```
num = 0
if (2 < 3 or 5 / num):
    print("Hello")
```

or یعنی حداقل یکیش برقرار باشه اوکیه!  
چک می‌کنه می‌گه عه! ۲ کوچکتر از ۳ هست. خب پس True. پس نیازه به چک کردن دومی نیست  
اصلاً. چون or حتی یکیش اوکی باشه، اوکیه. پس الکی قدرت CPU رو درگیر نمی‌کنم و شرط دومی  
چک نمی‌کنم و میرم توی بلاک if.

حواستون به این چیزا باشه که ممکنه کارتتون اشتباه باشه ولی چک نشه و بعداً به واسطه همین به ارور  
بر بخورین. چون می‌گین اگر num صفر بود که توی if ارور میداد. پس num برابر صفر نیست!

بریم سراغ یه سؤال خیلی طولانی!  
- وای! سخت شد!  
+ نه نه! اصلاً نترسین! حل سؤال آسونه. فقط می‌خوایم هی بهینه و بهینه ترش کنیم! یعنی  
چندین روش بریم و هی بگیم اینجاش میشه بهتر کرد و هی برنامه رو سریع تر کنیم! برای همین طولانیه!

## • یافتن عدد اول

برنامه‌ای بنویسین که چک کنه آیا یه عدد اول هست یا نه؟  
راهنمایی ۱: عدد اول چه عددی بود؟ عددی که صرفاً بر ۱ و خودش بخش پذیر باشه.  
راهنمایی ۲: یعنی درواقع اگر از ۲ تا قبل عدد پیش بریم، نباید بر هیچ عددی بخش پذیر باشه!  
راهنمایی ۳: اگر حتی بر یک عدد بخش پذیر بود، باید بگین خب اول نیست.  
راهنمایی ۴: این با چی بود؟ با flag. یعنی اگر حتی یه عدد هم برش بخش پذیر بود، پرچم is\_prime  
به صورت False در بیاد.  
راهنمایی ۵: بسه دیگه! یکمم خودتون فکر کنین!

روش اول:

```
num = int(input("Enter a number: "))
is_prime = True

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

```

if num % i == 0:
    is_prime = False

if is_prime:
    print("Prime")
else:
    print("Not prime")

```

اول یه عدد میگیرم به عنوان عددی که باید چک کنم اوله یا نه؟  
 بعدش فکر میکنم که باید از ۲ تا قبل عدد برم و چک کنم آیا برشون بخش پذیر هست یا نه؟ اگر بود،  
 خب باید یه جا ذخیره کنم که این عدد اول نیست! پس قبل for میام یه متغیر به نام is\_prime تعریف  
 می‌کنم. مقدار اولیش True هست. چرا؟ چون توی for می‌گم که حتی اگر بر یه عدد بخش پذیر بود، اول  
 نیست و باید False شه.

بعدش در نهایت چک می‌کنیم که اگر بعد تموم شدن کل عملیات‌های بالا، هنوز flag ما True مونده،  
 یعنی عدد اول هست و باید چاپ شه که بله اوله!

## روش دوم:

خب بریم یکم کد رو بهینه‌تر کنیم!  
 گفتیم اگر حتی بر یه عدد بخش پذیر باشه، یعنی اول نیست دیگه! مثلاً عدد ۱۲ رو در نظر بگیرین. هم  
 بر ۲، هم ۳، هم ۴، هم ۶ بخش پذیره.  
 ما فقط بدونیم بر ۲ بخش پذیره کافیه! دیگه چه نیازی بریم تا تهش؟ الکی داریم کار انجام میدیم.  
 اینجا پایتون اومده یه چیزی گذاشته به نام break. یعنی «بسه»! «کافیه»! دیگه نمی‌خواد بری جلوتر!  
 یعنی قسمت for کد بالا رو اینطوری می‌نویسیم:

```

for i in range(2, num):
    if num % i == 0:
        is_prime = False
        break

```

یعنی حتی یه دونه هم پیدا شد، flag رو بکن False و break کن. بسه دیگه!  
 درواقع کار break اینه که از داخلی‌ترین بلاک for یا while می‌پره بیرون. (توجه کنین که کاری به  
 if نداره! بلکه نگاه می‌کنه به نزدیک‌ترین for یا while و از اون می‌پره بیرون).  
 مثلاً فرض کنین یه کد اینطوری داشتیم:

```

for i in range(5):
    for j in range(6):
        if j == 3:
            break

```

صرفاً از for داخلی (یعنی for ای که متغیرش ز هست می پره بیرون) و هنوز توی for بیرونی هست. به اینا میگن nested for. حالا بعداً بیشتر باهاش آشنا میشیم.

خب تا اینجا به طرز خیلی خوبی برنامه رو بهینه کردیم! چون الکی مقادیر اضافه رو چک نکردیم! یکی که پیدا شد گفتیم پپر بیرون! بسه!

### روش سوم:

خب اگر عدد ما  $n$  باشه، ما باید  $n-2$  حالت رو چک کنیم. این اصلاً بهینه نیست. بیایم یکم بهترش کنیم. اگر عددی بر ۲ بخش پذیر نباشه، دیگه نیاز نیست بریم تا خود عدد. حداکثر نیازه تا نصفش برم. چون اگر بر ۲ بخش پذیر بود، حداکثر بزرگترین مقسوم علیه دیگه تا نصفش بود: یعنی بزرگترین مقسوم علیه یه عدد حداکثر میتونه نصفش باشه دیگه! چون اگر نصفش باشه، از ضرب نصفش و ۲ تشکیل شده. یعنی دیگه اعداد بزرگتر از نصف اون عدد رو نیاز نیست چک کنم! صرفاً نیازه تا همون نصف برم!

```
num = int(input("Enter a number: "))
is_prime = True
```

```
for i in range(2, num//2):
    if num % i == 0:
        is_prime = False
        break
```

حواستون باشه که تقسیم صحیح کردم. چون اعشاری میشد و گرنه و نمیشد توی for به کارش برد. چون for عدد با عدد صحیح کار می کنه نه اعشاری!

### روش چهارم:

اما بازم این بهینه نیست. ما تقریباً  $n / 2$  بار باید طی کنیم. بهتره بریم تا  $\sqrt{n}$ . حداکثر مقسوم علیه اول یه عدد، جذرش هست دیگه.

- چرا؟

+ بر هر عدد دیگه ای بخش پذیر باشه، مثلاً بزرگتر از جذر، حتماً یه مقسوم علیه اول کوچکتر از جذر داره. مثلاً ۵۵، درسته بر ۱۱ هم بخش پذیره، اما یه مقسوم علیه کوچکتر از جذر هم داره که ضربدر ۱۱ بشه و برابر ۵۵ بشه. اونم ۵ هست. پس حداکثر اون عدد ما یه عدد مربع کامل هست پس تا جذر پیش بریم. چون دیگه هرچقدر عدد بدقلق باشه، بزرگترین مقسوم علیه جذرشه. اگر نبود، حتماً یه مقسوم علیه کوچکتر از جذر داره!

قبلش باید با نحوه جذرگرفتن در پایتون آشنا شیم:

ببینین فرض کنین شما یه سری کد نوشتین. مثلاً کد اینکه آیا یه عدد اول است یا خیر. این کد رو دیگه نمی‌خوان هر بار بنویسین! چی میشد که یه اسم براش انتخاب کنیم و هر بار که خواستیم ازش استفاده کنیم، فقط بگیم که فلان اسم.

اینجا یه سری چیز میز ساخته شدن به نام `function` یا تابع. درواقع مثل تابع ریاضی که مثلاً می‌گیم:

$$y = 2x + 5 \rightarrow y(3) = 11$$

عین همینم توی پایتون هست. یعنی هم خودتون می‌تونین بسازیدش و هم افراد دیگه می‌تونن بسازن و در اختیار شما قرار بدن.

یه سری افراد نشستن کد نوشتن برای محاسبه جذر یه عدد، سینوس یه عدد، کسینوس یه عدد و کلی چیز دیگه. گفتن آقا ما اینا رو نوشتیم! شما نیاز نیست زحمت بکشی! هر بار خواستی استفاده کنی، فقط اسمشو صدا بزن! ما اینا رو توی یه جا که بهش می‌گیم کتابخونه یا `library` گذاشتیم. اسم اون کتابخونه رو گذاشتیم کتابخانه ریاضی یا «`math`». هر وقت خواستی ازش استفاده کنی، خط اول کدت بنویس:

```
import math
```

یعنی `math` رو وارد کدم کن.

این خیلی خوبه. مثلاً بخوایم جذر یه عدد چاپ شه می‌تونیم بنویسیم:

```
import math
print(math.sqrt(25))
```

یعنی اول گفتیم `math` رو وارد کدم کن.

بعدش گفتیم چاپ کن از کتابخونه `math`، جذر رو بیار و ۲۵ رو به عنوان پارامتر بهش دادم. `sqrt` مخفف `square root` یا همون ریشه دوم عدد هست. می‌تونین یکم تمرین کنین چیزای مختلف انجام بدین. مثلاً:

```
import math
print(math.sin(90))
```

عه چرا سینوس ۹۰ رو ۱ نشون نداد؟

چون تابع `sin` پایتون مبناش رادیانه.

پس حالا می‌تونیم کد رو بنویسیم:

```
import math

num = int(input("Enter a number: "))
is_prime = True

for i in range(2, int(math.sqrt(num)) + 1):
    if num % i == 0:
        is_prime = False
        break

if is_prime:
```

```

print("Prime")
else:
    print("Not prime")

```

به نظرتون چرا نوشتیم `int(math.sqrt(num))`؟ همون `cast` کردن و تبدیل کردنه. یعنی تبدیلیش کردم به عدد صحیح یا `int`. چون گفتیم `for` با عدد صحیح کار می‌کنه! چرا بعلاوه یک‌ش کردم؟ چون گفتیم `for` میره تا قبل خود عدد! یعنی اگر ۲۵ بدیم، تا قبل ۵ میره. یعنی تا ۴! بعلاوه یک کردم که بشه ۶ و تا ۵ که جذر هست پیش بره!

اما یه نکته! جذر گرفتن، کاری زمان‌بر هست. برای کامپیوتر جذر گرفتن یه کار سخته. پس چیکار کنیم؟ به جای اینکه بگیم `i` کوچکتر جذر باشه، بگیم ضرب `i` در `i` کوچکتر و یا مساوی عدد باشه. عملاً برعکسش رفتیم. ولی چون عملیات ضرب برای کامپیوتر ساده‌تر هست، سریع‌تره. این رو با `while` پیاده‌سازی می‌کنم:

```

num = int(input("Enter a number: "))
is_prime = True

i = 2
while i * i <= num:
    if num % i == 0:
        is_prime = False
        break
    i += 1

if is_prime:
    print("Prime")
else:
    print("Not prime")

```

یه نکته باحال! امنیت `https` بر این استواره که ضرب دو عدد برای کامپیوتر سادس ولی فاکتورگیری و یافتن ریشه‌های اول یه عدد برای کامپیوتر سخته. رمزنگاری `https` وب بر این استواره.<sup>۴۸</sup> همین مفاهیم ساده، پیش‌زمینه مهم‌ترین چیزای روزمره زندگیمون!

## روش پنجم؛

خب بیایم یکم باز بهترش کنیم! به نظرتون چطور میتونیم بهترش کنیم؟

---

۴۸ درواقع رمزنگاری `RSA` بر این استواره. می‌تونین آموزشش رو از قسمت «`RSA`» بخونین!

+ ببینین اگر عدد زوج نباشه، ما الکی داریم ۲، ۴، ۶، ۸ و... رو هی چک میکنیم. درحالی که اگر زوج نباشن، یعنی بر ۲ بخش پذیر نباشن، عملاً چک کردن ۴، ۶، ۸ و... بی فایده! چون بر این ها هم بخش پذیر نیستن. یعنی عملاً داریم یکی درمیان اعداد رو الکی چک می کنیم! پس بیایم بهینه ترش کنیم:

```
num = int(input("Enter a number: "))
is_prime = True

if num % 2 == 0:
    is_prime = False
else:
    i = 3
    while i * i <= num:
        if num % i == 0:
            is_prime = False
            break
        i += 2

if is_prime:
    print("Prime")
else:
    print("Not prime")
```

اول چک میکنم آیا بر ۲ بخش پذیره یا نه؟ اگر بود که اول نیست. اگر نبود، دیگه از ۳ شروع میکنم و دیگه به جای اینکه یکی یکی بریم که اعداد زوج رو بخوایم دوباره چک کنیم، میگیرم دوتا دوتا میریم. یعنی ۳، ۵، ۷، ۹ و... یعنی صرفاً اعداد فرد رو چک می کنیم.

خب اما به نظرتون مشکل تمام این کدها چیه؟  
راهنمایی: فکر کنید روی کدون عدد جواب نمیده؟  
درسته! اعداد ۱ و ۲ رو درست جواب نمیدن! مثلاً ۲ زوج و توی if اولی اشتباهی می گیم که چون زوج اول نیست! ولی خب استثناء هست و حواسمون به استثناءها باشه!  
پس می تونیم قسمت if که چک می کنه که اگر زوج is\_prime رو زوج می کنه، بگیریم که اگر زوج و ۲ نیست. یعنی:

```
if divisor % 2 == 0 and divisor != 2:
```

سعی کنیم تغییرات سرعت برنامه رو با بنچمارک چک کنیم.<sup>۴۹</sup> چون گاهی فکر می‌کنیم که تغییراتی که دادیم کد سریع‌تر شده ولی فکرتون اشتباهه. باید تست کنیم تا مطمئن بشیم.

یه راه ساده برای تست مدت زمان اجرای یه برنامه، استفاده از ابزارهای تحت ترمیناله. مثلاً توی لینوکس ابزار «hyperfine» می‌تونه کمک کنه.

سعی کنیم ورودی خیلی بزرگ باشه که تأثیر رو بهتر متوجه بشیم. مثلاً روی ورودی خیلی کوچیک، ممکنه تفاوت سرعتی مشخص نباشه و حتی یکی که کندتره، اشتباهی سریع‌تر نشون بده. پس برای همین، سعی کنیم ورودی رو تا حد امکان بزرگ بدیم.

```
hyperfine 'python3 file.py'
```

که `file.py` اسم فایل هست که می‌خواهیم اجرا شه. (حواستون باشه که کدتون نباید ورودی بگیره. باید مقادیر رو توی خود کد ست کنیم و ورودی نگیریم! مثلاً اولین کدی که اجرا کردم، زمانش شد:

```
Time (mean ± σ): 5.115 s ± 0.125 s [User: 5.111 s, System: 0.005 s]
Range (min ... max): 4.907 s ... 5.330 s
```

و این کد، زمانش شد:

```
Time (mean ± σ): 7.6 ms ± 2.0 ms [User: 6.3 ms, System: 1.3 ms]
Range (min ... max): 6.4 ms ... 16.4 ms
```

در اصل بنچمارک اینطوری عمل می‌کنه که پردازنده در یه دمای خاص، بدون ران شدن چیز اضافه و با شرایط یکسان تست می‌کنه که شرایط خارجی مثل اجرا شدن برنامه‌ای دیگه روی سرعت اجرای این برنامه تأثیر نذاره. معمولاً سایتا بهتر می‌تونن عمل کنن چون مکانیزمای بهتری دارن نسبت به شما.

درواقع فرض کنیم شما یه کدی دارید که صرفاً با یه `for` اجرا میشه و این `for` شما حداکثر  $n$  بار انجام میشه. (مثل روش اول یافتن عدد اول) یه کد دیگه داریم که یه `for` داره که حداکثر  $n / 2$  بار انجام میشه. (مثل روش سوم یافتن عدد اول)

- به نظرتون کدوم کندتره؟

+ قاعده‌تاً اولی! چون داریم  $n$  بار انجام میدیم ولی توی دومی،  $n / 2$  بار. دومی کمتر کار انجام میدیم و پس سریع‌تره.

توی روش چهارم درواقع ما داریم حداکثر  $\sqrt{n}$  بار یه کاری رو انجام میدیم. که از  $n / 2$  بار خیلی کمتره.

<sup>۴۹</sup> یه وبسایت خوب برای نحوه محاسبه زمان:

How to Measure Execution Time of a Program: <https://serhack.me/articles/measure-execution-time-program/>

درواقع تفاوت توی عددهای بسیار بزرگ معلوم میشه. مثلاً:

```
int(sqrt(4773338041828177)) = 69089348
```

```
4773338041828177 // 2 = 2386669020914088
```

دیدین؟ توی حالت رادیکالی نیاز به تعداد حالات خیلی خیلی کمتری از تقسیم بر ۲ طی کنیم.

## • Time complexity

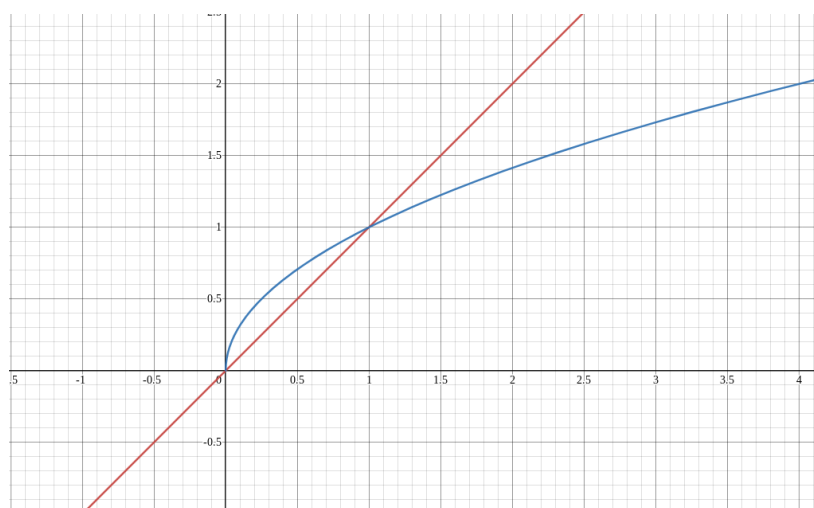
برنامه‌نویسا برای اینکه بتونن کدشونو از لحاظ سرعتی مقایسه کنن، از یه مفهوم ریاضی‌طوری به نام «time complexity» یا «پیچیدگی زمانی» استفاده می‌کنن.

اگر پیچیدگی زمانی یه الگوریتم  $n$  باشه؛ یعنی تابع زمان اجراش  $y(n)=n$  هست.

اگر پیچیدگی زمانی یه الگوریتم  $\sqrt{n}$  باشه؛ یعنی تابع زمان اجراش  $y(n)=\sqrt{n}$  هست.

**سوال:** کدوم بهتره؟ رشد تابع به صورت  $n$  باشه بهتره یا  $\sqrt{n}$ ؟ (زمان  $n$  برابر شه یا زمان  $\sqrt{n}$  برابر

شه؟)



تابعی که شکلش خط صافه و قرمز رنگه، تابع  $y = n$  و تابعی که انحنا داره و آبی رنگه، ( $y = \sqrt{n}$ ) هست.

**پاسخ:** همونطور که می‌بینین، از یه نقطه به بعد، همیشه تابع  $\sqrt{n}$  زیر تابع  $n$  قرار داره. پس با افزایش ورودی، زمان خیلی رشد پیدا نکرده و در زمان کمتر و سریع‌تری اجرا شده. یعنی هرچی تابع ما کوچکتر باشه، برنامه‌مون مطمئناً از یه سری ورودی به بعد سریع‌تر انجام میشه.<sup>۵۰</sup>

درواقع time complexity مفهوم خیلی مهمیه. درواقع شما اگر الگوریتمی بنویسین که  $n^3$  باشه و من الگوریتمی بنویسم که  $n$  باشه، با بزرگ‌شدن ورودی، الگوریتم شما به شدت بد عمل می‌کنه. چون با  $n$  برابر شدن ورودی، زمان  $n^3$  برابر میشه ولی مال من صرفاً  $n$  برابر الگوریتم به شدت کند میشه. مثلاً:

<sup>۵۰</sup> چرا گفتم از یه سری ورودی به بعد؟ شکل  $n$  و  $\sqrt{n}$  رو یاد خودتون بیارین. تا یه سری جاها ممکنه  $n$  زیر  $\sqrt{n}$  بوده باشه، اما از یه جایی به بعد همیشه  $\sqrt{n}$  زیر  $n$  هست.



	Time(n)	Time(n <sup>3</sup> )
Input = 1	1	1
Input = 100	100	1000000

مقایسه کنیم که به ازای ورودی ۱۰۰، مال شما ۱ میلیون کار انجام می‌ده ولی مال من صرفاً ۱۰۰ تا کار. مال شما ۱۰ هزار برابر کندتر از منه.  
ورودی رو یکم بزرگ‌تر کنیم:

$n^3 \rightarrow \text{input} = 1000 \rightarrow \text{time} = 1,000,000,000$

$n \rightarrow \text{input} = 1000 \rightarrow \text{time} = 1000$

نسبت به قبل که ۱۰۰ بود، ورودی ده برابر بزرگ‌تر شد. پس زمان من ده برابر میشه و تبدیل به ۱۰۰۰ میشه. اما توی  $n^3$ ، زمان ۱۰ به توان ۳ برابر میشه!  
حالا نسبت به مال من ۱ میلیون برابر کندتره! یعنی هرچه ورودی بزرگ‌تر، الگوریتم شما هی بدتر و بدتر میشه!

خب کد رو خیلی خوب بهینه کردیم و درباره پیچیدگی زمانی چیز می‌ز یاد گرفتیم! خسته نباشین!  
پر قدرت! آفرین به شمایی که همراه بودی و تلاش برای بهینه کردن کد انجام دادی! ایول بهت ؛)  
برو برای جایزه یه سیب بخور :)  
- عه معمولاً میگن شکلات بخور!!  
+ سیب سالم‌تره :)

## کوویز

هم به وسیله **while** و هم به وسیله **for** حل کنید!  
۱- کامپیوتر بلد نیست اعداد رندوم تولید کنه. درواقع اصلاً بلد نیست چیزی رو شانسی انجام بده! کامپیوتر یه رباته که هرچی بهش بگی انجام میده. باید بهش دستور بدی. خارج دستور هیچی نمی‌فهمه. شانسی کاری نمی‌کنه. دستور باید واضح و مشخص باشه. یعنی مثلاً بگی ۲ رو با ۴ جمع کن. ولی نمی‌تونی بگی یه عدد رندوم بده. بلد نیست! برای همین یکی از مهم‌ترین چالش‌ها اینه که چطور اعداد رندوم تولید کنیم؟ اصلاً ممکنه؟

اینجاست که اگر بخوایم صرفاً به صورت نرم‌افزاری عدد رندوم تولید کنیم، ریاضی‌دان‌ها دور هم جمع میشن و سعی می‌کنن با استفاده از فرمول، یه سری اعداد شبه‌رندوم تولید کنن. یکی از اون الگوریتم‌ها «**Blum Blum Shub**» نام داره که توسط «**Lenore Blum, Manuel Blum and Michael Shub**» ساخته شده. (**Blum Blum** ها برنده جایزه نوبل کامپیوتر (تورینگ) هستن!)

این الگوریتم اینطوریه که یه  $p$  و یه  $q$  و یه  $x_0$  می‌گیره و از اون به بعد،  $x$  های بعدی رو اینطوری می‌سازه:

$$x_i = (x_0^{(2^i \bmod \text{lcm}(p-1, q-1))}) \bmod M$$

خب هرکدوم از اینا چی هستن؟

مورد ۱: مرحله  $i$  ام

مورد  $M$ : ضرب  $p$  در  $q$ . یعنی  $M = q * p$

مورد  $\text{lcm}(p-1, q-1)$ : این یعنی ک.م.م (کوچکترین مضرب مشترک)  $(p-1)$  و  $(q-1)$ . که به انگلیسی

میشه  $\text{least common multiple (lcm)}$ .

### ورودی

به شما به ترتیب  $p$  و  $q$  و  $x_0$  و  $n$  داده میشه:

$p$   
 $q$   
 $x_0$   
 $n$

### خروجی

شما باید  $n$  عدد بعدی  $x_0$  را چاپ کنید:

$x_1$   
 $x_2$   
 $x_3$   
 $\dots$   
 $x_n$

مثال:

input:<sup>51</sup>

11  
 23  
 3  
 6

output:

9  
 81  
 236  
 36

۵۱) همونطور که می بینین،  $p = 11$  و  $q = 23$  و  $x_0 = 3$  و  $n = 6$  است.

31

202

-----

input:

7

11

5

4

output:

25

9

4

16

-----

پاسخنامه:

پاسخ ۱:

```

# lcm of two numbers:
p = int(input())
q = int(input())
m = p * q
p = p - 1
q = q - 1

if p < q:
    min_mul = q
else:
    min_mul = p
is_found = False
while (is_found != True):
    if min_mul % p == 0 and min_mul % q == 0:
        is_found = True
    else:
        min_mul += 1

# implementation of blum blum shub to print n random numbers:
i = 1
x0 = int(input())
n = int(input())
while i <= n:
    xi = (x0 ** (2**i % min_mul)) % m
    print(xi)
    i += 1

```

حالا این رو حل کنیم:

برنامه‌ای بنویسین که مقسوم‌علیه‌های اول یه عدد رو چاپ کنه.

**راهنمایی:**

خب چی شد؟ سخت بود؟ اول باید مقسوم‌علیه‌های یه عدد رو پیدا کنیم. بعد دوباره چک کنیم اون مقسوم‌علیه‌های یافته شده اول هستن یا نه؟ خود اول بودن یا نبودن خودش چندان `if` و `while` داشت! پس چیکار کنیم؟ سخت شد نه؟ کدا زیادی میرن تو هم! اما وایسین! این سؤال رو با عمد اوردم! چی میشد کدهای تشخیص اینکه یه عدد اوله یا نه رو یه جا بذاریم و وسط کدمون بگیریم خب برو با اون کد چک کن ببین این عدد اوله یا نه؟ یعنی برای این چند خط کدمون اسم بذاریم و هر وقت خواستیم ازشون استفاده کنیم، فقط اسمشونو بیاریم. به این میگن «تابع» یا همون «function».

## 9. Function

درواقع شما به تیکه کدتو میبری به جا. اسمشو به چیزی می گذاری. بعداً هر وقت بهش نیاز داشتی، صدش می زنی. درواقع برای کدت به تعریف (definition) ارائه میدی. مثلاً میگی تعریف می کنم که این به تابع هست. این تابع به مقداری رو می گیری و به مقداری رو پس می ده. مثلاً به عدد می گیری و بهت به boolean (یا هر مقداری که دوست داری) پس میدی که می گه اوله یا نه. بیایم عیناً همین تعریف کردن و اینا رو به زبون پایتون بنویسیم:

به مثال ساده می زنم که مثال زوج بودن یا فرد بودن عدد:

```
def is_even(num):  
    if num % 2 == 0:  
        return True  
    else:  
        return False
```

```
if is_even(5):  
    print("Even")  
else:  
    print("Odd")
```

میام به گوشه از کدهام (فرقی نداره کجا! هرکجا باشه اوکیه! فقط همونطور که گفتیم که پایتون از بالا به پایین کد رو می خونه، باید اول تابع باشه و بعد از اون ازش استفاده بشه! همیشه اول استفاده کنین بعد بگذارینش!)، define می کنم (def) که به تابع دارم می سازم به نام is\_prime. بعدش توی پرانتز می گم که این تابع قراره به متغیر بگیره. این متغیر توی تابع اسمش num خواهد بود!

میگم این عددی که گرفتی رو چک کن. بین اگر بر ۲ بخش پذیر بود، به من برگردون (return کن) مقدار True رو. ولی اگر نبود، بهم برگردون False رو.

درواقع هر جا return انجام شد، تابع تموم میشه و از تابع میاد بیرون.

در بیرون از تابع من می نویسم که اگر اون کد بودا که اسمشو گذاشتم is\_even، اگر بهش ۵ رو بدم، مقداری که برگردوند True بود، چاپ کن Even. یعنی درواقع می گم اگر مقدار برگشتی (return) تابع برای عدد ۵، True بود، بنویس Even.

اگر نه (مقدار False بود)، چاپ کن Odd.

### تمرین!

حالا به تابع is\_odd بنویسین که تشخیص بده که به عدد فرده یا نه. اگر بود، True برگردونه و اگر نبود False.

### پاسخ

```
def is_odd(num):  
    if num % 2 == 1:
```

```

return True
else:
    return False

```

```

if is_odd(5):
    print("Odd")
else:
    print("Even")

```

راستی این `is_even` رو اینطورم میشد نوشت:

```

def is_even(num):
    if num % 2 == 0:
        return True
    return False

```

چرا؟ چون نیاز به `else` نبود! اگر `if` برقرار بود که میره توی بلاکش و `True` رو `return` می‌کنه و از تابع میاد بیرون! اگر هم `if` برقرار نبود، خب نمیره تو `if` و میره خط بعدیش و خط بعدیشم نوشته `return False` یعنی `False` رو برگردون. هردو یه کار انجام میدن.

حتی میشد اینطوری هم نوشتش:

```

def is_even(num):
    return num % 2 == 0

```

یعنی برگردون `boolean` حاصل اینکه عدد بر ۲ شده ۰ یا نه. اگر شده خب `True` بر می‌گردونه. اگر صفر نشده مثلاً `۰ == ۱` میشه و خب حاصل این `False` هست و `False` بر می‌گردونه.

دقت کنین که هر متغیری میشه به تابع داد. یعنی من می‌تونم خارج تابع، به تابع `number` رو بدم. یا عدد ۵ مثل بالا رو بدم. فرقی نداره! صرفاً توی تابع اسمش `num` میشه. همین!

خب بریم اون تابع اینکه یه عدد اوله یا نه رو بسازیم:

```

def is_prime(num):

```

میام `define` می‌کنم (`def`) که یه تابع دارم می‌سازم به نام `is_prime`. بعدش توی پرانتز می‌گم که این تابع قراره یه متغیر بگیره. این متغیر توی تابع اسمش `num` خواهد بود! خب چیزایی که قراره انجام بده رو می‌نویسیم و تابع رو تکمیل می‌کنیم:

```

def is_prime(num):
    if num % 2 == 0 and num > 2:
        return False

```

```

else:
    i = 3
    while i * i <= num:
        if num % i == 0:
            return False
        i += 2
    return True

```

اگر بر یکی هم بخش پذیر بود، return می‌کنم False رو. اگر هم کل مراحل بالا طی شد و خب تابع به وسیله return تموم نشده، یعنی اوله و پس در آخر return می‌کنم True رو.

این تابع is\_prime. حالا هر وقت خواستم ببینم یه عدد اوله یا نه، صرفاً صداش می‌زنم! از روی مقدار boolean ای که بر می‌گردونه می‌فهمم اوله یا نه!

*حالا جواب تمرینی که قبل دونهستن تابع مطرحش کردیم!*

اول باید از ۲ تا خود عدد پیش بریم.

- چرا تا خود عدد؟

+ چون خود عدد هم ممکن اول باشه و هر عددی مقسوم علیه خودش هست! ما قرار بود مقسوم علیه‌های اول یه عدد رو چاپ کنیم! اگر عدد اول باشه، تنها مقسوم علیه اولش، خودش! و بعدش مقسوم علیه‌هاشو پیدا کنیم و پاسشون بدین به تابع که اگر اون مقسوم علیه اول بود، چاپش کنیم:

```

def is_prime(num):
    if num % 2 == 0 and num > 2:
        return False
    else:
        i = 3
        while i * i <= num:
            if num % i == 0:
                return False
            i += 2
        return True

```

```

num = int(input("Enter a number: "))
for i in range(2, num + 1):
    if num % i == 0:
        if is_prime(i):
            print(i)

```

از ۲ تا خود عدد رفتیم. (یادمه که for تا یکی قبل از پایانی میرفت. یعنی  $num + 1$  به ما میگه که تا خودش میره. (یکی کمتر از  $num + 1$  میشه  $!num$ ))

بعدش دونه دونه  $i$  ها رو چک می‌کنم که اگر عدد برشون بخش‌پذیر بود، یه چک کنه ببینه اول هستن یا نه؟ اگر بودن چاپشون کنه. اگر هم نبودن که هیچی! وارد بلاکش نمیشه و میره دوباره بالا و  $i$  یکی بهش اضافه میشه و تا جایی که برسه به عدد که آخرین باره که میاد داخل بلاک for.

اینجا باید با نحوه دیباگ کردن (debugging) شین ولی خب توضیحش توی متن سخته. پس برید یه فیلم درباره نحوه دیباگ کردن یاد بگیرید. (یا منتظر بمونین من خودم ویدیو ضبط کنم:))

خب حالا با تابع آشنا شدین؟ دیدین چقدر کار رو ساده می‌کنه؟ اول تعریفش می‌کنیم و بعد تعریف ازش استفاده می‌کنیم. کدآمون خیلی قشنگ میشه!

از همین الان سعی کنین چیزایی که میشه تابعش کرد رو تابع کنین. نخواین همیشه هی کد `is_prime` رو ده جای کد بنویسین. صرفاً تابعش می‌کنین که بیاد انجامش بده. این خیلی به تمیزشدن کدتون کمک می‌کنه.

شما تابع رو می‌نویسین و تستش می‌کنین. دیگه مطمئنین که اون تابع داره حداقل ۹۹ درصد درست کار می‌کنه. دیگه توی کدتون هر جا به مشکل خوردین، صرفاً روند خارجی رو چک می‌کنین. چون میدونین تابعتون درسته.

ولی اگر تابع نمی‌نوشتین، حالا بیا توی ۱۰ هزار خط کد، پیدا کن مشکل کجاست! نمیشه! درواقع تابع چند تا مزیت داره:

- کد رو زیباتر، خواناتر، ساده‌تر<sup>۵۲</sup> و قابل دیباگ‌تر می‌کنه.
- یه بار تستش کن، همیشه استفاده کن!
- کدت سریع‌تر میشه!<sup>۵۳</sup>

### تمرین!

۱- برنامه‌ای رو با کمک تابع بنویسین که یه عدد از کاربر بگیرد. اگر عدد اول بود چاپ کنه `prime`.

اگر نه، چک کنه ببینه اگر زوجه و بزرگ‌تر از ۲۰ هست، چاپ کنه `Even and bigger than 20`

اگر نه چاپ کنه `Odd and not prime`

<sup>۵۲</sup> شما نیاز نیست ۵۰۰ خط پشت کم کد بزنی. می‌تونی یه کد ساده بزنی که از فانکشن‌های متفاوتی استفاده کنه. که قشنگ مشخص باشه هر قسمت چیکار می‌کنه. (روی این کلی تمرین می‌کنیم)

<sup>۵۳</sup> بله کدتون سریع‌تر میشه! زمانی که رسیدین به `dictionary`. بیاین متن زیر رو بخونین:

متغیرهای `global`، درون یه دیکشنری ذخیره میشن که برای یافتن متغیر مورد نیاز، یه سری محاسبات ریاضی انجام میشه. (هش و...) اما متغیر درون تابع، درون یه لیست (یا بهتر بگیم آرایه) ذخیره میشه. برای همین یافتنش ساده‌تر و سریع‌تره. (درواقع تفاوت سرعتی از استفاده از متغیر `global` و `local` هست). بیشتر:

<https://stackabuse.com/why-does-python-code-run-faster-in-a-function/>



۲- برنامه‌ای رو با کمک تابع بنویسین که یه `username` و یه `password` از کاربر بگیرد. اگر `username` برابر `admin` و پسورد برابر ۱۲۳۴ بود، بنویسد `Welcome` و در غیر این صورت، تا زمانی که کاربر درست وارد نکرده است، هی از اون `username` و `password` بگیرد. راهنمایی: تابع اگر بخواد دو تا چیز ورودی بگیره، می‌تونین توی پرانتز با کاما اون دوتا چیز رو جدا کنین.

۳- قضیه نامساوی مثلثی میگه که هر ضلع مثلث، از مجموع دو ضلع دیگر کوچکتره.<sup>۵۴</sup> برنامه اینه که سه عدد بهتون میدن و شما باید بگین می‌تونه مثلث باشه یا نه؟

پاسفنامه:

پاسخ ا:

```
def is_prime(num):
    if num % 2 == 0 and num > 2:
        return False
    else:
        i = 3
        while i * i <= num:
            if num % i == 0:
                return False
            i += 2
        return True

def is_even(num):
    return num % 2 == 0

num = int(input("Enter a number: "))
if is_prime(num):
    print("Prime")
elif is_even(num) and num > 20:
    print("Even and bigger than 20")
else:
    print("Odd and not prime")
```

خیلی تمیزتر نیست؟  
تابع خیلی کار رو قشنگ می‌کنه.

---

<sup>۵۴</sup> البته قسمت تفاضل این نامساوی رو فعلاً نادیده می‌گیریم

```
def is_login_valid(username, password):
    if username == "admin" and password == "1234":
        return True
    return False

username = input("Enter username: ")
password = input("Enter Password: ")

while is_login_valid(username, password) == False:
    print("Invalid username or password")
    username = input("Enter username: ")
    password = input("Enter Password: ")

print("Welcome")
```

خب به `username` و به `password` می‌گیرم. بعد می‌گم تا وقتی که تابع چک‌کردن درستی مقادیر `login` (اسمش گذاشتم `is_login_valid`)، بهم مقدار `False` رو برگردوند (یعنی خروجیش `False ==` بود)، بنویس که یکیشو اشتباه وارد کردی و دوباره `username` و `password` رو بگیر. اگر هم درست بود، از `while` خارج میشه و چاپ می‌کنه `Welcome`. تابع رو هم اینطوری تعریف کردم که دو تا چیز می‌گیره. وقتی قراره دوتا چیز بگیره، از کاما استفاده می‌کنیم. می‌گیم `username` و `password` رو ورودی می‌گیره. و خب اگر شرط برقرار بود، `True` رو `return` می‌کنه. و خب اگر برقرار نبود، نمیره توی بلاک `if` و `return` می‌کنه `False` رو. (نیازی به `else` نبود. چون در هر صورت باید `False` رو `return` کنه)

- مزایای این چیه به نظرتون؟

+ خب اینکه من اگر بخوام به تغییری توی برنامه به وجود بیارم و مثلاً بگم که از این به بعد اگر `password` برابر فلان چیز بود، اجازه ورود بده، راحت می‌دونم که صرفاً نیاز به برم توی تابع تغییرش بدم. جاشو می‌دونم. اما اگر تابع نمی‌نوشتم، ده ساعت باید کل کد رو می‌گشتم تا بدونم همه جا رو درست تغییر دادم؟ این زمانی به چشم میاد که کدهاتون ده‌ها هزار خط کد شه! الان کارتون سادس و بدون تابع می‌نویسین اما ذهن شما مثل یه خمیره که باید درست شکل بگیره. اگر درست شکل بگیره، در نوشتن تابع ماهر میشین و خیلی تمیز کاراتونو پیش می‌برین.

نکته! متغیرهای درون تابع، با متغیر بیرون تابع فرق دارن! درون تابع مال داخلشه. بیرون مال بیرونه. یعنی username داخلی با بیرونی فرق دارن و یکی نیست!

تذکر! همچنین پسوردهایی خیلی سادن و هیچوقت نباید ازشون استفاده کنین. رمز عبورتون بالای ۲۰ رقم باشه و شامل موارد زیر میتونه باشه:

A-Z / a-z / 0-9 / `~!@#\$%^&\*()-\_+=+{}[\];./?.,<>

یعنی هم حروف کوچک، هم بزرگ، هم اعداد، هم سمبل‌ها. نمونه یک رمز قوی:

aW?eN3#os^BR2@jxfTupA8%vcM\$k

- چه خبره بابا! یادمون میره!

+ از پسورد منیجر استفاده کنین. (پسورد منیجر چیه؟؟ سرچ کنین دربارهش! پیشنهاد: Bitwarden)

پاسخ ۳:

روش اول:

میگم اگر هر سه شرط برقرار باشه، یعنی بله می‌تونه مثلث باشه. کدوم سه شرط؟ اینکه هر ضلع از مجموع دو ضلع دیگه کوچکتر باشه.

```
def is_triangle(a, b, c):  
    if (a + b > c) and (a + c > b) and (b + c > a):  
        return True  
    return False
```

روش دوم:

به صورت پیشفرض فرض می‌کنم که می‌تونه تشکیل بده. یعنی یه متغیر در نظر می‌گیرم که مقدارش True هست. حالا هر وقت شرطی از اون سه شرط برقرار نبود، میگم خب باشه پس برقرار نیست و مثلث نیست و پس متغیر رو False می‌کنم.

در نهایت مقدار متغیر رو return می‌کنم.

```
def is_triangle(a, b, c):  
    flag = True  
    if a + b <= c:  
        flag = False  
    elif a + c <= b:  
        flag = False  
    elif b + c <= a:  
        flag = False  
    return flag
```

حالا دیدین چقدر به کار بردن and و or و اینا برای جلوگیری از تکرار تعداد if خوبه؟

تابع لزوماً نیاز نیست boolean برگردونه! می‌تونه هر چیزی برگردونه. چه عدد اعشاری چه متن. هرچی!

حتی return یه تابع می‌تونه چندتا چیز برگردونه ولی موقعی که صداش می‌زنیم، باید به همون تعداد متغیر بهش پاس بدیم:

```
def f():  
    num1 = 1  
    num2 = 2  
    num3 = 3  
    return num1, num2, num3  
  
num1, num2, num3 = f()  
print(num1, num2, num3)  
  
print(f'The result of f is: {f()}')
```

حتی نیاز نیست لزوماً چیزی رو بگیره! مثل اینجا. تابع چیزی نگرفت. لزوماً که نیاز نیست حتماً یه چیزی بگیره!

حواستون باشه که موقع صدازدن تابع، متغیر اشتباه به تابع پاس ندین. یا تعداد اشتباهی متغیر پاس ندین! مثلاً تابع یه ورودی می‌گیره ولی موقع صدازدنش دوتا بهش دادیم. ارور می‌خوره:

```
def return_num(num):  
    return num  
  
print(return_num(5, 6))
```

همونطور که می‌بینیم ارور داده:

```
TypeError: return_num() takes 1 positional argument but 2 were given
```

می‌تونیم به ورودی‌های تابع (بهش می‌گن argument) یه مقدار پیشفرض بدیم که اگر موقع صدازدن تابع چیزی بهش ندادیم، به صورت پیشفرض، مقدار رو اون در نظر می‌گیره.

```
def return_num(num=5):  
    return num  
  
print(return_num())
```

اینجا چیزی به تابع پاس ندادیم. پس خودش می‌گه به صورت پیشفرض مقدار `num` رو ۵ در نظر می‌گیرم.

اگر چندتا `argument` داشته باشیم، اگر به یکی مقدار پیشفرض بدیم، باید از اون به بعد هم مقدار پیشفرض بدیم.

```
def return_num(num1, num2=7, num3=10):  
    return num1 + num2 + num3
```

```
print(return_num(1))
```

چون `num2` مقدار پیشفرض دادم، باید هر `argument` بعد از اون هم مقدار پیشفرض بدم.

- آیا همیشه ساخت تابع باعث سریع‌تر شدن کد میشه؟  
+ خیر! مثلاً:  
به نظرتون کدوم کد سریع‌تره:

Code 1:

```
def f():  
    for i in range(1_000_000_00):  
        1 + 1
```

`f()`

Code 2:

```
def f():  
    1 + 1  
  
for i in range(1_000_000_00):  
    f()
```

کد یک سریع‌تره. چون که خود صدا زدن تابع هزینه‌برداره. پس برای هر چیزی تابع ننویسین! یا اگر `for` ای که خیلی بار اجرا میشه (مثل مورد بالا) رو روی یه تابع پیاده‌سازی می‌خوانین کنین، برای سریع‌تر شدنش، `for` رو ببرین توی تابع که تعداد `function call` هامون کمتر شه.

برنامه‌نویسای حرفه‌ای دو کار خوب انجام میدن:

- ۱- کدشون رو تا حد ممکن خوانا می‌کنن که تا یه نگاه کلی به کد بندازیم، بفهمیم چیکار کرده.
  - ۲- از ابزارهایی استفاده می‌کنن که کد رو بررسی و آنالیز کنه و مشکلاتو پیدا کنه.
- یکی از کارهایی که میشه کرد که دو چیز بالا رو در بر می‌گیره، «type hint» هست. به این صورت که وقتی تابعی تعریف می‌کنیم، صرفاً برای راهنما، می‌گیم که پارامترهایی که می‌گیره، از چه تایپی هستن و چه تایپی رو بر می‌گردونه؟
- همونطور که از اسمش معلومه، صرفاً راهنماست! هیچ تأثیری در اجرای برنامه نداره. صرفاً باعث خوانایی بیشتر و اینکه ابزارها بتونن کد رو برای یافتن مشکل آنالیزکنن به کار میره.
- مثال ۱:

به جای اینکه بنویسیم:

```
def stringConc(s1, s2):  
    return s1 + s2
```

می‌تونیم بنویسیم:

```
def stringConc(s1: str, s2: str) -> str:  
    return s1 + s2
```

یعنی داریم راهنمایی می‌کنیم که پارامتر اول، `string` هست. پارامتر دوم هم همینطور. همچنین با فلش نشون می‌دیم که تایپی که برمی‌گردونه، چیه. تذکر! این صرفاً یه راهنماس. وگرنه بازم می‌تونیم هر تایپی بخوایم به تابع پاس بدیم. این کلمات صرفاً راهنمایی هستن.

مثال ۲:

way 1:

```
def is_even(n):  
    return n % 2 == 0
```

better way:

```
def is_even(n: int) -> bool:  
    return n % 2 == 0
```

یعنی ورودی `n` به `int` هست. خروجی هم عبارتی `boolean` هست.

مثال ۳:

```
from typing import Any
```

```
def add(a: Any, b: Any) -> Any:  
    return a + b
```

داریم راهنمایی می‌کنیم که هر تایپی می‌تونه بگیره.

سعی کنید از این روش استفاده کنید. کدتون حرفه‌ای‌تر و بهتر میشه.

## 10) String

تا اینجا خیلی با اعداد بازی می‌کردیم. اما مهم‌تر از اعداد، متن و رشته‌ها (string) هستن. خب یه متغیر string رو چطور تعریف می‌کردیم؟ اینطوری:

```
string1 = "Hello World"
```

توی کامپیوتر هریک از این حروف یه شماره دارن. شماره جایگاه (index) از ۰ شروع میشه تا آخر استرینگ.

یعنی شماره جایگاه‌ها به این ترتبه:

H	e	l	l	o		W	o	r	l	d
0	1	2	3	4	5	6	7	8	9	10

ایندکس ۵، کرکتر فاصله هست.

و اگر بخوایم به کرکتره دسترسی پیدا کنیم، می‌گیم:

```
string1[1]
```

مثلاً بخوایم کرکتر دوم (ایندکس اول) رو بریزیم توی یه متغیر. اینطوری می‌گیم:

```
string1 = "Hello World"
```

```
second_char = string1[1]
```

```
print(second_char)
```

همونطور که دیدین ریختیم توی متغیر second\_char و چاپش کردیم.

حالا فرض کنید که می‌خواهیم کرکتر اول (ایندکس صفر) تا قبل از کرکتر چهارم (ایندکس ۳) رو پرینت کنیم:

```
string2 = "abcdefgh"
```

```
print(string2[0:3])
```

به این کار می‌گن slicing.

مثل for اگر اولی رو نگذاریم، یعنی از ۰ شروع کن:

```
string2 = "abcdefgh"
```

```
print(string2[:4])
```

از ایندکس ۰ تا قبل ۴.

یا حتی اینطوری:

```
string = "abcdefgh"
```

```
print(string[2:])
```

از ایندکس ۲ شروع کن برو تا تهش!

اینجا هم مثل for بود که سه پارامتر می گرفت، اینجا هم می تونیم پارامتر سوم رو می تونیم increment اش بدیم. یعنی بگیم مثلاً ۲ تا ۲ تا ایندکس برو جلو:

```
string2 = "abcdefgh"
print(string2[0:5:2])
```

قبل اجرا کردنش، یکم روش فکر کنین که ببینین چه کرکترایی چاپ میشه؟  
+ خب میگیریم از ایندکس ۰ شروع کن برو تا قبل ۵. دو تا دوتا برو جلو. اون کرکترایی که توی رنج جا میگیرن رو چاپ کن.

دفعه قبل که عدد سوم رو ندادیم، خودش به صورت پیشفرض فکر می کرد که یکی یکی باید بره جلو. مثل for برعکس هم می تونیم کنیم. یعنی:

```
string2 = "abcdefghi"
reverse = string2[7:4:-1]
print(reverse)
```

از ایندکس ۷ شروع کن. بیا تا قبل ۴. چه جوری بیا؟ یکی یکی کم شو.

یه چیز جالب! پایتون (برخلاف خیلی از زبان های دیگه) ایندکس منفی هم داره.  
یعنی:

a	b	c	d	e	f	g	h
0	1	2	3	4	5	6	7
-8	-7	-6	-5	-4	-3	-2	-1

یعنی با ایندکس منفی هم می تونین کار کنین. اما یکم سخت تره. بیشتر زمانی به کار میاد که می خواین بگین مثلاً حرف آخر رو چاپ کنین. راحت می گین:

```
print(string[-1])
```

یکم با این ایندکس ها وَر برین تا بهتر درکش کنین. سعی کنین با منفی کار کنین. با عادیا. یه بار increment رو زیاد کنین یه بار کم. یه بار برین تا تهش یه بار برگرین و... خلاصه یکم باهش بازی کنین.

## 11) for + string

پایتون یه تابع خودش داره که بهتون طول یه string رو میده. تابع len. مثلاً:

```
s = 'abcde'
length = len(s)
print(length)
```



خب حالا سعی کنیم تابع بنویسیم که تعداد «a» های درون یه یه string رو بهمون بده.

```
def a_count(s):  
    count = 0  
    for i in range(len(s)):  
        if s[i] == 'a':  
            count += 1  
    return count  
string = "abca"  
print(a_count(string))
```

یه for میزنم به اندازه طول متن (یعنی از ۰ تا قبل طول متن که قبل طول متن = ایندکس آخر. چون ایندکس یکی از شمردن عادی که از ۱ هست کمتره) و دونه دونه ایندکسها رو چک می‌کنم که اگر a بودن، یکی اضافه شه.

در آخر هم مقدار تعداد رو return می‌کنم.

خوبی تابع اینه که بخوام استفاده کنم، راحت می‌زنم که تعداد a های string منو بده. دیگه کدم تمیزتره. همه می‌فهمن که منظورم اینه که تعداد رو بده. (وقتی یه تابعی رو توی پرینت می‌نویسیم، می‌گه خب این تابعس! من نمی‌دونم چیه! برم اول ببینم این تابع در آخر چیو بهم پس میده و اونو چاپ می‌کنم. درواقع print مقدار return شده یه تابع رو پرینت می‌کنه)

## نکته بسیار مهم درباره تابع!

همیشه سعی کنیم توابع رو به صورتی بنویسیم که قابلیت استفاده مجدد داشته باشن. یعنی اینطور نباشه که یه کد رو صرفاً بردین توی تابع ولی هیچ‌جای دیگه جز اونجایی که مدنظرتون بود، نشه استفاده کرد. خب این به چه دردی می‌خوره! زحمت کشیدیم که بردیم توی تابع! بلکه تابع برای اینه که من بتونم بعداً از همین تابع جاهای دیگه هم استفاده کنم! یعنی اینطور نباشه که صرفاً برای همین کار باشه!

مثلاً برای سؤال بالا، بهتر بود یه تابع یافتن تعداد یک حرف در یه string رو می‌نوشتیم. نه اینکه صرفاً بیاد برام تعداد حرف «a» رو حساب کنه. درواقع اگر اینطوری بنویسم، می‌تونم بعداً برای حرفای دیگه، جاهای دیگه به کارش ببرم و نخوام دوباره تابع رو بازنویسی یا ادیت کنم:

```
def char_count(char, s):  
    count = 0  
    for i in range(len(s)):  
        if s[i] == char:  
            count += 1  
    return count
```

```
string = "abca"
print(char_count('a', string))
```

درواقع اومدم به یه حالت کلی تر نوشتم که بشه بعداً هم استفاده کرد و صرفاً برای یه کار خاص نباشه.

هرچی تابعتون کلی تر باشه که بشه جاهای بیشتری استفاده کرد، بهتره! فرض کنین شما بعداً می‌خواین یه کد بنویسین که بره از یه فایل متنی، متن‌ها رو بخونه و علامت‌های ویرگول «،» و نقطه‌ها «.» رو حذف کنه.

به جای اینکه کدی بنویسین که توی یه تابع هم بره فایل متنی رو بخونه و هم ویرگول و هم نقطه پاک کنه، بهتره دو تابع بنویسین:

- تابعی که بره یه فایل رو بخونه.
- تابعی که یه کرکتر خاص رو از یه متن پاک کنه. مثل `remove(char, s)`
- حالا کدتون رو اینطور می‌نویسین:
- اول یه فایل خونده شه و متنش توی یه متغیر `string` ذخیره شه. ← مثلاً متغیر `s`.
- دوم تابع پاک کردن یه کرکتر خاص رو برای ویرگول صدا می‌زنین ← `remove(',', s)`
- سوم تابع پاک کردن یه کرکتر خاص رو برای نقطه صدا می‌زنین ← `remove('.', s)`

دیدین چقدر مرحله به مرحله فکر کردن بهتره؟ درواقع اگر صرفاً کدتون که خارج از تابع بود رو برید توی تابع بنویسین که هنر نکردین! هنر تقسیم کارها برای تمیزتر شدن کد و بررسی راحت تر کد هست. مثلاً من یه بار چک می‌کنم که تابع حذف کرکتر درست کار می‌کنه. دیگه نیازی نیست اون قسمت چک کنم و اگر مشکلی توی کد بود، حداقل می‌دونم که مال اون قسمت نیست. اینطوری خیلی راحت تر می‌تونم مشکلات کدمو پیدا کنم. کدم تمیزتر و خواناتر.

## 12) Comment

معمولاً برنامه‌نویسا جاهایی که نیاز به توضیح داره، یه سری توضیحات می‌ذارن. بهش می‌گن کامنت. برای خوانایی بیشتر کد.

کامنت تک خط با « شروع میشه. هرچی جلوش باشه، توسط پایتون نادیده گرفته میشه و اجرا نمیشه مثلاً:

```
def char_count(char, s): # Count the number of a char in a string
    count = 0
    for i in range(len(s)):
        if s[i] == char:
            count += 1
    return count
```

اگر می‌خواین کامنت رو توی همون خط بگذارین، با دو تا فاصله از کد قرار بدین.

کامنت رو می‌تونین توی یه خط تنها هم بگذارین. مثلاً:

```
# Count the number of 'a' in a string
def char_count(char, s):
    count = 0
    for i in range(len(s)):
        if s[i] == char:
            count += 1
    return count
```

اگر هم کامنت چند خطی می‌خواین بگذارین، از سه تا کوتیشن یا دبل کوتیشن می‌تونین استفاده کنین. مثلاً:

```
def char_count(char, s):
    ''' Count the number of a char in a string
    get: string, char
    return: the number of char in the string
    '''
    count = 0
    for i in range(len(s)):
        if s[i] == char:
            count += 1
    return count
```

تمیز نویسی:

<https://peps.python.org/pep-0008/#documentation-strings>

کامنت‌گذاری خوبه ولی به اندازه‌ش! قرار نیست همه جا کامنت بگذارین! بلکه صرفاً جاهایی که فکر می‌کنین نیاز به یه توضیح بیشتر و توضیح کاری که کردین داره. مثلاً این تابع به اندازه کافی واضح بود که چیکار می‌کنه. نیاز به کامنت نبود. صرفاً کامنت گذاشتم که یادتون بدم همچین چیزی هم هست ولی خب وقتی تابع کارش مشخصه یا اسمش کامل می‌گه داره چیکار می‌کنه، نیازه به کامنت نیست!

تمرین!

۱- با تابع برنامه‌ای بنویسین که یه یه متن بگیره، و کرکترهای ایندکس‌های زوج (یعنی ایندکس ۰، ۲، ۴ و...) رو چاپ کنه. راهنمایی: تابع یه تیکه کده. میشه توش پرینت انجام داد. پس میشه پرینت رو توی همون تابع انجام بدین!

۲- با استفاده از تابع، یه برنامه‌ای بسازین که تعداد کرکترهای بزرگ یه string رو بهمون بده. راهنمایی: بعد پاسخ ۱ راهنماییتون کردم!

۳- برنامه‌ای بنویسین که دو تا string بگیره و ببینه آیا string دومی توی اولی وجود داره یا نه؟ اگر داره چاپ کنه Yes. اگر نه چاپ کنه No.

راهنمایی؟ پاسخش وابسته به پاسخ ۲ هست!

۴- برنامه‌ای به کمک for (و نه به کمک in ... if) بنویسین که دو تا string بگیره و ببینه آیا string دومی توی اولی وجود داره یا نه؟ (از تابع استفاده شود!) ساینز string دومی هم باید بگیرین. اگر وجود داره چاپ کنه True. اگر نه چاپ کنه False. بدون گرفتن ساینز، سخته! فعلاً نمی‌خوام درگیرش شین.

پاسفنامه:

پاسخ ا:

```
def even_indexed_chars(string):  
    for i in range(0, len(string), 2):  
        print(string[i])
```

```
string = input("Enter a string: ")  
even_indexed_chars(string)
```

گفتم که از ایندکس ۰ برو تا آخر و دو تا دوتا برو که زوجا رو چاپ کنی.

- خب گفتمی همیشه که تابع رو نوشتیم تست کنیم. اینو چه جوری تست کنیم بهتره؟  
+ راه اول و ساده اینه که ورودی abcdefghi اینا بدیم و ببینیم آیا ایندکس زوج پرینت شده یا نه. اما راه هوشمندانه‌تر اینه که ورودی «۰۱۲۳۴۵۶» رو بدیم. یعنی امتناظر هر ایندکس عددشو گذاشتیم. و خب چون متنی می‌گیریم، پایتون به شکل یه متن نگاش می‌کنه و نه عدد. خب خروجی چی باید باشه؟  
+ ۰ و ۲ و ۴ و ۶! اینطوری نیاز نیست هی تطابق بدیم و بگیم c ایندکسش چند بود و اینا. با ورودی خوب، تست هوشمندانه‌تری انجام میدیم.

ورودی «۰۱۲۳۴۵۶۷» هم میدیم که مطمئن شیم با این گروه دوم test cast هم درست جواب میده. (فرقش با قبلی اینه که ایندکس آخری فرده و قبلی زوج بود. یعنی یه گروه متفاوت تست کیسه)  
برنامه‌نویسی همش خلاقیتیه! هوشمندتر باشین! یا حتی من بهتون slicing رو گفتم. پس بدون تابع سعی کنین پیاده‌سازیش کنین که سریع‌تر و کوتاه‌تر و احتمالاً به خاطر کوتاه‌تر بودن و ساده‌تر بودنش، خطای انسانی و باگ کمتره!

```
string = input("Enter a string: ")  
print(string[::2])
```

از ۰ برو تا آخر. ۲ تا ۲ تا برو جلو.

اما خب یه راه دیگه هم میشه for رو برای string ها پیاده سازی کرد. اینطوری:

```
def a_count(string):  
    count = 0  
    for char in string:  
        if char == "a":  
            count += 1  
    return count
```

اینطوری ترجمه میشه:

برای تک تک حروف درون string که اسمشو char گذاشتم. این اسم میتونه هرچی باشه. می تونین بذارین:

```
for hello in string
```

ولی خب همیشه گفتیم اسما با معنی باشه. یعنی برای کرکتهایی درون string. یعنی دونه دونه کرکترها رو طی می کنه. یعنی اولین بار char، کرکتر ایندکس ۰ هست. بعدش کرکتر ایندکس ۱. بعدش کرکتر ایندکس ۲ و الی آخر. اینطوری با ایندکس کار نمی کنیم و خواناتر هم هست!

پاسخ ۲:

```
def uppercase_count(string):  
    count = 0  
    uppercase_chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"  
    for char in string:  
        if char in uppercase_chars:  
            count += 1  
    return count
```

```
string = input("Enter a string: ")  
print(uppercase_count(string))
```

یه متغیر ساختم که تمام حروف بزرگ رو توش گذاشتم. بعد گفتم برای تک تک کرکترهای string ام، اگر کرکتر توی uppercase\_chars بود، یه دونه به تعداد اضافه کن.

پاسخ ۳:

```
input_string = input("Enter a string: ")  
sub_string = input("Enter a substring: ")  
  
if sub_string in input_string:
```

```
print("Yes")
else:
    print("No")
```

به همین سادگی! می‌گیم اگر دومی توی اولی بود، چاپ کن Yes. اگر نه، چاپ کن No.

پاسخ ۴:

خب فرض کنیم زیرمجموعه یا sub\_string یا همون استرینگ که می‌خوایم ببینیم توی اولی هست یا نه، سائزش ۳ هست. باید روی اولی حرکت کنیم و هی سه تا سه تا جدا کنیم و ببینیم با sub\_string برابره یا نه؟

```
abcdefgh
abcdefgh
abcdefgh
abcdefgh
abcdefgh
abcdefgh
abcdefgh
```

اینطوری تمام سه‌تایی‌های پشت هم رو چک کردیم که آیا توش هستن یا نه؟ همیشه با خودتون فکر کنید که چه‌جوری باید حل کنید و بعد الگوریتم رو بنویسین. یعنی slice بندی می‌کنیم. اینطوری:

```
s[0:3]
s[1:1+3]
s[2:2+3]
...
```

خب حالا کلی‌ترش کنیم. به جای ۳، سائز sub\_string رو می‌گذاریم و با for یکی یکی میریم جلو:

```
def if_in_string(input_string, sub_string):
    sub_string_size = len(sub_string)
    for i in range():
        if input_string[i:i+sub_string_size] == sub_string:
            return True

    return False
```

خب for رو نمی‌دونم تا کجا پیش ببرم. توی پایتون اگر به ایندکسی بخواین دسترسی پیدا کنید که خارج از مکزیموم یا مینیموم ایندکس استرینگ باشه، بهتون ارور زیر میده:

```
IndexError: string index out of range
```

میگه توی رنج نیست! سائز استرینگت ۳ هست. تو به من میگی مثلاً s[100] رو نشون بده؟ چطوری؟ ۳ تا کرکتر توشه. من چجور ایندکس ۱۰۰ رو نشونت بدم؟!

برای همین من در قدم اول نمی‌دونم for باید تا کجا پیش بره؟ نمیدونم! پس فعلاً بقیه کد رو نوشتم که بفهمم for باید تا کجا پیش بره که out of range نخورم. ببینین قرار نیست همیشه کد رو همون اول بنویسین! بلکه کلیت رو فکر می‌کنین و قدم به قدم پیش میرین و تکمیلش می‌کنین.

- خب کجا ممکنه out of range رخ بده؟  
+ قسمت if. چون داریم میگم از i تا i + sub\_string\_size پیش برو.  
خب بیایم مقدار بدیم که بتونیم درکش کنیم:

```
input_string = "0123456"  
sub_string = "123"
```

خب سائز sub ما ۳ هست. حداکثر ایندکس input ما هم ۶ هست. پس i + sub\_string\_size حداکثر باید ۷ بشه. چون میگیم از i تا قبل i + sub\_string\_size پیش برو. یعنی درواقع بخوایم تا آخرین ایندکس input رو پوشش بدیم، باید بگیم مثلاً sub\_string == input\_string[4:7]. قبول دارین این آخریشه دیگه؟

یعنی آخری، i چهار بوده:

```
input_string[i:i+3] == sub_string
```

یا درواقع می‌تونیم بگیم که:

```
i + 3 = input_size  
i + 3 = 7  
i = 4
```

پس مقدار i برای for ما، حداکثر باید ۴ شه. یا درواقع for ما باید بگیم تا range(5) بره. یا درواقع:

```
7 - 3 + 1  
input_size - sub_size + 1
```

یا درواقع:

```
for i in range(len(input_string) - sub_string_size + 1):
```

پس کد رو تکمیل کنیم:

```
def if_in_string(input_string, sub_string):  
    sub_string_size = len(sub_string)  
    for i in range(len(input_string) - sub_string_size + 1):  
        if input_string[i:i+sub_string_size] == sub_string:  
            return True  
  
    return False
```

```
input_string = input("Enter a string: ")
sub_string = input("Enter a substring: ")
print(if_in_string(input_string, sub_string))
```

out of range یکی از مهم‌ترین ارورهاست. همیشه حواستون باشه که موقعی که اسلایس بندی می‌کنین یا به ایندکسی دسترسی پیدا می‌کنین، مشکل پیش نیاد! لزوماً قرار نیست شرط while و for رو همون لحظه بنوسین! بلکه ببینین کجا ممکنه out of range پیش بیاد و بر اساس اون تعیینش کنین!

### مرفه‌ای باش!

برنامه‌نویس خوب کسیه که بتونه از ارور جلوگیری کنه. جوری فکر کنه که اگر یه روند نامتعارفی رخ داد چی؟ به نظرتون اینجا روند نامتعارف چی می‌تونه باشه؟ - من فکر می‌کنم که اگر کاربر sub\_string رو جوری بده که سایش بزرگ‌تر از string اصلی ما باشه، توی for مشکل به وجود میاد. چون مثلاً میشه:

```
3 - 7 + 1
```

+ آفرین! برنامه نویس خوب باید همیشه همه جنبه‌ها رو در نظر بگیره. همیشه باید بگه اگر کاربر مطابق چیزی که من می‌خواستم رفتار نکرد چی؟ ببینه اولی از دومی کوچکت‌ر نیست؟!<sup>۵۵</sup>

## 13) String concatenation

کد زیر رو اجرا کنین ببینین چی میشه؟

```
s1 = 'Hello'
s2 = 'World'
s3 = s1 + s2
print(s3)
```

- عه مگه میشه دو string رو با هم جمع کرد؟ + آره! اجراش کنین ببینین چی میشه؟ خب همونطور که از اسم رشته (string) معلومه، مثل یه رشتس. با جمع کردن، رشته بعدی، بهش متصل میشه! یعنی درواقع concatenate میشن با هم. به هم اضافه میشن و می‌چسبن به هم.

---

<sup>۵۵</sup> در بعضی از زبان‌ها گیر نمیدن ولی در زبانای زیادی ارور میخورین.



حالا فرض کنیم من نام و نام خانوادگی رو جداگانه گرفتم و می‌خوام توی یه متغیر با یه فاصله از هم قرارش بدم. مثلاً:

```
first_name = 'Bruce'
last_name = 'Schneider'
```

رو داریم. و می‌خوام یه `full_name` داشته باشیم که اینطوری باشه:

```
full_name = 'Bruce Schneider'
```

خب با `concatenation` بسازینش.

```
full_name = first_name + ' ' + last_name
```

توضیح: اول `first_name` بعد یه فاصله بهش اضافه کن و بعد `last_name` رو اضافه کن. همشو بریز توی `full_name`.

یا می‌تونستیم توی `print` هم انجامش بدیم:

```
print(first_name + ' ' + last_name)
```

یادتونه اوایل توی `print` گفتیم که با کاما می‌تونیم چیزا رو چاپ کنیم؟ خب اینجا هم می‌تونیم از کاما استفاده کنیم. کاما خودش فاصله میده. پس نیاز به گذاشتن فاصله توسط ما نیست:

```
print(first_name, last_name)
```

پس حالا فهمیدین که توی پرینت، بین دو `string`، می‌تونین علامت جمع هم بگذارین. فقط توجه کنین که علامت جمع بین دو چیز یکسان کار می‌کنه. یعنی شما نمی‌تونین ۲ رو با یه `string` جمع کنین! همیشه! ولی `string` رو با `string` می‌تونین!

من با عمد همون اول توی `print`، علامت جمع برای `string` رو بهتون یاد ندادم و گذاشتم برای اینجا. چون اگر بخوایم همه نکات رو همون لحظه بگیم، یادتون میره و شیوه خوبی نیست. یه دفعه با انبوهی از نکات مواجه میشین که معلوم نیست کجا به کار میرن! ولی گذاشتم جا و زمان درستش بهتون بگم که دقیق درکش کنین (:

یکی از تفاوتایی که خواستم آموزش با بقیه جاها داشته باشه این بود که مفاهیم رو پله‌پله بگم و شما رو یکدفعه درگیر هزار تا نکته نکنم (:

## تمرین!

• عدد `n` را ورودی بگیرین. سپس کوچکترین عدد بزرگ‌تر از اون رو پیدا کنین که انتهای اون، عدد `n` ما تکرار شده باشه و همچنین بر خود `n` هم بخش‌پذیر باشه.

input:

14

output:<sup>56</sup>

714

-----

---

۵۶ اولین عدد بعد ۱۲ که در پایانش ۱۲ رو داره و بر ۱۲ بخش‌پذیره، ۳۱۲ هست.

input:

2

output:

12

-----

input:

217

output:

217217

-----

- ۱- برنامه‌ای بنویسین که یه رشته و یه کرکتر  $c$  و یه عدد به عنوان  $n$  بگیره و کرکتر  $n$  ام (رشته رو به  $c$  تغییر بده). (دوست دارم به نقاط نامتعارف برنامه که ممکنه به ارور بر بخوره یا نخوره رو فکر کنین)
- ۲- یه رشته رو بگیرین و برعکسش رو چاپ کنین.
- ۳-

<https://quera.org/problemset/591/>

راهنمایی؟ پاسخش وابسته به سؤال قبلیه!

- ۳.۱ - عدد باینری، عددی عجیب و غریب نیست! سیستم باینری، نحوه ذخیره‌سازی اعداد توی کامپیوتر هستن و صرفاً از ۰ و ۱ تشکیل شدن. مثلاً یه عدد باینری به طول ۸:

01010110

به شما یه عدد باینری داده میشه. تضمین میشه که طول این عدد باینری از ۰ تا ۳۲ هست. (خود ۰ و ۳۲ هم جزئشه!)

کار شما اینه که این عدد باینری رو هی با اضافه کردن رقم اولش در سمت چپ، گسترش بدین تا به ۳۲ رقم برسه. (اکستند کنین).<sup>۵۷</sup>

input:

01010101101110

output:<sup>۵۸</sup>

00000000000000000000000001010101101110

-----

input:

101010

output:<sup>۵۹</sup>

1111111111111111111111111111101010

<sup>۵۷</sup> به این کار می‌گن «Binary Sign-Extension».

<sup>۵۸</sup> با صفر اکستند شده. چون عدد باینری ما سمت چپش صفر هست.

<sup>۵۹</sup> با یک اکستند شده. چون عدد باینری ما سمت چپش یک هست.

توجه! اگر عدد باینری ما طولش صفر باشه (یعنی صرفاً یه enter بزنیم)، هیچ خروجی نباید بدین!

۴- برنامه‌ای بنویسین که یه عدد به صورت string بگیره و تعداد ارقام فرد عدد را حساب کنه.

۵- برنامه‌ای بنویسین که یه عدد بگیره و بدون استفاده از عملگر توان، بزرگ‌ترین توان دو کوچکتر از اون عدد رو به صورت عدد صحیح چاپ کنه. مثال:

input: 5

output: 4

```
input: = 62
```

```
output: = 32
```

۶- یک عدد گرفته و فاکتوریش را با کمک یک تابع حساب کنید. بعدش تعداد صفرهای سمت راست عدد را چاپ کنید. این کار را هم با کمک ریاضی و هم با کمک `string` انجام دهید.

چندتا تست کیس با مقدار فاکتوریل. (مقدار فاکتوریل هم دادم بهتون که اگر کدتون اشتباه کار می‌کرد، اول مقدار فاکتوریل رو حساب کنید و ببینید اگر مقدار فاکتوریلتون درسته، پس مشکل اون نیست. همیشه بگین خب کجاها ممکنه نباشن که بگذاریمشون کنار)

input: 5

*factorial:* 120

output: 1

input: 100

*factorial:*

93326215443944152681699238856266700490715968264381621468592963895  
21759999322991560894146397615651828625369792082722375825118521091  
686400000000000000000000000000

output: 24

input: 76

*factorial:*

18854947016660502549879322608611465582303945353793293356724879829  
618440434955379231177299722240000000000000000

output: 18

۷- په عدد طبيعي، بگيږين و به ازای مقدار هر رقم، اون عدد رو اونقدر بار چاپ کيږي. مثال:

input: 1234

output:

1: 1  
2: 22  
3: 333  
4: 4444

۸- یه string و یه عدد بگیرین. به تعداد عدد، string رو rotate کنین.

input:

abcd  
1

output:

bcda

-----

input:

abcd  
2

output:

cdab

-----

input:

abcd  
4

output:

abcd

-----

input:

abcd  
10

output:

cdab

پاسفنامه:

پاسخ:۰:

در نگاه اول شاید بگیم که خب مثلاً  $i$  از  $n+1$  شروع می‌کنم و میرم تا بالا. هر عدد رو چک می‌کنم که آیا این عدد ما توی راستش تکرار شده یا نه و بعد چک می‌کنم ببینم بخش‌پذیره یا نه.

اما یکم فکر کنین! این راه خیلی در دسر داره! اول بیینی دونه دونه رقمای  $n$  توی ارقام  $i$  وجود دارن یا نه؟! بعدش بخش پذیر. بعدشم هی یه دونه یه دونه داریم میریم بالا. این کار بسیار بسیار زمان بره! بیایم یه راه میونبر بزنینم!

بگیم صرفاً روی اعدادی پیش برو که سمت راستشون  $n$  هست. اینطوری عدد  $n$  همیشه راست `string` ما هست. با یه متغیری به اسم  $i$  میایم هی اول اون متنمون رو عوض می کنیم:

```
i = 1
n = 14

i = 1 → 114
i = 2 → 214
i = 3 → 314
...
i = 9 → 914
i = 10 → 1014
i = 11 → 1114
...
```

خب چطور این رو به کد بنویسیم؟

اول نیاز داریم  $n$  رو بگیریم. بعد برای اینکه بتونیم  $i$  رو به اولش اضافه کنیم، نیاز داریم `string` اش هم داشته باشیم. بعد هی  $i$  رو به اولش اضافه می کنیم و چک می کنیم ببینیم اوکی هست یا نه؟

```
n = int(input())
n_str = str(n)
i = 1
while True:
    res = str(i) + n_str
    if int(res) % n == 0:
        print(res)
        break
    i += 1
```

اول یه `while` بی نهایت می داریم. تا وقتی که جواب پیدا نشه (`break`) صورت نگیره، ادامه پیدا کنه. بعدش هر بار میایم یه `res` می گیریم به عنوان پاسخ. که `res` هر دفعه از اضافه شدن `str(i)` به ابتدای عدد ما که به استرینگ ذخیره شده به دست میاد.

**تست کیس خودساخته و افیشتن اپروچ برای  $10^{25}$**

پاسخ ا:

در نگاه اول شاید بگیم که:

```
s = '11111111'
```

```
s[1] = '0'
print(s)
```

اما جواب نمیده! چون string ها اصطلاحاً immutable (تغییرناپذیر) هستن. همونطور که دیدین من مقدار همش رو ۱۱۱۱۱ دادم و مقدار جدید رو ۰ که بهتر تمایز و تغییر رو ببینم. (تست هوشمندانه‌تر)

پس راه چیه؟ بیایم بگیم که حالا که همیشه یه قسمت رو عوض کرد، بگیم از این به بعد این string مقدارش برابر:

تا جایگاه n (ایندکس n-1) مقدار قبلی، ایندکس n-1 مقدار جدید و از ایندکس n به بعد هم مقدار قبلی.

این با چی ممکن بود؟ با slicing و concatenation.

```
input_string = input("Enter a string: ")
c = input("Enter a character: ")
n = int(input("Enter the place of char: "))

input_string = input_string[:n-1] + c + input_string[n:]
print(input_string)
```

خب اما این ارور هندلینگ نداره! بیایم انجامش بدیم. بهم بگین کجاها ممکنه کاربر خطا کنه و روند درست برنامه طی نشه؟

- خب من فکر می‌کنم که همون اول کاربر ممکنه کرکتر وارد نکنه! یعنی مثلاً یه رشته وارد کنه و چند کرکتری باشه! پس من می‌تونم یه if بگذارم و طول c رو چک کنم که اگر برابر ۱ نبود، یه چیزی چاپ کنه.

- و فکر می‌کنم حتی ممکنه n رو یه عدد پرت بده. مثلاً ۱۰۰. اینم ممکنه مشکل به وجود بیاره. + هر دو مورد بالا ممکنه رخ بده ولی خب ارور به وجود نمیارن. صرفاً روند نامتعارفن. ولی خوبه بهشون فکر کردین!

پاسخ ۲:

روش اول:

اول یه متغیر رشته ولی خالی رو می‌سازیم که رشته معکوس رو توش بریزیم. میایم از ایندکس آخر (طول رشته - ۱) شروع می‌کنیم و تا ۱- پیش میریم. یکی یکی کم می‌کنیم و دونه‌دونه کرکترها رو توی متغیر جدید میریزیم.

```
input_string = input("Enter a string: ")
reverse_string = ''
for i in range(len(input_string) - 1, -1, -1):
```

```
reverse_string += input_string[i]
```

```
print(reverse_string)
```

- چرا تا ۱- پیش رفتیم؟  
+ گفتیم تا ۱- یعنی یکی قبل تر از اون. یعنی تا ایندکس ۰ پیش میره!

روش دوم:

```
input_string = input("Enter a string: ")
```

```
print(input_string[::-1])
```

درواقع وقتی چیزی ندیم بهش، اگر step (اینکه چه اندازه چه اندازه بره جلو) رو مثبت بدیم، خودش اولی رو صفر میده و میره تا آخر.  
اما اگر step رو منفی بدیم، میگه شروع از آخر و بیاد تا اول.

روش سوم:

توی همون for بخوایم پرینتش کنیم، اینطوری میشه.

```
input_string = input("Enter a string: ")
```

```
for i in range(len(input_string) - 1, -1, -1):
```

```
    print(input_string[i])
```

خب مشکلش چیه؟ مشکلش اینه که هر بار که print انجام میشه، به واسطه خاصیت print، حروف توی خطهای مجزا چاپ میشن. پس چیکارش کنیم؟  
پایتون راه حل داده! گفته یه پارامتر دیگه هم توی print به من بده و بگو که هر بار که خط پرینت رو اجرا کردم، چیکار کنم؟ یه فاصله بدم؟ یه اینتر بزنم؟ اصلاً فاصله ندم؟ چیکار کنم. اینطوری بهش می گیم

```
print('a', 'b', end='$')
```

```
print('c')
```

output:

a b\$c

میگه خط پرینت اولی رو که اجرا کردم، تهش چی پرینت کنم. بهش میگم که علامت «\$» رو پرینت کن. (دیگه enter نمی زنه).  
یا مثلاً می تونیم بگیم:

```
print('a', 'b', end='$')
```

```
print('c')
```

output:

a bc

میگه خب بهم گفتی یه استرینگ خالی (هیچی) رو چاپ کن و برو دستور بعدی.  
حالا بریم سراغ سؤال اصلیمون:

```
input_string = input("Enter a string: ")
```

```
for i in range(len(input_string) - 1, -1, -1):
    print(input_string[i], end='')
```

میگیم هر بار پرینت تموم شد. هیچی چاپ نکن. که بعدش میای دوباره پرینت کنی، پرینت جدید، ادامه پرینت قبلی باشه.

حالا اجرا کنین میبینین که کرکترها پشت هم چاپ میشن و همچنین هیچ اینتری نداره. این هم یه نکته دیگه از print که در زمان نیاز یادش گرفتیم. نه اینکه توی پرینت تمام چیزاشو یهو بهتون نکته وار بگم که فراموش کنین. هرچیزی باید زمان نیازش یاد گرفته شه :

پرینت یه پارامتر دیگه هم می تونه بگیره به نام «sep» مثل همین end ولی زماناییه که شما چندتا چیز توی print چاپ می کنین. کما در حالت پیش فرض میگه یه فاصله بده بین چیزا. ولی خب شما می تونین با «sep» دلخواه بگین بین چیزا چیکار کنه. مثلاً مقایسه کنین:

```
print('Hello', 'World', sep='@')
```

بینشون به جای فاصله، علامت «@» قرار میده. یا مثلاً:

```
print('Hello', 'World', sep='AAAAA', end='')
```

یعنی بین چیزا «AAAAA» رو چاپ کن و در پایان پرینت هم چیزی چاپ نکن. یعنی \n که اینتر بود رو چاپ نکن!

درواقع یادتونه \n و \t چی بودن؟ یه سری کرکتر که می گفت که اینجا باید یه اینتر زده شه توی چاپ کردن. اونجا باید یه tab زده شه. از اینا میشه کمک گرفت و باهاشون بازی کرد.

پاسخ ۳:

روش اول:

خب بیایم مسأله رو به چند بخش تقسیم کنیم.

بخش ۱: چاپ کردن ضلع بالا.

بخش ۲: چاپ کردن دو ضلع کناری.

بخش ۳: چاپ کردن ضلع پایین.

بخش ۱:

من به تعداد n پیام for بزنم و ستاره‌هایی پشت هم چاپ کنم:

```
for i in range(n):
    print('*', end='')
```

خب اجراش کنیم می بینیم بخش ۱ رو انجام دادیم.

بخش ۲:

باید یه string ای چاپ کنم یه دو تا ستاره دوطرف و بینش n-2 تا فاصله باشه. پس string رو

می سازم و با for پرینتش می کنم. چند بار؟



کل ضلع مربع  $n$  تاست. بالا و پایین که جدان. پس بینش  $n-2$  بار باید چاپ شه.

```
def print_square(n):  
  
    for i in range(n):  
        print('*', end='')  
  
    middle = ''  
    for i in range(n-2):  
        middle += ' '  
    middle = '*' + middle + '*'  
  
    for i in range(n-2):  
        print(middle, end='\n')
```

یعنی اومدم گفتم اسم این string ما spaces هست که  $n-2$  تا فاصله رو کنار هم قرار دادم. بعد متغیری ساختم به نام middle که شامل اون فاصله‌ها و دو ستاره کناری هست. بعد هم توی for آخری،  $n-2$  بار کل string رو چاپ کردم.

تا اینجا اجراش کنیم (با مثلاً `print_square(5)`) که ببینیم درسته؟  
عه چرا بد چاپ شد؟

یکم دقت کنیم که چرا اولین خط middle پشت قبلی چاپ شده؟  
آها! چون آخرین `print`، مقدار `end` اش هیچی بود. پس یعنی فاصله نداده بود بین این و قبلی. پس بینشون یه پرینت عادی می‌کنیم که درست شه. و خب در نهایت هم عیناً for اولی رو تکرار می‌کنیم که ضلع پایینی (بخش ۳) هم انجام شه:

```
def print_square(n):  
  
    for i in range(n):  
        print('*', end='')  
  
    print()  
  
    middle = ''  
    for i in range(n-2):  
        middle += ' '  
    middle = '*' + middle + '*'  
  
    for i in range(n-2):  
        print(middle, end='\n')
```

```
    for i in range(n):
```

```
print('*', end='')
```

```
print_square(5)
```

روش دوم:

یادتونه گفتیم که همیشه یه عدد صحیح رو با یه رشته جمع زد؟ جمع همیشه درسته، اما پایتون اجازه میده شما یه رشته رو در یه عدد ضرب کنی! بله! درست متوجه شدین! مورد زیر رو امتحان کنین:

```
print('*' * 4)
```

چهار بار رشته رو تکرار می‌کنه. حالا با این، سؤالمون رو تمیزتر حل می‌کنیم:

```
def print_square(n):
```

```
    print('*' * n)
```

```
    middle = ' ' * (n-2)
```

```
    middle = '*' + middle + '*'
```

```
    for i in range(n-2):
```

```
        print(middle)
```

```
    print('*' * n)
```

```
print_square(5)
```

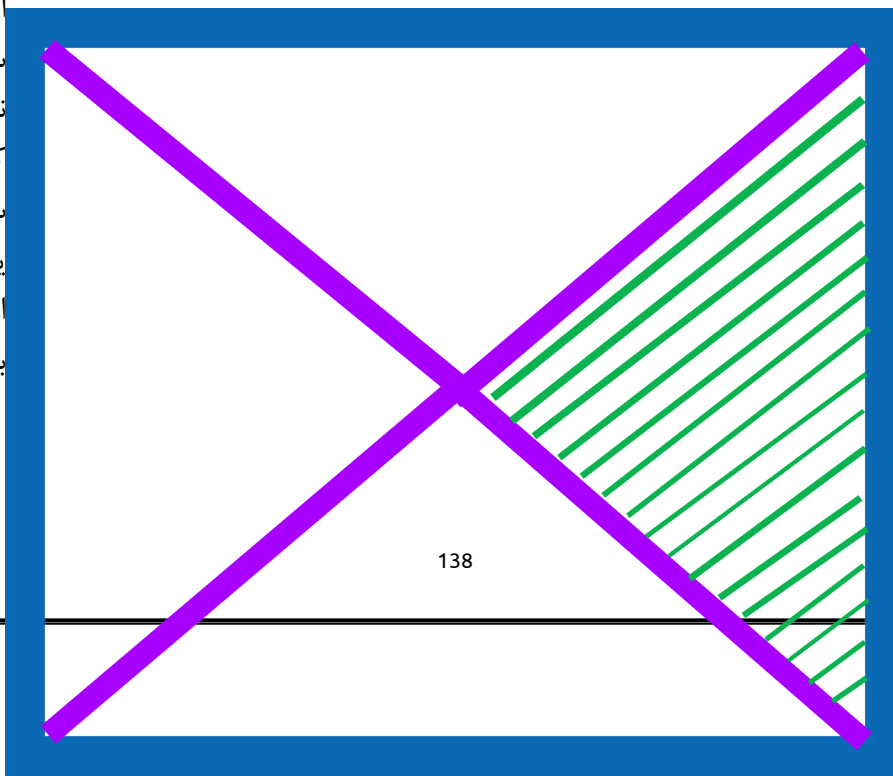
تمیزتر نشد؟

تمرین بیشتر:

<https://quera.org/problemset/283/>

<https://quera.org/problemset/296/>

اگر توی  
سؤال دومی  
نمی‌دونین  
که یه چهارم  
سمت راست  
یعنی چی،  
این شکل رو  
بینین:



درواقع می‌گه قطرا رو بکش. قطرا رو که با هشتگ کشیدی، سمت راست رو هم پر کن.

### پاسخ ۱.۳:

بیایم اول یه عدد باینری بگیریم. این عدد باینری رو به صورت همون `string` می‌گیریم. چرا؟ چون می‌دونم این استرینگ هست که می‌تونم اکستندش کنم و یه چیزی به اول و آخرش به سادگی و با `concatenate` کردن اضافه کنم.

میگم اگر کرکتر اولش برابر یک بود، خب یه سری یک باید اولش اضافه شه. چه تعداد؟ به اندازه‌ای که با عددمون بشه ۳۲ رقم. خب پس ۳۲ منهای سائز عددمون تا ۱ اضافه شه.

```
bin_str = input()
n = len(bin_str)

def sign_extend(n, bin_str):
    if bin_str[0] == "1":
        bin_str = "1"*(32-n) + bin_str
```

خب برای حالتی که صفر هم تهش بود، اضافه می‌کنیم:

```
bin_str = input()
n = len(bin_str)

def sign_extend(n, bin_str):
    if bin_str[0] == "1":
        bin_str = "1"*(32-n) + bin_str
```

```

print(bin_str)
elif bin_str[0] == "0":
    bin_str = "0"*(32-n) + bin_str
print(bin_str)

```

```
sign_extend(n, bin_str)
```

خب فکر کنین ببینین کدمون چیزی کم نداره؟ راهنمایی: به تست کیس‌ها و نقاط مرزی دقت کنین. خب ببینین مشکل کد ما زمانیه که سایز عدد باینریمون صفر باشه. اینطوری اصلاً ایندکس صفر رو نداره که بخواد ببینه ۱ یا صفر هست! پس ارور می‌خوره! (امتحان کنین! موقع ورودی دادن، صرفاً به اینتر بزنین.)

پس من باید قبل اینکه `bin_str[0]` رو استفاده کنم، چک کنم ببینم سایز صفر نباشه. برای این کار از `and` استفاده می‌کنم.<sup>۶۰</sup>

```

bin_str = input()
n = len(bin_str)

def sign_extend(n, bin_str):
    if n != 0 and bin_str[0] == "1":
        bin_str = "1"*(32-n) + bin_str
        print(bin_str)
    elif n != 0 and bin_str[0] == "0":
        bin_str = "0"*(32-n) + bin_str
        print(bin_str)

```

```
sign_extend(n, bin_str)
```

پرینت رو هم توی خود `if` و `elif` انجام دادم. چون اگر بیرونشون انجام می‌دادم، روی همین حالت سایز برابر صفر، نباید چیزی چاپ میشد، اما به واسطه وجود

```
print(bin_str) → print('')
```

یه خط خالی چاپ می‌شد. که نباید میشد! (کلاً هر وقت پرینت داشته باشین، یه `new line` هم انگار دارین.)

پاسخ ۴:

قاعدتاً باید روی تک‌تک کرکترها حرکت کنم و تشخیص بدم اون کرکتر عدد فرده یا نه؟

روش اول:

```
def odd_digit_count(num_string):
```

<sup>۶۰</sup> چون گفتیم `and` صرفاً زمانی میره سراغ شرط دوم که اولی درست باشه. اگر اولی درست نباشه، اصلاً براش مهم نیست دومی چیه! نمیره سراغ دومی! این رو توی سرفصل «Boolean» بررسی کردیم!

```

count = 0
for char in num_string:
    if char in '13579':
        count += 1
return count

```

روی تک تک کرکترها حرکت می‌کنم و هر وقت که کرکتری توی رشته «۱۳۵۷۹» بود، فرد و یکی اضافه می‌کنم.

روش دوم:

```

def odd_digit_count(num_string):
    count = 0
    for char in num_string:
        if int(char) % 2 == 1:
            count += 1
    return count

```

این دفعه از cast کردن استفاده کردم. یعنی اول تبدیل به عدد صحیحش کردم و بعد حالا اون باقی‌مانده گرفتم که ببینم فرد و یا نه.

پاسخ ۵:

خب باید بگیم هی توان ۲ ها رو برو جلو. تا کجا بره جلو؟ تا وقتی کوچکتر از عدد باشه. این رو به صورت while می‌نویسیم:

```

def square_below(num):
    result = 2
    while result < num:
        result *= 2 # result = result * 2

    return result // 2

```

```

input_num = int(input("Enter a number: "))
print(square_below(input_num))

```

همونطور که دقت کردین، من یه کامنت کنار `result *= 2` گذاشتم که نشون بدم درواقع یعنی چی. کامنت توسط پایتون نادیده گرفته میشه و فقط برای خوانایی هست.  
- خب به نظرتون چرا مقداری که برگردوندم، تقسیم صحیح بر ۲ داره؟

+ بیایم `while` رو بررسی کنیم. چه زمانی از `while` خارج میشه؟ زمانی که `result > num` بشه. پس یعنی چیزی که ما می‌خوایم نیست! ما می‌خواستیم `result < num` بشه. ولی شرط بیرون اومدن اینه که `result` بزرگ‌تره. پس درواقع یه ضربدر ۲ اضافه هست. پس تقسیم بر ۲ می‌کنیم که اون از بین بره.

- حالا چرا تقسیم صحیح بر ۲؟

+ چون گفتم مقدار صحیح رو بهم بده. وگرنه به جای مثلاً ۸ بهم میداد ۸.۰. تقسیم عادیش کنین و امتحانش کنین تا متوجه شین که اعشاری می‌ده.

- خب به نظرتون مشکل و نقطه حساس این چیه؟

راهنمایی: همونطور که گفتم نقاط حساس رو چک کنین. گروه تست کیس رو چک کنین.

+ پاسخ: نقطه حساس اینه که من ورودیم توان ۲ باشه. مثلاً ۶۴. باید بهم ۳۲ بده. چک کنیم هم این کار رو انجام می‌ده.

خب به نظرتون اگر راه‌حل رو اینطور می‌نوشتیم، مشکل کد چی بود؟

```
def square_below(num):  
    result = 2  
    while result < num:  
        result *= result # result = result * result  
  
    return result // 2
```

```
input_num = int(input("Enter a number: "))  
print(square_below(input_num))
```

خب به تفاوتش با کد قبلی دقت کنین! من به جای اینکه هر بار ضربدر ۲ کنم، ضربدر `result` کردم. مقدار اولیه `result` هم برابر ۲ هست. خب چه تفاوتی داره به نظرتون؟!

راهنمایی ۱: `result` یه متغیر بود درسته؟ خب امیدوارم کمک کرده باشه!

راهنمایی ۲: متغیر تغییر می‌کرد درسته؟!

پاسخ:

متغیر تغییر می‌کنه. مقدار `result` اول ۲ هست، بعدش ۴، بعدش ۸ و... یعنی بار اول ضربدر ۲، بار بعدی ضربدر ۴ و بعدش ۸ و... میشه! درواقع مشکل اینه که هر بار ضربدر ۲ نمیشه! توان اینطوریه دیگه:

```
2 ** 1 = 2  
2 ** 2 = 2 * 2  
2 ** 3 = 2 * 2 * 2  
2 ** 3 = 2 * 2 * 2 * 2
```

یعنی هر بار یه ضربدر ۲ اضافه میشه. نه اینک هر بار ضربدر خودش شه!

پاسخ ۶:

روش اول:

خب اول روش‌های محاسبه فاکتوریل رو حساب می‌کنیم:

```
def factorial(num):  
    result = 1  
    for i in range(1, num + 1):  
        result *= i  
    return result
```

از ۱ تا خود عدد پیش میرم. دونه دونه ضربدر هم می‌کنم و result رو می‌سازم.

روش دومش:

```
def factorial(num):  
    result = 1  
    while num > 0:  
        result *= num  
        num -= 1  
    return result
```

این دفعه به کمک while از عدد تا ۱ رو در هم ضرب می‌کنم و result رو می‌سازم.

خب بریم سراغ تابع محاسبه تعداد صفر به کمک string.

یادتونه گفتیم تابع یه تیکه کده که یه اسم براش انتخاب کردیم؟ همیشه حواستون باشه که تابع باید یه چیز گلی باشه. یعنی برای مسائل زیادی بتونه جواب بده که نخوایم هی عوضش کنیم. مثلاً اینجا من قراره یه تابع بنویسم و اون تابع برام تعداد صفراشو اضافه کنه. این خیلی بهتر از اینکه یه تابع بنویسم و هم فاکتوریل حساب کنه و هم تعداد صفرا!!

بهتره دو تابعش کنم. یکی فاکتوریل حساب کنه و یکی تعداد صفر.

شما توابع کوچیک کوچیک زیادی می‌نویسین که اون توابع در کنار هم عمل خواهند کرد و این خیلی بهتره! این خیلی بهتر می‌تونه باشه و گلی‌تر.

پس یه تابع می‌نویسم که یه عدد بگیره و تعداد صفر بده. عدد رو به صورت عدد صحیح (integer) پاس میدم. چون منطقی‌تره که فرض کنم کسی که داره با تابع کار می‌کنه، یه عدد صحیح میده و نه یه string عددی. حالا خودم بر حسب نیازم توی تابع تبدیل به string اش می‌کنم.

خب اول با کمک string:

عدد رو تبدیل به string می‌کنم و بعدش از ایندکس آخر استرینگ (راست‌ترین کرکتر) شروع می‌کنم و هی میام چپ‌تر. هر بار به صفر خوردم، یکی به تعداد اضافه می‌کنم و اولین کرکتر غیر صفر رو دیدم از اون حلقه‌ام می‌پرم بیرون.

```
def right_zeros(num):
```

```

num_string = str(num)
count = 0

for i in range(len(num_string)-1, -1, -1):
    if num_string[i] == '0':
        count += 1
    else:
        break

return count

```

اول عدد رو تبدیل به string می‌کنم. اسمش می‌گذارم به string هست. اسمش می‌گذارم num\_string.

بعدش از سمت راست یا index آخر که برابر  $len - 1$  هست میام سمت چپ و دونه‌دونه صفرا رو می‌شمرم. هر جا هم دیدم صفر نیست (بلاک else)، break می‌کنم. یعنی از for پیر بیرون! حالا به بار کد رو در کنار هم ببینیم:

```

def factorial(num):
    result = 1
    for i in range(1, num + 1):
        result *= i
    return result

def right_zeros(num):
    num_string = str(num)
    count = 0

    for i in range(len(num_string)-1, -1, -1):
        if num_string[i] == '0':
            count += 1
        else:
            break

    return count

num = int(input('Enter a number: '))
num_factorial = factorial(num)
print(right_zeros(num_factorial))

```



اول فاکتوریل رو با کمک یه تابع حساب کردم و مقدارش رو ریختم توی یه متغیر به نام num\_factorial (اسمش بامعنا گذاشتم که کسی که کد می‌خونه متوجه شه فاکتوریل) بعدش مقدار فاکتوریل رو پاس می‌دم به right\_zeros که تعداد صفرها رو return کنه. مقدار return در حالت عادی به چه دردم می‌خوره؟ باید چاپش کنم که مقدار return شده چاپ شه.

البته می‌شد با ایندکس منفی هم پیش رفت. یعنی:

```
def right_zeros(num):  
    num_string = str(num)  
    count = 0  
  
    i = -1  
    while num_string[i] == '0':  
        count += 1  
        i -= 1  
  
    return count
```

روش دوم:

اما بدون کمک string انجامش بدیم. یعنی با عدد. خب چه جوری من صفرها رو بشمرم؟ قبول داریم که صفرها یعنی اینکه بر ۱۰ بخش پذیره؟ پس یعنی من هر بار چک کنم ببینم عدد بر ۱۰ بخش پذیره یا نه و اگر بود، صفر رو می‌ریزم بیرون.  
- چه جور میشه ریخت بیرون؟  
+ با شیفِت دادن به سمت راست. یعنی با تقسیم صحیح بر عدد ۱۰. یعنی رقم سمت راست رو می‌ریزم بیرون.

```
def right_zeros(num):  
    count = 0  
    while num % 10 == 0:  
        count += 1  
        num //= 10  
    return count  
print(right_zeros(factorial(num)))
```

انتخاب یکی از توابع factorial هم که بالا نام بردیم به عهده خودتون. یعنی خروجی فاکتوریل رو میدیم به right\_zeros که برامون تعداد صفرها رو چاپ کنه.

روش سوم:

صفر جلوی عدد از چی به دست میاد؟ از ضرب یه ۲ در یه ۵ درسته؟ پس صرفاً نیاز به بینیم که تعداد پنج‌های درون عددمون چند تاست و بگیریم تعداد صفرهای جلوی عدد هم همونه.

درواقع تعداد ۵ ها همیشه کمتر و یا مساوی تعداد ۲ ها هست. پس نیاز نیست تعداد ۲ رو بشماریم. چون تعداد ۲ به اندازه کافی هست توی فاکتوریل. تعداد ۵ ممکنه کمتر باشه ولی. پس تعداد صفرا همون تعداد ۵ های درون اعدادی که فاکتوریل رو می‌سازنه.

پس از ۱ تا خود عدد پیش میریم. و هی تعداد پنج‌های درون هر عدد رو حساب می‌کنیم:

```
num = int(input())
zero_count = 0
for i in range(num+1):
    divisor = i
    while divisor % 5 == 0 and divisor != 0:
        zero_count += 1
        divisor //= 5
print(zero_count)
```

- به نظرتون چرا یه متغیر جدید به نام divisor تعیین کردم و مستقیم با i کار نکردم؟  
+ چون نیازه هی ببینیم اون عدد ما بر ۵ بخش‌پذیر هست یا نه؟ تا زمانی که ۵ توی خودش داشت، هی یکی به تعداد صفرا اضافه کنیم و اون عدد رو تقسیم بر ۵ کنیم. تا وقتی که عدد صفر نشده.  
اگر یه متغیر تعریف نکنیم و با خود i کار کنیم، مقدار i که اون متغیر for ما هست، تغییر می‌کنه و اشتباه رخ میده. درواقع حلقه یه جورایی بی‌نهایت میشه. چون هر بار داریم i رو به صفر می‌رسونیم. هر بار for انگار از اول شروع شده!

حالا تفاوت سرعتی این روش:

Time1: 21.8 ms

Time2: 33 s

دیدین چقدر تفاوت سرعتی داشت؟ روش سوم ۱۵۰۰ برابر سریع‌تره!

پاسخ ۷:

```
num_string = input()
for i in num_string:
    print(i + ":", int(i) * i)
```

دونه‌دونه روی کرکترهاش حرکت می‌کنم و و اول خود کرکتر کانکت با «:» و بعد cast اش می‌کنم به اینتیجر و ضربش می‌کنم در تعداد اون عدد. یعنی مثلاً ۷ ضربدر کرکتر ۷. که توی string یعنی ۷ بار کرکتر ۷ رو چاپ کن. اینطوری هم میشد انجامش داد:

```
num_string = input()
for i in num_string:
    print(f"{i}:", int(i) * i)
```

پاسخ ۸:

```
def rotate_string(s, rotate_times):
    for i in range(n):
        s = s[1:] + s[0]
    return s
```

```
string = input('Enter a string: ')
rotate_count = int(input('Enter a number: '))
print(rotate_string(string, rotate_count))
```

هر بار string ما برابر اینه که اولش از کرکتر دوم (ایندکس ۱ به بعد) + کرکتر اول بره آخر باشه.

کوییز:

۱- سؤال در لینک زیر:

<https://quera.org/problemset/298/>

پاسفنامه:

پاسخ ا:

```
def factorization(n):
    result = ''
    power = 0
    i = 2
    while n != 1:
        is_a_factor = False
        power = 0
        while n % i == 0:
            n //= i
            power += 1
        is_a_factor = True
        # last one (no need for *) and power is 1 (no need for ^)
        if (is_a_factor) and (n == 1) and (power == 1):
            result += f'{i}'
        # Not last one (need for *) and power is 1 (no need for ^)
        elif (is_a_factor) and (power == 1):
            result += f'{i}*'
        else:
            result += f'{i}^{power}'
```

```

# last one (no need for *) and power is not 1 (need for ^)
elif (is_a_factor) and (n == 1):
    result += f'{i}^{power}'

# Not last one (need for *) and power is not 1 (need for ^)
elif is_a_factor:
    result += f'{i}^{power}*'
    i += 1
return result

n = int(input())
print(factorization(n))

```

چهار حالتی که ممکن بود عدداً بیان رو به ترتیب نوشتیم. `power` اول صفر هست و هر بار که بخش‌پذیره، یکی به توان اضافه میشه.

`is_a_factor` هم نشون میده که آیا اون `i` ما، فاکتوری از عدد هست یا نه؟ اگر نیست، هیچ‌وقت شرطاً اجرا نمیشن و به `i` یکی اضافه میشه تا بره عدد بعدی.

- نیاز نبود مقسوم‌ها رو چک کنیم که اول باشن؟  
 + نه! یکم فکر کنین خودتون دلیلش رو پیدا کنین!

پاسخ: دلیلش اینه که از کوچک‌ترین اعداد شروع میشه و وقتی مثلاً بر ۲ بخش‌پذیر باشه، اونقدر تقسیم بر ۲ میشه که دیگه بر ۲ بخش‌پذیر نباشه. یعنی مثلاً می‌فهمیم بر ۲ با تعداد (`power`) مثلاً ۴ بخش‌پذیره. پس ۴ بار تقسیم بر ۲ صورت گرفته و دیگه بر ۲ بخش‌پذیر نیست اصلاً! دیگه عددی تولید شده که بر ۲ اصلاً بخش‌پذیر نیست!

## • See character as a number! (ASCII (American Standard Code for Information Interchange))

خب خب! می‌دونیم که کامپیوتر فقط صفر و یک بلده! صفر و یک‌ها در کنار هم چی رو می‌سازن؟ یه عدد رو می‌سازن! یعنی کامپیوتر فقط عدد می‌فهمه. کامپیوتر `abc` که نمی‌فهمه! فقط عدد رو می‌فهمه. پس باید عددی باهاش صحبت کنیم!

درواقع همه چیز توی کامپیوتر عددی پردازش میشه. حتی همون `a` که شما نوشتین!

یه استاندارد تعریف شده که برای هر کرکتر یه عدد بدیم. بهش میگن `ASCII`.

توی اینترنت سرچ کنین «`ASCII table`». جدولشو واستون میاره. خلاصه اینکه کرکترهای انگلیسی هرکدوم یه عدد دارن درواقع. مثلاً `a` برابر ۹۷ هست.

یا مثلاً b برابر ۹۸.

A برابر ۶۵.

خلاصه که اول حروف بزرگ هستند، بعد حروف کوچک. حتی عدد کرکتری هم، عدد مخصوص به خودشو داره. حتی علامت سؤال (?) و ...

پایتون برای تبدیل این چیزا به هم خودش توابعی داره. توابع ord و chr. مثلاً با دادن هر کرکتر به تابع ord، بهتون عددش رو return می‌کنه. مثلاً:

```
print(ord('A'))
```

یا مثلاً:

```
print(chr(65))
```

یعنی بخوایم تو کد کرکتر رو به حرف و حرف رو به کرکتر تبدیل کنیم، می‌تونیم از این استفاده کنیم. نکته! برنامه‌نویس خوب حواسش هست که یه وقت عددی خارج از رنج ASCII به «chr» نده! چون کرکترهای عجیب غریب چاپ میشن. پس اگر یه وقت کرکترهای عجیب غریب چاپ شدن، یه نگاه بندازین شاید متغیری که پاس دادین، عددی خارج رنجش داد.

حالا این به چه درد ما می‌خوره؟

فرض کنیم من یه string دارم و می‌خوام با استفاده از یه تابع، حروف بزرگشو تبدیل به حروف کوچک کنم. سعی کنیم خودتون اول بنویسینش و بعد نگاه پاسخ کنیم. راهنمایی ۱: فاصله بین حروف بزرگ و کوچک چقدر بود؟ مثلاً a تا A، فاصلشون ۳۲ بود. پس من برای تبدیل A به a باید مقدارشو بعلاوه ۳۲ کنم. یعنی اول برای هر کرکتر بزرگ مقدار عددیشون پیدا کنم. بعلاوه ۳۲ کنم. حالا مقدار عددی تبدیل به مقدار عددی حروف کوچک شد. حالا باید کرکتریش کنم.

پاسخ:

```
def tolower(string):  
    for i in range(len(string)):  
        if string[i] >= "A" and string[i] <= "Z":  
            string[i] = chr(ord(string[i]) + 32)  
    return string
```

یه string می‌گیره. دونه دونه روی index هاش حرکت می‌کنم و نگاه می‌کنم ببینم کدوم ایندکس در محدوده بین A تا Z هستن (طبق جدول ASCII، اینا حروف بزرگن). حالا میاد اول تبدیل به عددش می‌کنه (ord) و بعدش مقدار عددی رو بعلاوه ۳۲ می‌کنه و بعد کل مقدار عددی رو با chr تبدیل به کرکتر می‌کنه و می‌گذاره جای خودش.

اگر هم حروف بزرگ نبودن (حروف کوچک بودن یا هر کرکتر دیگه‌ای مثل فاصله) که هیچی! نمی‌خواد کاری کنه!  
در پایان هم string رو return می‌کنه.

خب وایسین ببینم! این کد آیا مشکلی نداره؟!  
می‌تونین تستش کنین:

```
input_string = input("Enter a string: ")  
print(tolower(input_string))
```

یا چون تسته، نمی‌خواد مقدار از کاربر بگیرین! خودتون توی کد می‌تونین بهش مقدار بدین:

```
print(tolower("AaZz bBHh"))
```

مقدار رو هوشمندانه دادم. یعنی هم اولین و آخرین کرکتر (اون edge case ها) و هم یه مقدار وسط توش باشه. یه فاصله هم گذاشتم توی کرکترها که ببینم درست کار می‌کنه؟ (چون فاصله یه کرکتر حرفی نیست و همونطور که یادمونه توی ASCII Table، کرکترای علامت سؤال و فاصله و... هم بودن. می‌خوام ببینم برای اونا هم درست کار می‌کنه؟ یه وقت کاربر کرکتر علامت تعجب رو وارد کرد. برای اونم اوکیه؟!)  
عه ارور داد! نوشت:

```
TypeError: 'str' object does not support item assignment
```

چرا به نظرتون؟

یادتونه گفتیم string ها تغییر ناپذیرن؟ یعنی immutable هستن. پس همیشه یه کرکتر داخلشون رو عوض کرد! صرفاً میشه یه استرینگ رو گذاشت جای استرینگ قبلی (مقدار متغیر رو عوض کرد) و اینطوری کل مقدار قبلی پاک میشه.  
پس باید چیکار کنیم؟

توی یه متغیر جدا، مقدار string با حروف کوچک رو بسازیم:

```
def tolower(string):  
    lower_string = ""  
    for char in string:  
        if char >= "A" and char <= "Z":  
            lower_string += chr(ord(char) + 32)  
        else:  
            lower_string += char  
  
    return lower_string
```

یه متغیر جدا و خالی به نام lower\_string تعیین کردم و داخلش مقدار رو ساختم. هر بار کرکتر جدید رو به آخر lower\_string اضافه کردم. (concatenate کردم).

## Vigenère cipher (16<sup>th</sup> century)

ما به پیام داریم و به کلید. کلید رو میایم اونقدر تکرار می‌کنیم که هم‌طول پیام بشه. (مثلاً کلید ما اینجا، KEY هست. بعدش میایم پیام رو با کلید جمع می‌کنیم و متن رمز شده رو می‌سازیم. پیام رو با کلید جمع می‌کنیم یعنی چی؟ بیایم روی جدول توضیحش بدیم:

Text	H	E	L	L	O	H	O	W	A	R	E	Y	O	U	A	R	E	Y	O	U	O	K
Key	K	E	Y	K	E	Y	K	E	Y	K	E	Y	K	E	Y	K	E	Y	K	E	Y	K
Encrypted (Text + Key)	R	I	J	V	S	F	Y	A	Y	B	I	W	Y	Y	Y	B	I	W	Y	Y	M	U

یعنی مثلاً H که ۷ حرف جلوتر از A هست (چون هشتیمن حرف الفباست) رو با K جمع می‌کنیم. یعنی از K، تعداد ۷ تا بار میریم که می‌رسیم به R. خب رمز ما ساخته شد. اگر از Z جلو زدیم، میریم اول حروف الفبا؛ یعنی به دونه از Z جلو زدیم، حرف A قرار میگیره. دوتا B سه تا C و همینطور تکرار میشه. خب کار شما اینه که اول به متن از کاربر می‌گیرین. بعد به کلید می‌گیرین. بعدش رمز رو چاپ می‌کنین. توجه کنین که متن می‌تونه شامل حروف غیر کرکتری هم باشه! اما حروف کرکتریش، همش کرکتر حروف غیر کرکتری مثل «؟!» نیاز به رمز شدن نداره.

برای سادگی کار، در ابتدا فرض کنین که طول کلید هم اندازه طول متن هست. و نیاز به گسترش توسط شما نیست! و حالا بعداً حالتی که کلید کوچکتر از متن هست رو در نظر بگیرین!

راهنمایی:

خب باید دونه‌دونه رو کرکتر پیش بریم و کرکتر کلیدی که هست رو باهاش جمع کنیم (مقدار عددیشونو جمع کنیم).

باید حواسمون باشه که از Z نزنه جلو. اگر زد، باید برش گردونیم به عقب. عدد اسکی «a» مقدارش ۹۷ هست. خب کار با عدد اسکی که از ۹۷ (a) شروع میشه تا ۱۲۲ (Z) میره ساده‌تره یا اینکه من پیام بگم a مقدارش ۰ هست. b مقدارش a و Z مقدارش ۲۵. حالا اگر از ۲۵ جلو زد، پیام از صفر پیش برم؟ قاعدتاً حالت دوم ساده‌تره برای نوشتن. اینطوری خیلی بهتر میشه نوشت. پس سعی می‌کنم اول مقدار عددیشون رو حساب کنم و بعد پیام منهای ۹۷ کنم که از صفر شروع شن:

$$a = 97 \rightarrow 97 - 97 = 0$$

$$b = 98 \rightarrow 98 - 97 = 1$$

$c = 99 \rightarrow 99 - 97 = 2$

...

$z = 122 \rightarrow 122 - 97 = 25$

پس حواسم هست بهش.  
تست کیس:

input:

text: **a**

key: **a**

output:

**a**

-----

input:

text: **b**

key: **b**

output:

**c**

-----

input:

text: **hello**

key: **key**

output:

**rijvs**

-----

input:

text: **z**

key: **z**

output:

**y**

تست کیس بیشتر می‌خوانیم؟



من یه سری تست کیس حساس مثل a و a و Z و Z رو بهتون دادم که مطمئن شین کدتون این نقاط حساس رو هم پوشش میده. اگر باز می‌خوان، خودتون با وبسایت زیر تولید کنین:

<https://vigenerecipher.com/>

پاسخ:

خب تابعی می‌سازم که ازم یه متن می‌گیره و یه کلید. فعلاً هم برای سادگی فرض می‌کنم که طول کلید هم اندازه طول متنه. بعدش باید روی تک‌تک کرکتر حرکت کنم و با کلید جمع کنم. یعنی ایندکس‌های متناظر رو با هم جمع می‌کنم. اگر هم کرکتری نبود، باید همونطوری ولش کنم و کاریش نکنم.

من میام رمز رو توی یه متغیر به نام encrypted\_text می‌گذارم:

```
def vigenere_encrypt(plain_text, key):
    encrypted_text = ''
    for i in range(len(plain_text)):
        if is_alpha(plain_text[i]):

            else:
                encrypted_char = plain_text[i]

    encrypted_text += encrypted_char
```

می‌گم اگر اون کرکتر من (یعنی همون plain\_text[i]) به صورت کرکتری بود، یه کار انجام بده. اگر نبود، همون توی رمز قرار می‌گیره. بلاک else می‌گه که خود کرکتر رو بریز توی encrypted\_char. و در نهایت encrypted\_char مرحله به مرحله به آخر encrypted\_text اضافه میشه.

خب دیدین؟ لزوماً قرار نیست همون اول if رو کامل کنم. فعلاً else رو نوشتم. قسمت بلاک if رو بعداً کامل می‌کنم.

تازه یه تابعی رو نوشتم که هنوز تعریفش نکردم. یعنی دیدم عه نیازه به یه چیز برای فهمیدن اینکه کرکتر حروفی هست یا نه دارم. خب فعلاً اسمشو می‌گذارم is\_alpha و if ام رو تکمیل می‌کنم و بعد تکمیل if، تابع is\_alpha رو می‌نویسم.

این مواقع می‌تونین از کلمه pass استفاده کنین که به خاطر عدم تکمیل کد، کدایی پایینی به ارور نخورن. کلمه pass هیچ کاری نمی‌کنه. صرفاً می‌گه عبور کن از این خط:

```
def vigenere_encrypt(plain_text, key):
    encrypted_text = ''
    for i in range(len(plain_text)):
        if is_alpha(plain_text[i]):
            pass
        else:
            encrypted_char = plain_text[i]
```

```
encrypted_text += encrypted_char
```

خب میایم قسمت بلاک if رو بنویسیم:

```
def vigenere_encrypt(plain_text, key):
    encrypted_text = ''
    for i in range(len(plain_text)):
        if is_alpha(plain_text[i]):
            encrypted_char = (ord(plain_text[i]) - ord('a')) +
(ord(key[i]) - ord('a'))
        else:
            encrypted_char = plain_text[i]
```

```
encrypted_text += encrypted_char
```

قسمت بلاک if خیلی طولانی شد. این تمیز نیست! اصطلاحاً کدتون نباید به صورت افقی scroll بشه. یا horizontal scrolling نباید داشته باشه. اینطوری تمیزتره که هی نخوایم اسکرول کنیم! پس اینطوری می‌نویسمش:

```
def vigenere_encrypt(plain_text, key):
    encrypted_text = ''
    for i in range(len(plain_text)):
        if is_alpha(plain_text[i]):
            encrypted_char = (
ord(plain_text[i]) - ord('a')) + (ord(key[i]) -
ord('a'))
        else:
            encrypted_char = plain_text[i]
```

```
encrypted_text += encrypted_char
```

درواقع می‌گیم که بیا ord برای کرکتر plain\_text ما حساب کن. منهای ord کرکتر a کن که انگار کرکتر از ۰ شماره‌گذاری شده. بعدش بعلاوه ایندکس متناظرش در key کن. اونم منهای a کن که اونم از صفر انگار نامگذاری شه. پرانتزها رو هم براتون رنگی کردم که بهتر درکش کنید  $\text{ord(plain\_text[i])} - \text{ord('a')} + \text{ord(key[i])} - \text{ord('a')}$  در آخر هم chr رو حساب کردم که یعنی تبدیل به کرکترش کنه.

خب حالا چی نیازه؟

اینکه چک کنیم آیا از ۲۶ زده بیرون یا نه؟ (شماره‌گذاریمون از ۰ تا ۲۵ بود) اگر زده بیایم از اول. این کار رو می‌تونیم با منهای ۲۶ انجام بدیم. می‌تونیم هم از باقی‌مونده «%» کمک بگیریم. یعنی باقی‌موندش به ۲۶ بگیریم. بعدش هم باید مقدار عددی که از ۰ تا ۲۵ هست رو بعلاوه مقدار عددی a کنیم که بعدش بتونیم با chr کرکترش رو بسازیم:

```
def vigenere_encrypt(plain_text, key):
    encrypted_text = ''
    for i in range(len(plain_text)):
        if is_alpha(plain_text[i]):
            encrypted_char = (
                ord(plain_text[i]) - ord('a')) + (ord(key[i]) -
ord('a'))
            if encrypted_char >= 26:
                encrypted_char = encrypted_char % 26
            encrypted_char = chr(encrypted_char + ord('a'))
        else:
            encrypted_char = plain_text[i]

        encrypted_text += encrypted_char

    return encrypted_text
```

در آخر هم مقدار encrypt شده رو return کردیم.

خب خیلی هم عالی! حالا که اینو ساختیم، بیایم حالتی رو بسازیم که کلید کوچکتر از متنه. یعنی فرض کنیم طول کلید ۳ بود. پس:

```
abc   defg hi
qwe   qwe  qw
```

یعنی درواقع ما باید یه ارتباطی پیدا کنیم که هر ایندکس، کدوم ایندکس کلید رو می‌خواد؟

Text index	Key index
0	0
1	1
2	2
3	0
4	1
5	2
6	0

یعنی درواقع ارتباط اینه:

```
plain_text[i] → key[i % len]
```

یعنی ایندکس i ام، باید با ایندکس  $i \% \text{len}$  تناظر پیدا کنه. اینطوری i همیشه بین ۰ تا ۲ (یعنی همون رنج ایندکس‌های key قرار می‌گیره. پس کد رو درست می‌کنیم:

```
def vigenere_encrypt(plain_text, key):
    encrypted_text = ''
    for i in range(len(plain_text)):
        if is_alpha(plain_text[i]):
            encrypted_char = (
                ord(plain_text[i]) - ord('a')) + (ord(key[i %
len(key)]) - ord('a'))
            if encrypted_char >= 26:
                encrypted_char = encrypted_char % 26
            encrypted_char = chr(encrypted_char + ord('a'))
        else:
            encrypted_char = plain_text[i]

        encrypted_text += encrypted_char

    return encrypted_text
```

خب حالا کد is\_alpha رو بنویسیم:

حواستون باشه که تابع is\_alpha باید قبل از تابع vigenere\_encrypt باشه. چون توی اینجا داره استفاده میشه. پس قبلش باید تعریف شده باشه.

```
def is_alpha(char):
    if 'a' <= char <= 'z':
        return True
    return False
```

اگر کرکتر بین a تا z بود (حروف کوچک) ریترن کن True رو. البته اینطورم می‌تونستیم بنویسیمش:

```
def is_alpha(char):
    return 'a' <= char <= 'z'
```

البته بهتر بود اسم تابع رو می‌گذاشتیم is\_lower به معنای اینکه «آیا حروف کوچیکه؟» چون اینطوری بهتر بود. این اسم is\_alpha یکم غلط اندازه که آدم فکر می‌کنه حروف بزرگ هم بدیم جواب درست می‌ده. ولی خب چون سوالمون صرفاً حروف کوچیک داشت، اوکیه.

خب حالا کد رو تکمیل کنیم:

```
def is_alpha(char):
    return 'a' <= char <= 'z'
```

```

def vigenere_encrypt(plain_text, key):
    encrypted_text = ''
    for i in range(len(plain_text)):
        if is_alpha(plain_text[i]):
            encrypted_char = (
                ord(plain_text[i]) - ord('a')) + (ord(key[i %
len(key)]) - ord('a'))
            if encrypted_char >= 26:
                encrypted_char = encrypted_char % 26
            encrypted_char = chr(encrypted_char + ord('a'))
        else:
            encrypted_char = plain_text[i]

    encrypted_text += encrypted_char

    return encrypted_text

plain_text = input("Enter the text: ")
key = input("Enter the key: ")
print(vigenere_encrypt(plain_text, key))

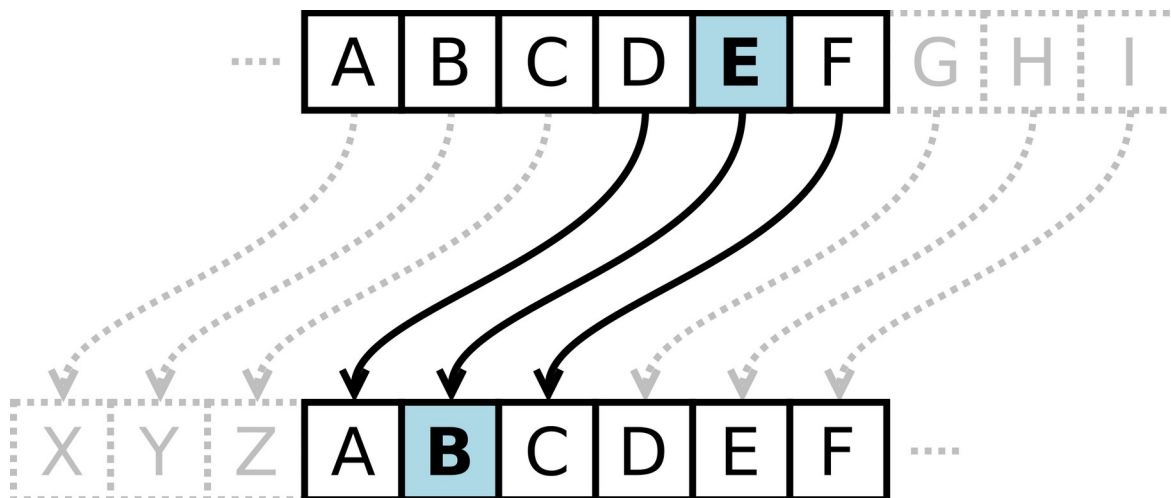
```

حالا به من بگین که برنامه‌نویس خوب حواسش به چی هست؟  
 + حواسش به این هست که طول کلید رو بزرگ‌تر از طول متن ندن! یه if ساده می‌تونه شرط رو چک کنه. از همین الان سعی کنین ذهنتونو درست پرورش بدین. می‌دونم سخته هی بخواین به حالتی که کاربر چیز غلط میده رو فکر کنین و یا هی فکر کنین که چه چیزایی ممکنه اشتباه شه ولی اینو بگم که ذهن شما الان مثل یه خمیره. اگر درست شکل بگیره، بعداً هم برنامه‌نویس خوبی میشین ولی اگر بد شکل بگیره و از همین الان که کدا ساده هستن نتونین فکر کنین که حالتای حساس و اینا چه‌جوری ممکنه رخ بدن، ذهنتون بد شکل می‌گیره و بعداً مشکل خواهید داشت. از همین الان این مهارت رو تمرین کنین!

تمرین!

### تمرین ۱:

رمز سزار اینطوری بوده که هر حرف با سه حرف قبلش جابه‌جا میشده. یعنی:



اینطوری پیام رمز می‌شده. مثلاً عبارت:

abcd

تبدیل به عبارت زیر می‌شده:

xyza

خب من می‌خوام شما به پیام بگیرین و رمزش کنین. اما یکم سخت‌تر. تعداد شیفت‌دادن‌ها ۳ تا نیست! بلکه چیزیه که کاربر می‌گه. یعنی می‌تونه مثلاً بگه ۱۰۰ بار شیفت بده! اگر گفت مثلاً ۳- شیفت بده، یعنی به جای سمت چپ، باید به سمت راست shift بدین!

۹۹۹۹:

خط دوم: یه متن که می‌تونه شامل کرکتر کوچیک، بزرگ و یا کرکترهای غیر حروف الفبا مثل فاصله و علامت سؤال و... باشه

خط سوم: تعداد شیفت که می‌تونه هر عدد صحیحی باشه!  
توجه! کرکترهای غیرحروف الفبایی نیاز به رمزشدن ندارن!

تست کیس:

input:

If he had anything confidential to say, he wrote it in cipher, that is, by so changing the order of the letters of the alphabet, that not a word could be made out.

3

output:

Jg if ibe bozuijoh dpogjefoujbm up tbz, if xspuf ju jo djqifs, uibu jt, cz tp dibohjoh uif psefs pg uif mfuufst pg uif bmqibcfu, uibu opu b xpse dpvme cf nbef pvu.

-----

input:

hello, how are you? Qwertyuiopasdfghjklzxcvbnm

3

output:

ebiil, elt xob vlr? ntboqvrflmxpacdeghiwuzsykj

## تمرین ۲:

ما اسم متغیرهامون رو می‌تونیم به صورت‌های مختلف بنویسیم.

Snake Case: `is_prime`

Pascal Case: `IsPrime`

در `snake_case` کلمات با یه «\_» جدا میشن.

در `Pascal Case`، اول هر کلمه حرف بزرگه.

کار شما چیه؟

یه متن میدن بهتون. اگر `PascalCase` بود، تبدیل به `snake_case` کنین.

اگر `snake_case` بود، تبدیل به `PascalCase` کنین.

اگر هیچکدوم نبود، چاپ کنین «Invalid Input».

input:

HelloWorld

output:

hello\_world

-----

input:

is\_prime

output:

IsPrime

-----

input:  
isPrime  
output:  
Invalid Input

راهنمایی ۱:  
توابع زیر رو بسازین:

snake\_to\_pascal  
pascal\_to\_snake

is\_snake  
is\_pascal

دوتای آخری برای اینه که بدونین این متنی که دادیم معتبره یا باید invalid\_input چاپ کنین؟  
حواستون به حالتای خاص باشه.

راهنمایی ۲:

حالتای خاصی که نه snake\_case و نه PascalCase هستن شامل:

\_is\_prime → به دلیل کرکتر اول که «\_» هست

is\_Prime → چون کرکتر بزرگ داریم

is\_prime\_ → به دلیل کرکتر آخر که «\_» هست

helloWorld → حرف اول بزرگ نیست

### تمرین ۳:

یه متن بهتون و یه کلمه بهتون میدن. تمام تکرارهای اون کلمه توی اون متن، به جز اولی رو حذف کنین.

یعنی:

input:  
Hello my name is john. John is my first name. My father's name is also john.

john

output:

Hello my name is john. John is my first name. My father's name is also .

توجه کنین که پایتون حروف بزرگ و کوچیک براش مهمه. یعنی case sensitive هست. پس John

با john متفاوته!



پاسفنامه:

پاسخ ا:

خب باید دونه دونه حرکت کنیم و شیفت بدیم. یعنی درواقع مقدار عددیشو منهای shift کنیم. مثل سؤال Vigenère cipher، برای سادگی کار، باید هر کرکتر رو منهای کرکتر ابتدایی کنیم (اگر کوچیک بود منها «a» و اگر بزرگ بود منهای «A») که ساده تر شه و بهتر بتونیم باهاش کار کنیم. کرکترهای غیر حروف الفبایی هم نیازی به رمزشدن ندارن پس عیناً چاپ میشن. پس تا اینجا کد رو می نویسیم:

```
1. def caesar_encrypt(plaintext, shift):
2.     ciphertext = ""
3.     for char in plaintext:
4.         if is_alpha(char):
5.             if is_lower(char):
6.
7.                 else: # upper case
8.
9.                 else:
10.                    ciphertext += char
11.
12.
13. return ciphertext
```

خط ۳: روی کرکترها دونه دونه حرکت می کنیم. اگر کرکترم جزء حروف الفبا بود، باید رمزش کنم. در غیر این صورت (خط ۹)، میام به ciphertext، کرکتمو اضافه می کنم.

حالا قسمت اگر حروف الفبا بود رو می کنیم:

اگر is\_lower (حروف کوچیک) یه سری کارها انجام بده؛ اگر نه (حروف بزرگ بود) یه کار دیگه.

چرا حروف بزرگ و کوچک رو جدا کردم؟ چون در صورت بزرگ بودن باید منهای «A» کنم و در صورت کوچک بودن، منهای «a».

دیدین؟ بازم قرار نبود در همون لحظه اول همه چیز رو کامل کنیم. بلکه قدم به قدم مراحل رو پیش میریم و جزئیات رو کامل می‌کنیم! حالا بریم در صورت حروف کوچک بودن، باید منهای «a» کنیم که انگار کرکتا از صفر شماره‌گذاری شن. بعدش منهای عدد shift می‌کنیم. چون سزار به سمت چپ شیفت میداد:

```
def caesar_encrypt(plaintext, shift):
    ciphertext = ""
    for char in plaintext:
        if is_alpha(char):
            if is_lower(char):
                cipher_char = ord(char) - ord('a') - shift
                while cipher_char < 0:
                    cipher_char += 26
                while cipher_char > 26:
                    cipher_char -= 26
                cipher_char = chr(cipher_char + ord('a'))
            else: # upper case

            else:
                ciphertext += char

    return ciphertext
```

بعدش چک می‌کنم اگر منفی شد بعلاوه ۲۶ و اگر بیشتر از ۲۶ بود، منهای ۲۶ می‌کنم. اینطوری در رنج کرکترا قرار می‌گیره و وقتی تبدیل به ASCII می‌کنم، کرکترا قابل چاپ و درست چاپ میشه و کرکترای غیرقابل چاپ ASCII چاپ نمیشه.

در آخر هم به ASCII تبدیلیش می‌کنم. یعنی چون مقدار عددی بین ۰ تا ۲۵ بود، باید بعلاوه «a» کنم که بیاد توی رنج عددی اسکی و بعدش با chr تبدیل به کرکترا ASCII می‌کنم. حالا قسمت upper و کد توابع مورد نیازم رو هم کامل می‌کنم:

```
def is_alpha(char):
    return ('a' <= char <= 'z') or ('A' <= char <= 'Z')

def is_lower(char):
    return 'a' <= char <= 'z'
```

```

def caesar_encrypt(plaintext, shift):
    ciphertext = ""
    for char in plaintext:
        if is_alpha(char):
            if is_lower(char):
                cipher_char = ord(char) - ord('a') - shift
                while cipher_char < 0:
                    cipher_char += 26

                cipher_char = chr(cipher_char + ord('a'))

            else:
                cipher_char = ord(char) - ord('A') - shift
                while cipher_char < 0:
                    cipher_char += 26

                cipher_char = chr(cipher_char + ord('A'))
            ciphertext += cipher_char
        else:
            ciphertext += char

    return ciphertext

input_text = input("Enter your text: ")
shift = int(input("Shift value: "))
print(caesar_encrypt(input_text, shift))

```

پاسخ ۲:

برای تبدیل pascal به snake باید هر جا حروف بزرگ دیدیم (جز حرف اول ← ایندکس صفر)، حرف بزرگ رو کوچک کنیم و قبلش یه کرکتر «\_» بذاریم.

```

def is_upper(char):
    return 'A' <= char <= 'Z'

def tolower(char):
    return chr(ord(char) + 32)

def pascal_to_snake(text):

```

```

snake_text = ""
for i in range(len(text)):
    if is_upper(text[i]):
        if i != 0:
            snake_text += "_"
        snake_text += tolower(text[i])
    else:
        snake_text += text[i]
return snake_text

```

همونطور که دیدین، اگر ایندکس صفر نبود، بیاد اول یه کرکتر «\_» اضافه کنه و بعد کرکتر.

برای تبدیل snake به pascal، می‌تونیم هرجا «\_» دیدیم، ایگنورش کنیم و حرف بعدیشو بزرگ کنیم.

```

def toupper(char):
    return chr(ord(char) - 32)

def snake_to_pascal(snake):
    pascal_text = ""
    pascal_text += toupper(snake[0]) # first letter

    i = 1
    while i < len(snake):
        if snake[i] == "_":
            i += 1
        else:
            pascal_text += toupper(snake[i])
        pascal_text += snake[i]
        i += 1
    return pascal_text

```

اول کرکتر اول رو بزرگ کردم و گذاشتم توش. بعدش روی کرکتر حرکت کردم. اگر برابر «\_» بود، میرم کرکتر بعدی رو بزرگ می‌کنم و بعد می‌گذارم توی pascal\_case. اگر نه خب عادیشو می‌گذارم. در هر صورت ایندکس هم باید برم جلو به خاطر while که روی کرکتر بتونم حرکت کنم.

اینجاش یه مشکلی ممکنه باشه. به نظرتون چیه؟ همون قسمت بزرگ کردن اولین کرکتر ممکنه مشکل به وجود بیاره. به نظرتون چه مشکلی؟

+ برنامه‌نویس خوب کسیه که حواسش باشه که ورودی رو اشتباه ندن. مثلاً ممکنه string خالی پاس داده شه. خب `snake[0]` معنایی نداره! چون کرکتری نداره! پس می‌تونیم یه `if` بگذاریم و بگیم اگر `len` بزرگ‌تر از ۰ بود (یعنی حداقل یک کرکتر داشت)، این کار رو انجام بده:

```
1. def snake_to_pascal(snake):
2.     pascal_text = ""
3.     if len(snake) > 0:
4.         pascal_text += toupper(snake[0]) # first letter
5.
6.         i = 1
7.         while i < len(snake):
8.             if snake[i] == "_":
9.                 i += 1
10.            pascal_text += toupper(snake[i])
11.        else:
12.            pascal_text += snake[i]
13.            i += 1
14.        return pascal_text
```

خب باز می‌تونیم کد رو بهینه‌تر کنیم. به نظرتون کجا داریم اضافه کاری انجام میدیم؟  
+ خط ۳ و خط ۷ دوبار داریم `len` رو حساب می‌کنیم. آیا بهتر نیست `len` رو بریزیم داخل یه متغیر و با اون کار کنیم؟ یعنی یه بار `len` رو حساب کنیم و مقدارشو یه جا نگه داریم. اینطوری سرعت برنامه زیاد میشه، چون نیاز نیست دو بار `len` حساب شه!

```
def snake_to_pascal(snake):
    pascal_text = ""
    snake_length = len(snake)
    if snake_length > 0:
        pascal_text += toupper(snake[0]) # first letter

    i = 1
    while i < snake_length:
```

```

        if snake[i] == "_":
            i += 1
            pascal_text += toupper(snake[i])
        else:
            pascal_text += snake[i]
            i += 1
    return pascal_text

```

خب حالا توابعی رو می‌نویسیم که تشخیص بده is\_snake یا is\_pascal هست یا نه. توابع رو جدا می‌نویسیم:

```

1. def is_snake(snake):
2.     i = 0
3.     while i < len(snake):
4.         if (not is_lower(snake[i])) and (not snake[i] ==
           "_"):
5.             return False
6.         if snake[i] == "_":
7.             if snake[i + 1] == "_":
8.                 return False
9.             i += 1
10.    return True

```

همیشه بدونین که موقعی که می‌خوایم عدد ایندکس (i) رو دستکاری کنیم، با while باید پیش بریم. برای این مواقع for همیشه به کار برد. خط ۴ و ۵: می‌گیم که اگر دونه‌ای از کرکتا هم پیدا شد که یا کرکت کوچک نبود یا کرکت «\_» نبود، return کن False کن. حالا برو ببین هر جا که «\_» رو دیدی، ببین کرکت بعدیش آیا بازم «\_» هست یا نه؟ اگر بود ریترن False کن.

این قسمت رو اینطور می‌تونستیم بنویسم که چک کن که اگر lower نبود، ریترن کن False. اما خب فرقی نداره. چون به خاطر خط ۴، صرفاً زمانی وارد این خط میشه که کرکت کوچک یا «\_» باشه. خلاصه هدف این کار اینه که من دوتا کرکت «\_» پشت هم رو به عنوان snake\_case قبول نکنم.

خب اما اینجا یه مشکلی هست! همونطور که توی راهنمایی ۲ اشاره کردم، باید حالت‌های خاص رو در نظر بگیرم. یعنی اینجا چک همیشه که کرکتر اول نباید «\_» باشه. پس شرط می‌گذارم که چکش کنم. همچنین حواسم هست که string خالی بهم پاس ندن! پس قبلش شرط اینکه len بزرگ‌تر از صفر باشه رو چک می‌کنم:

```
1. def is_snake(snake):
2.     if len(snake) == 0: # avoid empty string
3.         return False
4.     if snake[0] == '_': # first letter must not be '_'
5.         return False
6.
7.     i = 0
8.     while i < len(snake):
9.         if (not is_lower(snake[i])) and (not snake[i] ==
            "_"):
10.            return False
11.        if snake[i] == "_":
12.            if snake[i + 1] == "_":
13.                return False
14.            i += 1
15.        return True
```

خب به نظرتون کجا مشکل index out of range به وجود میاد؟  
خط ۱۲. چون while ما داره تا ایندکس آخر پیش میره. حالا ایندکس آخر + ۱ از رنج ایندکس‌ها می‌زنه بیرون! پس یه if می‌گذارم که چک کنه که ببینه که اگر ایندکس آخر بود، چون توی بلاک if خط ۱۱ هست (یعنی کرکتر آخر «\_» بوده)، return کنه False.

```
def is_snake(snake):
    if len(snake) == 0: # avoid empty string
        return False
    if snake[0] == '_': # first letter must not be '_'
```

```

    return False

    i = 0
    while i < len(snake):
        if (not is_lower(snake[i])) and (not snake[i] == "_"):
            return False
        if snake[i] == "_":
            if i == len(snake) - 1: # last letter must not be '_'
                return False
            if snake[i + 1] == "_": # two '_' must not be next to each
other
                return False
        i += 1
    return True

```

خب تکمیل شد!

حالا is\_pascal رو هم می‌نویسیم:

باید چک کنیم فقط حروف بزرگ و کوچک باشن. بعدش چک کنیم که هیچ دو حرف بزرگ کنار هم قرار نگیرن! (البته این یه مشکلی داره! مثلاً:

CountA

ItIsAFunction

رو اشتباه تشخیص میده و می‌گه Pacal نیست! ولی فعلاً همین اوکیه و با روشی که می‌گم پیش برین!

```

def is_pascal(pascal):
    if len(pascal) == 0: # avoid empty string
        return False
    if not is_upper(pascal[0]): # first letter must be upper
        return False

    i = 1
    while i < len(pascal):
        if (not is_lower(pascal[i])) and (not is_upper(pascal[i])):
            return False

        if is_upper(pascal[i]):
            if i == len(pascal) - 1:
                return False
            if is_upper(pascal[i + 1]):
                return False

```



```
i += 1  
return True
```

if اول درون while چک می‌کنه جز بزرگ و کوچیک، حروف دیگه‌ای مثل «!؟» نباشن.  
if دوم درون while چک می‌کنه که اگر حروف بزرگ بود، بعدیش بزرگ نباشه. (و چک می‌کنه که حرف بزرگ، آخرین حرف نباشه و out of range رخ نده)

خب کدامون رو کنار هم می‌چینیم:

```
def is_upper(char):  
    return 'A' <= char <= 'Z'
```

```
def is_lower(char):  
    return 'a' <= char <= 'z'
```

```
def tolower(char):  
    return chr(ord(char) + 32)
```

```
def toupper(char):  
    return chr(ord(char) - 32)
```

```
def pascal_to_snake(pascal):  
    snake_text = ""  
    for i in range(len(pascal)):  
        if is_upper(pascal[i]):  
            if i != 0:  
                snake_text += "_"  
            snake_text += tolower(pascal[i])  
        else:  
            snake_text += pascal[i]  
    return snake_text
```

```
def snake_to_pascal(snake):  
    pascal_text = ""
```

```

snake_length = len(snake)
if snake_length > 0:
    pascal_text += toupper(snake[0]) # first letter

    i = 1
    while i < snake_length:
        if snake[i] == "_":
            i += 1
            pascal_text += toupper(snake[i])
        else:
            pascal_text += snake[i]
            i += 1
    return pascal_text

def is_snake(snake):
    if len(snake) == 0: # avoid empty string
        return False
    if snake[0] == '_': # first letter must not be '_'
        return False

    i = 0
    while i < len(snake):
        if (not is_lower(snake[i])) and (not snake[i] == "_"):
            return False
        if snake[i] == "_":
            if i == len(snake) - 1: # last letter must not be '_'
                return False
            if snake[i + 1] == "_": # two '_' must not be next to each
other
                return False
            i += 1
    return True

def is_pascal(pascal):
    if len(pascal) == 0: # avoid empty string
        return False
    if not is_upper(pascal[0]): # first letter must be upper
        return False

```

```

i = 1
while i < len(pascal):
    if (not is_lower(pascal[i])) and (not is_upper(pascal[i])):
        return False

    if is_upper(pascal[i]):
        if i == len(pascal) - 1:
            return False
        if is_upper(pascal[i + 1]):
            return False

    i += 1
return True

input_text = input("Enter your text: ")
if is_snake(input_text):
    print(snake_to_pascal(input_text))
elif is_pascal(input_text):
    print(pascal_to_snake(input_text))
else:
    print("Invalid Input")

```

آخر هم يه ورودی می گیرم و اگر snake\_case بود، تبدیلشو به pascal پرینت کنه.  
 اگر نه، چک کنه ببینه اگر پاسکال بود، تبدیلشو به snake چاپ کنه.  
 در غیر این صورت، ورودی معتبر نیست و چاپ کنه «Invalid Input».

### پاسخ ۳:

خب من باید هی اندازه طول تکرارم، زیر رشته از متن اصلی جدا کنم که ببینم توش هست یا نه؟ یه flag به نام is\_first هم می گذارم که اولین تکرار رو حذف نکنم.  
 اینطوری که وقتی پیدا کرد، چک می کنه که is\_first مقدارش False باشه. اگر باشه، یعنی اولین تکرار نیست؛ پس تکرار رو حذف می کنه. اما اگر باشه، یعنی بار اوله و نباید تکرار رو حذف کنه. ولی باید بپره بره is\_first رو False کنه که برای دفعه های بعد مشخص باشه که اولی پیدا شده و حق داریم تکرارهای بعدی رو حذف کنیم:

```

def delete_duplication(text, duplicate):
    dup_len = len(duplicate)
    is_first = True

```

```

for i in range(len(text) - dup_len + 1):
    if text[i:i+dup_len] == duplicate:
        if not is_first:

else:
    is_first = False
return text

```

```

text_input = input("Enter your text: ")
duplicate_input = input("Enter your duplicate: ")
print(delete_duplication(text_input, duplicate_input))

```

رنج for رو همون اول نیاز نبود تعیین کنیم. بلکه فکر می کردیم که آخرین مقایسه برای مثلاً یه عنصر ۴ عضوی اینه:

johnjohn

یعنی آخرین i ما،  $\text{len}(\text{text}) - \text{len}(\text{duplicate})$  هست. (اگر نگاه کنین ۸ کرکتره. یعنی ۷ ایندکس. ما آخرین رو می‌خوایم از ایندکس ۴ تا ۷ مقایسه کنیم. پس درواقع می‌تونیم بگیریم آخرین i برابر ۴ هست.)

حالا چون for تا قبل اون عدد میره، یکی بیشتر از اون. میشه.

- به نظرتون چرا من توی شرط for مقدار len متنم رو حساب می‌کنم؟ برنامه‌کننده کمتر نمیشه هی حسابش بخوام هر بار موقع شرط چک کردن for، حسابش کنم؟

+ چون دارم توی بلاک for هی مقدار text رو تغییر میدم. پس i آخری مقدارش متفاوته. اگر بیرون حلقه حساب می‌کردم، مثلاً میشد ۲۰. اما من توی بلاک for سایز text رو تغییر میدم و کاهش پیدا می‌کنه. پس دیگه حداکثر ایندکس ۲۰ نیست! کم‌تره!

خب حالا قسمت if رو کامل کنیم. ما باید توی متن از اول تا ایندکس i ام رو concatenate کنیم با ایندکس  $i + \text{len}$  که درواقع داریم اون duplicate رو وسطش جا میندازیم و حذف می‌کنیم:

```

def delete_duplication(text, duplicate):
    dup_len = len(duplicate)
    is_first = True
    for i in range(len(text) - dup_len + 1):
        if text[i:i+dup_len] == duplicate:
            if not is_first:

```

```

        text = text[:i] + text[i+dup_len:]
    else:
        is_first = False
    return text

text_input = input("Enter your text: ")
duplicate_input = input("Enter your duplicate: ")
print(delete_duplication(text_input, duplicate_input))

```

برنامه‌نویس خوب اونیه که اول چک کنه و اگر string دوم از اولی بزرگ‌تر بود، همونجا مثلاً به چیزی به عنوان اینکه نامعتبره چاپ کنه و به چیزی هم ریترن کنه که نشون بده ورودی نامعتبر بوده. معمولاً این مواقع None رو return می‌کنن. یعنی هیچی!

```

def delete_duplication(text, duplicate):
    dup_len = len(duplicate)
    if dup_len > len(text):
        return None
    if dup_len == 0: # Empty string
        return None
    is_first = True
    for i in range(len(text) - dup_len + 1):
        if text[i:i+dup_len] == duplicate:
            if not is_first:
                text = text[:i] + text[i+dup_len:]
            else:
                is_first = False
    return text

text_input = input("Enter your text: ")
duplicate_input = input("Enter your duplicate: ")
deleted_duplication = delete_duplication(text_input, duplicate_input)
if deleted_duplication == None:
    print("Invalid input")
else:
    print(deleted_duplication)

```

همونطور که دیدین، من قبل از اینکه چیزی چاپ کنم، اول چک می‌کنم که «None» نباشه! چون اگر باشه، درواقع یعنی ورودی نادرست بوده و پس چاپ می‌کنم که ورودی نادرسته. مثلاً ورودی دوم string خالی باشه!

- چرا چک نکردی که ورودی اول string خالی باشه؟

+ چون نیازی نبود! اگر اولی خالی باشه و دومی پر، خب len دومی بزرگتر از اولیه و if اول کار رو انجام میده.

اگر اولی خالی باشه و دومی هم خالی باشه، خب if دوم کار رو درست انجام میده و باز هم None ریترن میشه. (چون دومی خالی بود!)  
پس هر حالتی که اولی خالی باشه، باز None ریترن میشه و مشکلی نیست!

توجه کنین که None نه True هست و نه False. خودش یه چیز خاص به معنای هیچیه!

این تمرین رو با عمد براتون آورده بودم که شما رو با توابع داخلی پایتون آشنا کنم.  
پایتون یه سری توابع درونی داره که میشه ازشون استفاده کرد و کار رو خیلی سریع تر کرد.  
یه خرده با توابع عادی فرق دارن.<sup>۶۱</sup> درواقع مخصوص string ها هستن. یعنی درواقع برای string ها هستن. برای همین با یه نقطه به string ها پیوند می خورن! این ها رو method (متد) صدا می زنیم.  
مثلاً:

```
string = "ABcdE"  
print(string.isalpha()) # True
```

متد isalpha، به ما میگه که آیا همه کرکترای این string، جزء حروف الفبا هستن یا نه؟ مقدار boolean بر می گدونه. مثلاً بالایی رو True خروجی میده. اما مثال زیر به خاطر اینکه عدد داره و همش الفبا نیست، False خروجی میده.

```
string = "Aa4b"  
print(string.isalpha())
```

حتی می تونین توی IDE یا Text editor تون، موس رو روی متدها ببرین تا بهتون توضیح بده. مثلاً:

```
string = "Aa4b"  
print(string.isalnum())
```

```
def isalnum() -> bool  
Return True if the string is an alpha-numeric string,  
False otherwise.  
  
A string is alpha-numeric if all characters in the string  
are alpha-numeric and  
there is at least one character in the string.  
  
Full name: builtins.str.isalnum
```

۶۱ برای اونایی که با برنامه نویسی شیء گرا هستن، این متدهای built-in کلاس string هستن!

مثلاً اینجا نوشته که `isalnum`، به `bool` برمی‌گردد. توضیح هم زیرش داده که:

Return True if the string is an alpha-numeric string, False otherwise.

یعنی اگر صرفاً شامل حروف الفبا و عدد بود، `True` برمی‌گردد. اگر نبود، `False` برمی‌گردد.

متدهای مهم دیگر:

```
string = "aBC"  
print(string.islower())
```

اگر صرفاً حروف کوچک الفبا باشد، `True`. اگر نه `False`.

```
string = "FS"  
print(string.isupper())
```

اگر صرفاً حروف بزرگ الفبا باشد، `True`. اگر نه `False`.

```
string = "743928"  
print(string.isdigit())
```

اگر صرفاً `digit` (رقم) باشد، `True` برمی‌گردد.

```
string = "223"  
print(string.isnumeric())
```

اگر صرفاً عدد باشد.

فرق بین `isnumeric` و `isdigit` چیست؟

راحت می‌تونین سرچ کنین و فرقتشو پیدا کنین. کاری نداره! بنویسین:

What is the difference between `isdigit()` and `isnumeric()`?<sup>62</sup>

```
uppercase = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"  
lowercase = uppercase.lower()  
print(lowercase)
```

تبدیل به `lowercase` توسط `lower`.

اینا متدها می‌تونن پارامتر هم بگیرن توی پرانتزشون. مثلاً:

```
string = 'Hello! My name is John'  
string = string.replace('name', '!!!')  
print(string)
```

مثلاً متد `replace`، اولی استرینگی که می‌خواد رو می‌گیره و با دومی عوض می‌کنه. اینجا `name` رو با «!!!» عوض کرد.

<sup>62</sup> من سرچ کردم به وبسایت Stackoverflow که یکی از وبسایت‌های معروف در زمینه سؤال برنامه‌نویسی هست رسیدم:

<https://stackoverflow.com/questions/44891070/whats-the-difference-between-str-isdigit-isnumeric-and-isdecimal-in-pyth>

مثلاً برای همین سؤال قبلی، به جای اینکه خودمون حذف کنیم، می‌تونیم از متد استفاده کنیم. متدهای مورد نیاز:

```
string.find('example')
```

یه `string` توی پرانتز بهش میدیم. حالا توی متن ما می‌گردد و ایندکس اولین کرکتر `example` رو توی متن که پیدا شده رو بهمون میده. اگر پیدا نکرد، ۱- رو بهمون میده. مثلاً:

```
print('abede'.find('e'))
```

همونطور که دیدین، به جای متغیر، می‌تونیم `string` رو مستقیم پاس بدیم. بهمون ۲ یعنی اولین `index` رو میده.

### نقطه فکری:

اول اولین `index` استرینگ رو توی متن پیدا می‌کنیم. بعد تمام اون `string` ها رو تو متن حذف می‌کنیم (درواقع با استرینگ خالی جایگزینش می‌کنیم و بعدش اون `string` رو در سر جای اولش می‌گذاریم. (که یعنی درواقع اون اولین `string` که حذف شده بود، برگرده)

```
1. def delete_duplication(text, duplicate):
2.     first_index = text.find(duplicate) # Find first index of duplicate
3.     text = text.replace(duplicate, "") # Replace all duplicate with empty string
4.     text = text[:first_index] + duplicate + text[first_index:] # Add duplicate to first index
5.     return text
6.
7.
8. text_input = input("Enter your text: ")
9. duplicate_input = input("Enter your duplicate: ")
10. print(delete_duplication(text_input, duplicate_input))
```

اما یادتونه که `horizontal scrolling` نباید وجود داشته باشه؟ پس کد رو یکم تمیزتر می‌کنیم. مثلاً کامنت خط ۳ رو می‌برم بالای خط:

```
def delete_duplication(text, duplicate):
    first_index = text.find(duplicate) # Find first index of duplicate
```



```
# Replace all duplicate with empty string
text = text.replace(duplicate, "")
text = text[:first_index] + duplicate + text[first_index:] # Add
duplicate to first index
return text
```

```
text_input = input("Enter your text: ")
duplicate_input = input("Enter your duplicate: ")
print(delete_duplication(text_input, duplicate_input))
```

و اما یه چیز دیگه. خط ۴ هم خوب نیست! اگر گزینه reformat یا «auto format on save» رو توی تنظیمات فعال کرده باشین، خودش براتون تمیزش می‌کنه. وگرنه خودتون می‌تونین اینطوری تمیزش کنین:

```
def delete_duplication(text, duplicate):
    first_index = text.find(duplicate) # Find first index of duplicate
    # Replace all duplicate with empty string
    text = text.replace(duplicate, "")
    text = text[:first_index] + duplicate + \
        text[first_index:] # Add duplicate to first index
    return text
```

```
text_input = input("Enter your text: ")
duplicate_input = input("Enter your duplicate: ")
print(delete_duplication(text_input, duplicate_input))
```

می‌تونیم بعد چیزایی مثل کاما و علامت جمع و اینا، یه بک‌اسلش «\» بگذاریم. یادتونه گفتیم که بک‌اسلش یه کرکتر خاصه؟ اینجا نشون میده که ادامهش خط پایینه.

قدرت پایتون دقیقاً به همینه که اون الگوریتم با for پیچیده رو اینطوری ساده‌تر با چهارتا متد پیاده‌سازی کردیم. این قدرت پایتونه که کار شما رو راحت می‌کنه.

خب حالا تا اینجا مییم، بذارین بهتون یاد بدم که string چند خطی چه‌جور می‌تونین بنویسین:

```
string = "hi this is a test string for testing" + \
    "to show multi line string in python" + \
    "and how to use it"
```

- چرا وقتی پرینتش کردم، به خط قبلی چسبید؟ چون گفتیم که ادامهش! پس اگر می‌خوانین نجسبه، یه فاصله بذارین:

```
string = "hi this is a test string for testing" + \
    " to show multi line string in python" + \
```

```
" and how to use it"
```

البته اینطوری هم می‌تونستیم بنویسیمش:

```
string = "hi this is a test string for testing" \
" to show multi line string in python" \
" and how to use it"
```

خود پایتون متوجه میشه که چون بینش هیچی نیست، ادامه.

البته بهتر می‌تونستیم بنویسیمش. درواقع string چندخطی رو اینطوری می‌نویسن:

```
string = '''hi this is a test string for testing
to show multi line string in python
and how to use it'''
```

چاپش کنین تا ببینین عیناً چیزی که نوشتین چاپ میشه!

لیست built-in method ها رو از لینک زیر می‌تونین بخونین:

[https://www.w3schools.com/python/python\\_ref\\_string.asp](https://www.w3schools.com/python/python_ref_string.asp)

روی هر کدام کلیک کنین، توضیح و مثال داره.

تمرین کنین روش.

String هایی که که توی [] یا () هستن، بهش میگن لیست و tuple که فعلاً نیاز نیست بخونیدشون.

## • List

فرض کنین من می‌خوام یه سری چیز پشت هم نگه دارم. یه data type یا یه سری متغیر خاص تعبیه شدن برای کارهای مختلف. مثلاً String، Float، Integer و ... یه نوع دیگه هست بهش می‌گن «List». شکلش اینجوریه:

```
mylist = [1, 2, 5, -1, -7, 200]
print(mylist)
```

عنصرها توی براکت و با کاما از هم جدا میشن.

فرض کنین می‌خواهین لیست نمرات دانشجویها رو یه جا نگه دارین، این خیلی به کمکتون میاد!

تازه تمام قدرت List رو هنوز ندیدین! تمام قدرتش، انعطاف‌پذیریشه! یعنی همه چیز می‌تونه توش

باشه. یعنی می‌تونین هم String، هم Integer و ... همش توی یه لیست باشه!

```
mylist = [1.28, 'John', -256]
print(mylist)
```

گاهی که تعداد اعضا زیاده، برای خوشگلی و خوانایی، اینطوری هم می‌نویسن:


```
my_list = ['Bruce Schneier',
           'Edward Felten',
           'Ronald Rivest']
```

که قشنگ تر معلومه تا اینکه بخوايم پشت هم بنويسيم. بلکه زیر هم می نویسیم. ولی یادتون نره که کما رو بگذارین. چون اگر نذارین، concatenate رخ میده و دوتا می چسبن به هم!

لیست هم مثل string، یه سری متد داره! مثلاً متد Append. با موس روش گرفتیم و بهم توضیح داد که چیه:

```
mylist = [1.28, 'John']
mylist.append('a')
```

def append(object: \_T, /) -> None  
Append object to the end of the list.  
Full name: builtins.list.append

 See Real World Examples From GitHub

میگه یه عنصر رو به ته لیست append (اضافه) می کنه. چیزی هم که Return می کنه، None هست.

- چرا None؟

+ چون به لیست اضافه کرد، دیگه خب نیاز نیست چیزی return کنه! اضافه کرده تموم شده دیگه! متغیر تغییر کرده. چیزی نیاز به return کردن نیست! پس بیاین print های زیر رو انجام بدیم که بفهمیم:

```
mylist = [1.28, 'John']
mylist.append('a')
print(mylist) # [1.28, 'John', 'a']
print(mylist.append('Hello')) # None
```

توی print دومی، اول میاد Hello رو اضافه می کنه. حالا اضافه کرد، باید حاصل متد چاپ شه. حاصل متد، همون مقدار return شده هست و خب چون None یعنی هیچی return میشه، پس None چاپ میشه. نمونه دیگه:

<https://www.youtube.com/shorts/jILDRAkiJXY>

بیایم خودمون همین متد رو به وسیله یه تابع خودنوشته پیاده سازی کنیم: حواستون باشه که لیست مثل string که شما string رو با string جمع می کردین و اضافه می کردین، لیست هم نمی تونه با یه متغیر عادی جمع شه! سعی کنین عادی جمع کنین بینین چه اروری میده:

```
mylist = [1]
mylist += 2
```

```
TypeError: 'int' object is not iterable
```

اما می‌تونین عادی با یه لیست جمع کنین:

```
mylist = [1]
mylist += [2]
print(mylist) # [1, 2]
```

یا:

```
value = 2
mylist = [1]
mylist += [value]
print(mylist)
```

یعنی توی براکت می‌گذاریم که لیست بشه.

یا مثل قبل که cast می‌کردیم، میشه که این هم cast کنیم به List:

```
mylist = [1]
list_str = '2'
mylist += list(list_str)
print(mylist) # [1, '2']
```

البته توجه داشته باشیم که نمیشه یه integer رو به لیست cast کرد. اینجا هم برای همین string رو cast کردم.

همینجا که هستیم بدونیم که ضرب یه لیست در یه عدد، باعث میشه که اعضايش همون تعداد بار تکرار شن. (مثلاً استرینگ بودا! اینجا هم همونطوره!):

```
s = 'a'
print(s * 3)
l = [1, 2, 3]
print(l * 3)
```

خب بیاین خودمون متد append رو به صورت یه تابع بنویسیم:

```
def append_to_list(mylist, value):
    mylist.append(value)
    return mylist
```

```
mylist = [1.28, 'John']
value = 'a'
print(append_to_list(mylist, value))
```

این ساده‌ترین حالتش که با کمک همون متد append بود.  
حالت دوم:

```
def append_to_list(mylist, value):  
    value_list = list(value)  
    mylist += value_list  
    return mylist
```

اول `value` رو تبدیل به `list` کردم و بعدش اضافه‌اش کردم. (حواستون باشه `Integer` پاس بدیم، مشکل به وجود میاد!)

خب اما توی همه اینها، خود مقدار `list` تغییر پیدا نمی‌کنه! یعنی اگر `mylist` رو چاپ کنین، می‌بینین که مقدارش تغییر پیدا نکرده. تابع یه مقداری رو ریترن می‌کنه که متفاوت از متغیر اصلی‌ه! یادتونه گفتیم که متغیر درون `function` با متغیر عادی فرقه؟ یعنی هر تغییری توی تابع بدیم، بیرون اعمال نمیشه! اگر بخوایم اعمال شه، صرفاً باید بگیم که توی این تابع، یه متغیر `global` (سراسری) داره استفاده میشه که یعنی تغییراتش همه جا اعمال شه و تغییرات صرفاً محلی (`local`) نباشه:

```
def change():  
    global x  
    x += 1
```

```
x = 1  
change()  
print(x)
```

اینجا نیاز نیست متغیر رو به تابع پاس بدیم. صرفاً می‌گیم که متغیر `x` یه متغیر از بیرونه و `local` نیست. وقتی تابع رو صدا بزنیم، `x` تغییر می‌کنه و مقدارش بیرون هم عوض میشه. پس وقتی `print` اش کنیم، مقدارش ۲ هست!

اگر بخوایم چند تا متغیر استفاده کنیم، بینشون کاما می‌گذاریم:

```
def change():  
    global x, y  
    x += 1  
    y *= 3
```

```
x = 1  
y = 3  
change()  
print(x, y) # 2 9  
print(change()) # None
```

همونطور که دیدین، `print` اولی، مقدار `x` و `y` تغییر پیدا کرده. `print` آخری هم به دلیل اینکه تابع چیزی ریترن نکرده، `None` رو چاپ می‌کنه.

پس اگر بخواهیم متد `append` رو پیاده کنیم، باید بگیم اون لیست ما، `global` هست (که تغییرات در بیرون هم اعمال شن!) و چیزی هم ریترن نکنیم!

```
def append_to_list(value):
    global mylist
    mylist.append(value)

mylist = [1.28, 'John']
value = 'a'
append_to_list(value)

print(mylist) # [1.28, 'John', 'a']
print(append_to_list(value)) # None
```

- `Split()`

تبدیل یه `string` به یه `List`:

```
numbers = input("Enter 3 numbers separated by spaces: ")
mylist = numbers.split()
print(mylist)

numbers = input("Enter 3 numbers separated by commas: ")
mylist = numbers.split(",")
print(mylist)
```

*input:*

```
12 23 -11
12,-1,22
```

*output:*

```
['12', '23', '-11']
['12', '-1', '22']
```

اگر چیزی بهش توی پرانتز ندین، به صورت پیشفرض با فاصله جدا می‌کنه. (مثل مثال اول)  
نکته: می‌تونین از چیزایی مثل «\n» و این‌ها استفاده کنین. یعنی مثلاً وقتی متغیرها توی خط‌های مجزا هستن.

از این متد می‌تونین برای گرفتن چندتا عدد همزمان توی یه خطر بهره بگیرین. ولی حواستون باشه که عدداً به صورت `string` توی یه لیست قرار می‌گیرن.  
متدهای لیست رو می‌تونین از سایت زیر بخونین. همش مثال داره:

[https://www.w3schools.com/python/python\\_lists\\_methods.asp](https://www.w3schools.com/python/python_lists_methods.asp)

شاید از دیدن متد `copy` توی لیست، تعجب کنین. بگین خب من اگر بخوام یه کاپی از مثلاً `l1` داشته باشم، به راحتی می‌تونم اینطوری بنویسم:

```
l1 = [1, 2, 3, 4]
l1_copy = l1
```

دیگه چه نیازی که یکساعت برم اینطوری بنویسم:

```
l1_copy = l1.copy()
```

اما یه دلیلی داره که متد `copy` وجود داره. بیاین کد زیر رو اجرا کنیم:

```
l1 = [1, 2, 3, 4]
l1_copy = l1
l1_copy[0] = 'a'
print(f'l1_copy: {l1_copy}')
print(f'l1: {l1}')
```

عه! چه عجیب! ما قاعداً داشتیم صرفاً کاپی رو تغییر میدادیم و گفتیم ایندکس صفرمش بشه «a». ولی خود `l1` هم تغییر کرد!

دلیلش از دانشی که شما الان دارین خارجه ولی صرفاً این بگم که برای چیزایی مثل لیست (و نه چیزایی مثل متغیرای عادی)، همچین اتفاقی میوفته که موقع `assign` کردن، انگار هردو یه چیز میشن. یا هر دو به یه چیز اشاره می‌کنن. برای همین. خلاصه خیلی خودتون رو درگیر نکنین فعلاً! فقط بدونین برای کاپی گرفتن، باید از متد `copy` استفاده کنین:

```
l1 = [1, 2, 3, 4]
l1_copy = l1.copy()
l1_copy[0] = 'a'
print(f'l1_copy: {l1_copy}')
print(f'l1: {l1}')
```

یا اینطوری هم میشه نوشت:

```
l1_copy = l1[:]
```

بازم انگار کاپی کردیم. یعنی گفتیم دونه دونه ایندکسها رو بریز توی `l1_copy`.

حواستون باشه که وقتی رو یه لیست حرکت می‌کنین، نباید تغییرش بدین. یعنی نباید عضو بهش کم و زیاد کنین. چون باعث ارورها و مشکلاتی که فکرش نمی‌کنین میشه. مثلاً:

```
l = [1, 2, 3, 4]
for element in l:
    l.pop(3)
```

این باعث ارور میشه. چون شما هی دارین ایندکس سوم رو پاک می‌کنین ولی بعد اینکه ایندکس سوم پاک شد، دیگه ایندکس سومی وجود نداره. پس `for` زدن روش بی‌معناس.



این مواقع روی یه لیست حرکت کنین و روی یکی دیگه که کاپیش هست تغییرات اعمال کنین.

مثال:

به شما یه سری نمره به صورت اعشاری داده میشه. تا وقتی که عدد ۱- وارد نشده، هی نمره می گیرین و به لیست اضافه می کنین. در آخر میانگین اعداد اون لیست رو چاپ کنین.

پاسخ:

روش اول:

```
score_list = []
input_score = float(input("Enter your score: "))
while input_score != -1:
    score_list.append(input_score)
    input_score = float(input("Enter your score: "))

sum = 0
for i in range(len(score_list)):
    sum += score_list[i]

print(sum / len(score_list))
```

هی می گیرم و هی به لیست اضافه می کنم. خارج while با for میام جمع رو حساب می کنم و در آخر هم میانگین می گیرم.

اما بیاین یکم کد رو بهینه تر کنیم. کجا داریم اضافه کاری می کنیم؟ قسمت محاسبه len. دوباره داره حساب میشه. پس می تونیم بگیم:

```
score_list = []
input_score = float(input("Enter your score: "))
while input_score != -1:
    score_list.append(input_score)
    input_score = float(input("Enter your score: "))

sum = 0
list_len = len(score_list)
for i in range(list_len):
    sum += score_list[i]

print(sum / list_len)
```

برنامه‌نویس خوب اونیه که حواسش باشه ورودی اشتباه، برنامه رو به هم نریزه. ممکنه نمره وارد نکنن و یه کلمه وارد کنن! پس باید بررسی کنیم که درست وارد شه. اما فعلاً کدش رو نمی‌زنیم. نیاز به چیزی داره که الان بلدش نیستین! ولی خب خط فکری رو داشته باشین که همیشه حواسمون به ورودی باشه!

### روش دوم:

گفتیم `while` تا شرطش ادامه پیدا می‌کنه. یعنی هر بار شرط چک میشه و اگر `True` بود، وارد `while` میشه و اگر نبود، خارج میشه. خب فرض کنیم یه `while` اینطوری داشته باشیم:

```
while True:
```

به نظرتون این چند بار تکرار میشه؟

+ به نظرم چون نوشته `while True`، چون `True` هست، همیشه `True` هست و برای همیشه تکرار میشه! اصطلاحاً بهش میگن «حلقه بی‌نهایت» یا همون «infinity loop».

این یکی از ارورای رایج هست که شرط جلوی `while` ممکنه طبق مقایسه یا اینکه حواستون نبوده متغیر `boolean` همیشه یه مقدار داره رو گذاشتین جلوی `while` و همیشه تکرار میشه و برنامتون اونجا گیر می‌کنه.

برای اینکه برنامتون اونجا گیر نکنه، باید یه جایی بگین بسه دیگه! این با چی بود؟ با `break`. مثلاً همین سؤال رو با `break` پیاده‌سازی می‌کنیم:

```
score_list = []
input_score = float(input("Enter your score: "))
while True:
    if input_score == -1:
        break
    score_list.append(input_score)
    input_score = float(input("Enter your score: "))

sum = 0
list_len = len(score_list)
for i in range(list_len):
    sum += score_list[i]

print(sum / list_len)
```

یعنی `while` بینهایت. هر بار ولی چک می‌کنم که اگر ورودی برابر -۱ بود، `break` کنه و از حلقه بپره بیرون!

البته یه نکته! ما یه تابع `sum` داریم که تابع درونی پایتون هست که جمع رو حساب می‌کنه. یعنی مثلاً به جای `for` می‌تونستیم اینطوری بگیم:

```

score_list = []
input_score = float(input("Enter your score: "))
while True:
    if input_score == -1:
        break
    score_list.append(input_score)
    input_score = float(input("Enter your score: "))

list_sum = sum(score_list)

print(list_sum / len(score_list))

```

برای همین بهتر بود که توی راه قبلی، اسم متغیر رو sum تعریف نمی کردیم. چون یادتونه قرار بود اسمایی با نام یکسان با توابع درونی پایتون انتخاب نکنیم؟!

حواستون باشه که اگر برنامه تون خیلی طول کشید و یا صدای فن لپ تاپتون بلند شد، احتمالاً توی لوپ بی نهایت افتادین! با `ctrl + c` می تونین متوقفش کنین. یا خود IDE ها دکمه توقف دارن.

### مثال:

یه سری عدد به شکل

1 -1 2 4 -3

بهتون میدن و شما باید بریزیدش توی لیست و هر دفعه جمع دوتا کناری ها رو حساب کنین و لیست جدید رو بسازین. این کار رو تا وقتی ادامه بدین که لیست یه عنصر داشته باشه و اون یه عنصر رو چاپ کنین:

```

l = [1, -1, 2, 4, -3]
l = [0, 1, 6, 1]
l = [1, 7, 7]
l = [8, 14]
l = [22]

```

برای گرفتن اعداد چند راه دارین.

۱- اول به صورت string بگیرین و با متد `split` جداش کنین.

۲- به صورت string بگیرین و بعدش خودتون `for` بزنین و اعداد رو جدا کنین و دونه دونه به لیست `append` کنین.

### پاسخ:

یه تابع می‌نویسیم که وقتی یه لیست رو بهش پاس بدیم، توی یه لیست جدید، جمع دوتا دوتا عناصر رو حساب کنه و قرار بده:

```
def two_member_sum(num_list):  
    ''' Get a list and return a new list with the  
    sum of each two members of the list which are  
    next to each other.  
    '''  
    new_list = []  
    for i in range(len(num_list) - 1):  
        new_list.append(num_list[i] + num_list[i + 1])  
    return new_list
```

همونطور که دیدین، کامنت هم گذاشتم که فردی که می‌خونه متوجه شه.

تا حالا چندبار درباره out of range صحبت کردیم. حواستون هست که چرا شرط for ما تا قبل len - 1 میره؟<sup>۶۳</sup>

حالا میایم یه سری عدد از کاربر می‌گیریم، به لیست تبدیلش می‌کنیم و تا وقتی که len لیست برابر ۱ نشده، هی باید دوتا دوتا جمع کنیم:

```
def two_member_sum(num_list):  
    ''' Get a list and return a new list with the  
    sum of each two members of the list which are  
    next to each other.  
    '''  
    new_list = []  
    for i in range(len(num_list) - 1):  
        new_list.append(num_list[i] + num_list[i + 1])  
    return new_list
```

```
numbers = input()  
num_list = numbers.split()  
list_len = len(num_list)  
while list_len > 1:  
    num_list = two_member_sum(num_list)  
    list_len = len(num_list)
```

```
print(num_list[0])
```

در آخر هم چون لیست ما یه عنصر داره، ایندکس صفرم رو پرینت می‌کنیم که جوابه. خب تستش کنیم ببینیم چی خروجی میده:

---

<sup>۶۳</sup> چون خط بعدیش ما داریم  $i + 1$  حساب می‌کنیم و خب نزنه بیرون! برای همین  $len - 1$

input:

1 -1 2 4 -3

output:

1-1-12-1224-1224244-3

- عه چرا اينطور شد؟! انگار به هم چسبیدن. انگار concatenate شدن كه!  
+ آره دقيقاً! يعنى انگار عدداً به صورت string بودن كه concatenate شدن. يكم فكر كنين كه كجا ممكنه اين رخ داده باشه؟  
قسمت for درون تابع جمع داريم كه چون حدس زديم concatenate شده، احتمالاً مشكل از اينجاست. ولى خب چرا اعداد به صورت string هستن؟  
+ چون متد split اگر يادتون باشه، string رو تقسيم به چند string كوچكتر مى كرد! پس نياز به موقع جمع، cast كنيم به Integer:

```
def two_member_sum(num_list):  
    ''' Get a list and return a new list with the  
        sum of each two members of the list which are  
        next to each other.  
    '''  
    new_list = []  
    for i in range(len(num_list) - 1):  
        new_list.append(int(num_list[i]) + int(num_list[i + 1]))  
    return new_list
```

```
numbers = input()  
num_list = numbers.split()  
list_len = len(num_list)  
while list_len > 1:  
    num_list = two_member_sum(num_list)  
    list_len = len(num_list)
```

```
print(num_list[0])
```

هميشه موقع ارور فكر كنين كه چي رخ داده كه اينطور شده. من موقع چسبیدنشون به هم، سريع ذهني سمت concatenate كردن در string رفت.

يه ارور رايج:

<https://www.youtube.com/shorts/Zn4Hh5fY2II>

۱- سؤال در لینک زیر:

<https://quera.org/problemset/1359/>

پاسخ ا:

خب یه تابع می‌نویسیم که ببینه استرینگ دوم (sub) توی string اول (main) ترتیبش وجود داره یا نه؟

یه بار خود استرینگ و یه بار برعکس رو بهش پاس میدیم. اینطوری از چپ به راست و راست به چپ هردو بررسی میشه! نیازه به نوشتن دو تابع نیست! صرفاً یه بار خودش و یه بار برعکسش رو به تابع پاس میدیم.

خب نیازه که از ایندکس صفر sub شروع کنیم. و روی main حرکت کنیم تا ببینیم کجا پیداش می‌کنیم؟ یعنی هی پیش میریم روی main تا وقتی بالاخره پیداش کنیم. اگر به ته main رسیدیم و پیدا نشد، خب return می‌کنیم False رو.

هر جا پیداش کردیم، میریم سراغ ایندکس بعدی sub و از ادامه main پیش میریم تا ببینیم این هم پیدا میشه؟ یعنی تا وقتی که برابر نشد با یه ایندکس.

این روند تا کجا پیش میره؟ تا وقتی که به ته یکی از string ها برسیم.

```
def is_sequenced(main, sub):
    main_i = sub_i = 0
    counter = 0
    while main_i < len(main) and sub_i < len(sub):
        while main[main_i] != sub[sub_i]:
            main_i += 1
        if main_i >= len(main):
            return False
        sub_i += 1
        main_i += 1
        counter += 1

    if counter == len(sub):
        return True

    return False
```

تا وقتی به پایان یکیشون نرسیدیم، میریم جلوتر. هر بار میندازمش توی یه حلقه که تا وقتی که sub رو توی main پیدا نکرده، همونجا بمونه تا پیداش کنه.

خب اگر رسیدیم ته main و پیدا نشد چی؟ یعنی پیدا نشده دیگه! پس شرط می‌ذارم که اینو چک چک کنه که اگر ایندکس بزرگ‌تر یا مساوی len شد، False رو ریترن کنه.

هر بار که پیدا شد، از حلقه میاد بیرون و خب یه دونه باید هم از main و هم از sub بره جلو.

حالا من یه counter هم گذاشتم. به دلش فکر کنین!

دلیل اینه که از while خارجی، زمانی میاد بیرون که یا sub تموم شده باشه و یا main تموم شده باشه.

اگر sub تموم شده باشه که خیالمون راحت‌تره که همه عناصر بودن. مثلاً ایندکس آخر رو در نظر بگیریم. میوفته توی while درونی و تا وقتی پیدا نشه، از توش بیرون نمیداد! این برای تمام ایندکس‌ها اتفاق میوفته.

اما اگر main تموم شه، تضمینی وجود نداره که همش توش پیدا شده باشه!

پس من counter گذاشتم که تعداد پیدا شده‌ها رو یه جا نگه دارم و بعد خروج از while بیرونی، ببینم همش پیدا شده یا نه. اگر آره خب True رو ریترن کنه. در غیر این صورت میره خط آخر و False رو ریترن می‌کنه.

توی این تابع، چندبار len حساب شده. می‌تونستیم len رو یه جا نگه داریم که نخوایم هر بار حسابش کنیم. مثلاً هر بار موقع چک شرط while، میاد len رو حساب می‌کنه و این باعث کندشدن و کار بیخودی کد ما میشه.

خب بریم خارج تابع رو هم تکمیل کنیم:

```
def is_sequenced(main, sub):
    main_i = sub_i = 0
    counter = 0
    while main_i < len(main) and sub_i < len(sub):
        while main[main_i] != sub[sub_i]:
            main_i += 1
            if main_i >= len(main):
                return False
            sub_i += 1
            main_i += 1
            counter += 1

    if counter == len(sub):
        return True
```

```

return False

result_list = []
n = int(input())
for i in range(n):
    main = input()
    sub = input()
    res = is_sequenced(main, sub)
    if not res:
        res = is_sequenced(main, sub[::-1])
    result_list.append(res)

for i in range(n):
    if result_list[i]:
        print("YES")
    else:
        print("NO")

```

به تعدادی که گفته main و sub می‌گیرم و بار اول خودشونو میدم به تابع. حالا به نظرتون چرا if not res نوشتم؟ فکر کنین روش! چون اگر res مقدارش True باشه، یعنی پیدا شده و نیازی نیست که برای برعکسشم یه دور دیگه تابع رو صدا بزنم! صرفاً تابع رو زمانی صدا می‌زنم که res مقدارش False باشه! بعدش به نظرتون چرا sub رو برعکس کردم؟ چرا بهتر از این بود که main رو برعکس کنم؟ + چون main بزرگ‌تره و برعکس کردنش زمان بیشتری می‌بره. پس اونی که کوتاه‌تره رو برعکس می‌کنم که قدم یه خرده سریع‌تر شه باز. همچنین نیاز نبود مقادیر True یا False رو بریزم توی یه لیست و بعداً دوباره روی لیست حرکت کنم و اگر به True خوردم، چاپ کنم YES و اگر False بود، چاپ کنم NO. همونجا می‌تونستم پرینتش کنم. دلیلشم این بود که Quera مقادیر خروجی رو جدا از ورودی حساب می‌کنه و خب براش مهم نیست همون لحظه چاپ شه یا بعد دادن تمام ورودی‌ها. پس کد بهینه‌تر رو می‌نویسیم:

```

def is_sequenced(main, sub):
    main_len = len(main)
    sub_len = len(sub)
    main_i = sub_i = 0
    counter = 0
    while main_i < main_len and sub_i < sub_len:

```



```

    while main[main_i] != sub[sub_i]:
        main_i += 1
    if main_i >= main_len:
        return False
    sub_i += 1
    main_i += 1
    counter += 1

    if counter == sub_len:
        return True

    return False

n = int(input())
for i in range(n):
    main = input()
    sub = input()
    res = is_sequenced(main, sub)
    if not res:
        res = is_sequenced(main, sub[::-1])
    if res:
        print("YES")
    else:
        print("NO")

```

## • List comprehension

مثال:

اعداد ۱ تا ۱۰ رو بریزین داخل یه لیست:

```

mylist = []
for i in range(1, 11):
    mylist.append(i)
print(mylist)

```

می‌تونیم همین for رو درون List به کار ببریم:

```
mylist = [i for i in range(1, 11)]
print(mylist)
```

میگیم به سری i درون mylist باشن که i ها در رنج ۱ تا قبل ۱۱ هستن.

یادتونه که موقع split کردن، string ریخته میشد توی لیست؟ خب فرض کنین بخوایم تک تک تبدیلش کنیم به Integer، باید به for بزنیم:

```
num_list = ['1', '2', '3', '4']
for i in range(len(num_list)):
    num_list[i] = int(num_list[i])
print(num_list)
```

اما خب میشه این رو ساده تر نوشت. یعنی:

```
num_list = ['1', '2', '3', '4']
num_list = [int(i) for i in num_list]
print(num_list)
```

تمیزتر نشد؟

میگیم int(i) ها رو بریز تو لیست؛ حالا int کدوم i ها؟ اونایی که داخل num\_list هستن. یعنی سؤال رو می تونستیم اینطوری حل کنیم:

```
def two_member_sum(num_list):
    ''' Get a list and return a new list with the
    sum of each two members of the list which are
    next to each other.
    '''
    new_list = []
    for i in range(len(num_list) - 1):
        new_list.append(num_list[i] + num_list[i + 1])
    return new_list
```

```
numbers = input()
num_list = numbers.split()
num_list = [int(i) for i in num_list]
list_len = len(num_list)
while list_len > 1:
    num_list = two_member_sum(num_list)
    list_len = len(num_list)

print(num_list[0])
```

مثالای دیگه از loop درون list:

```
mylist = [(i + 1) for i in range(10)]
```

توضیح: یه سری  $i + 1$  رو بذار توی لیست.  $i$  ها چین؟  $i$  ها از ۰ تا ۹ هستن.

```
l1 = ['a', 'b', 'c']  
l2 = [char.upper() for char in l1]  
print(l2)
```

توضیح: کرکترها رو uppercase کن و بذار توی لیست. خود کرکتر از کجا بیان؟ خود کرکتر، اونایی هستن که توی l1 وجود دارن.

نه تنها list comprehension ساده تره و کوتاه تره، بلکه سریع تره هست! (به جای اینکه یه for بنویسیم و مقادیر رو append کنیم).

## • Nested for

```
for i in range(3):  
    for j in range(3, 6):  
        print(f'i: {i}, j: {j}')  
    print('-----')
```

پرینت کنین ببینین چی میشه.

اول وارد for میشه و  $i = 0$ ، بعدش وارد for دومی میشه و  $j = 0$  و پرینتش می کنه. بعدش میره بالا، یه دونه به اضافه میشه و تا وقتی که به ۵ برسه. یعنی این روند تکرار میشه:

```
i: 0, j: 3  
i: 0, j: 4  
i: 0, j: 5
```

بعدش که  $j = 6$  ز شد، از for داخلی می پره بیرون و ادامه رو طی می کنه. می رسه به خطی که چندتا خط چین پرینت می کنه. بعدش میره بالا و  $i$  میشه ۱. دوباره میاد توی for دومی.  $j = 0$  تا  $j = 2$  پیش میره. این روند ادامه پیدا می کنه. با دیباگر IDE یا Text editor تون تستش کنین قشنگ تر متوجه میشین.

```
i: 0, j: 3  
i: 0, j: 4  
i: 0, j: 5  
-----  
i: 1, j: 3  
i: 1, j: 4  
i: 1, j: 5
```

```

-----
i: 2, j: 3
i: 2, j: 4
i: 2, j: 5
-----

```

به این‌ها می‌گن nested for. یعنی for توی for. معمولاً توی for های تو در تو، i و j و k به کار می‌برن:

```

for i in range(3):
    for j in range(3, 6):
        for k in range(6, 9):
            print(f'i: {i}, j: {j}, k: {k}')
        print('-----')
    print('=====')

```

یکم با nested for ها بازی کنین تا کامل درکش کنین.

## • Time complexity

خب به نظرتون پیچیدگی زمانی «nested for» ها چقدره؟ مثلاً کد زیر:

```

for i in range(n):
    for j in range(n):
        print(i, j)
    print(f'i = {i}')

```

خب ببینین ما باید ببینیم که کدوم قسمت کد بیشترین بار اجرا میشه؟  
 قاعدتاً قسمت `print(i, j)` درون حلقه دوم for بیشترین بار اجرا میشه. درواقع بیایم فکری حساب کنیم:

$n = 1 \rightarrow 1 \text{ times}$   
 $n = 2 \rightarrow 4 \text{ times}$   
 $n = 3 \rightarrow 9 \text{ times}$   
 $n = 4 \rightarrow 16 \text{ times}$

یا کدی رو هم می‌تونستیم تغییر بدیم ببینیم چندبار اجرا شده:

```

n = 3
count = 1
for i in range(n):
    for j in range(n):
        print(count)
        count += 1

```

درواقع می‌بینیم که داره  $n^2$  بار اجرا میشه. یعنی پیچیدگی زمانیش  $n^2$  هست. یا بخوایم ذهنی هم بگیریم، میگیریم حلقه اول  $n$  بار اجرا میشه. توی هر بار اجراش هم  $n$  بار حلقه دوم اجرا میشه. پس میشه  $n * n$  یا همون  $n^2$ . پس درواقع هرچی حلقه‌هاتون بیشتر شه، پیچیدگی زمانیتون بیشتر میشه.

مثال:

ماتریس زیر رو در خروجی چاپ کنین:

```
1 2 3
4 5 6
7 8 9
```

پاسخ:

```
for i in range(1, 10):
    print(i, end=' ')
    if i % 3 == 0:
        print()
```

- عه چرا nested for نرفتی؟ بهش می‌خورد nested for باشه‌ها!  
+ هر گردی گردو نیست! قبل حل مسأله فکر کنین که چه راهی مناسب‌تره!

خب حالا برای کاربرد nested for، بیاین مثال زیر رو حل کنیم:

## • Bubble Sort ( $\theta(n^2)$ )<sup>۶۴</sup>

توضیحات رو از وبسایت زیر بخونین:

<https://www.geeksforgeeks.org/bubble-sort/>

پس باید روی تک‌تک اعضا حرکت کنیم و دونه‌دونه با جلوییش چک کنیم ببینیم که نیازه swap شون کنیم یا نه؟

Input: 5 4 3 2 1

First time:

5 4 3 2 1 → Compare  $i = 0$ ,  $i = 1$

4 5 3 2 1 → Compare  $i = 1$ ,  $i = 2$

4 3 5 2 1 → Compare  $i = 2$ ,  $i = 3$

4 3 2 5 1 → Compare  $i = 3$ ,  $i = 4$

4 3 2 1 5

خب بار اول انجام شد. به نظرتون چه اتفاقی افتاد؟  
+ عدد ماکزیموم رفت آخر.

۶۴ تایم کامپلکسیتی  $n^2$  هست.

آره دقیقاً درسته!

Second time:

4 3 2 1 5 → Compare  $i = 0, i = 1$

3 4 2 1 5 → Compare  $i = 1, i = 2$

3 2 4 1 5 → Compare  $i = 2, i = 3$

3 2 1 4 5

آیا نیاز به ۴ و ۵ رو چک کنیم؟ نه! چون گفتیم عدد ماکزیموم رفته آخر. پس درواقع هر بار یه دونه چک کردنمون کمتر میشه. درواقع الکی نیاز به چک کردن نیست! چون مطمئنیم که نیاز به جابه‌جایی نیست.

Third time:

3 2 1 4 5 → Compare  $i = 0, i = 1$

2 3 1 4 5 → Compare  $i = 1, i = 2$

2 1 3 4 5

دیگه نیاز نیست با یکی مونده به آخری (عدد ۴) چک کنیم. چرا؟ چون گفتیم هر بار بزرگترین میره ته. اول ۵ رفت. بعد ۴. پس ۴ حتماً بزرگ‌تر از هر عددی هست که اینجا قرار گرفته. پس درواقع می‌تونیم به یه الگو برسیم که تا کجا پیش میریم. دفعه سوم، تا قبل از دوتا مونده به آخر.

Fourth time:

2 1 3 4 5 → Compare  $i = 0, i = 1$

1 2 3 4 5

Fifth time:

1 2 3 4 5 → Done!

خب مرتب شده و نیازی به کاری نیست.

خب پس درواقع ما ۵ بار باید یه سری کار انجام بدیم. (به تعداد اعضای لیست)

- چه کارهایی؟

+ هی بریم جلوتر و دوتا دوتا چک کنیم و اگر نیاز بود جاشونو عوض کنیم.

- تا کجا پیش بریم؟

+ درواقع بیایم for رو بنویسیم تا بهتر درکش کنیم:

```
def sort_list(num_list): # Bubble sort
    length = len(num_list)
    for i in range(length):
        for j in range():
            if num_list[j] > num_list[j + 1]:
                # swap
    return num_list
```

for بیرونی که کارش صرفاً اینه که به تعداد عناصر یه سری کار رو انجام بده.  
- اون کارا چی هستن؟

+ توی for دومی انجامش میدیم.

درواقع توی for درونی میگیریم که اگر ایندکس قبلی از بعدی بزرگتر بود، نیاز به اینه که swap کنیم.  
(جابه‌جا کنیم) اما فعلاً کدش رو ننوشتیم که با خط فکری مسأله آشنا شین.

خب for دومی تا کجا باید پیش بره؟ درواقع به دلیل قسمت if که  $j + 1$  داره، تا  $length - 1$ . اما  
خب یادمونه که نیازی به چک کردن یه سری چیزا نبود. مثلاً مرحله دوم، نیاز به چک کردن با آخری عنصر  
نبود. یعنی:

مرحله اول:  $i = 0$  و باید با همه عناصر چک کنیم. یعنی for باید تا  $length$  پیش بره.

مرحله دوم:  $i = 1$  و جز با عنصر آخر باید چک کنیم. یعنی for باید تا  $length - 1$  پیش بره.

مرحله سوم:  $i = 2$  و جز با دو عنصر آخری باید چک کنیم. یعنی for باید تا  $length - 2$  پیش بره.

الگو رو پیدا کردین؟ پس درواقع رنج  $j$  ها در for دومی باید تا  $length - i$  پیش بره. و به خاطر  $j + 1$  یکی کم میشه که out of range نشه و تا ایندکس آخر بره نه تا ایندکس آخر + 1.

پس درواقع بیایم تابع رو بنویسیم و قسمت swap هم کامل کنیم. (swap کردن رو که یادتون نرفته؟

اوایل یه سؤال swap کردن حل کردیم با هم!)

```
def sort_list(num_list): # Bubble sort
    length = len(num_list)
    for i in range(length):
        for j in range(length - i - 1):
            if num_list[j] > num_list[j + 1]: # Swap
                temp = num_list[j]
                num_list[j] = num_list[j + 1]
                num_list[j + 1] = temp
    return num_list
```

```
numbers_str = input("Enter numbers separated by space: ")
num_list = numbers_str.split()
num_list = [int(i) for i in num_list]
num_list = sort_list(num_list)
print(num_list)
```

پایتون گفته اگر دو یا چندتا چیز رو می‌خوای همزمان تعیین کنی، من یه راهکار میدم بهت. اینطوری  
بنویس:

```
num1, num2, num3 = 1, 2, 3
print(f'num1: {num1}, num2: {num2}, num3: {num3}')
```

یعنی num1 و num2 و num3 به ترتیب ۱ و ۲ و ۳ باشند.  
از این تکنیک می‌تونیم برای عوض کردن و swap استفاده کنیم:

```
num1 = 1
num2 = 2
num1, num2 = num2, num1
print(f'num1: {num1}, num2: {num2}')
```

یعنی num1 و num2 از این به بعد به ترتیب num2 و num1 هستند.  
جالب بود نه؟! پس کد پاسخ سؤال رو تمیزتر بنویسیم:

```
def sort_list(num_list): # Bubble sort
    length = len(num_list)
    for i in range(length):
        for j in range(length - i - 1):
            if num_list[j] > num_list[j + 1]:
                num_list[j], num_list[j + 1] = num_list[j + 1],
num_list[j]
    return num_list
```

```
numbers_str = input("Enter numbers separated by space: ")
num_list = numbers_str.split()
num_list = [int(i) for i in num_list]
num_list = sort_list(num_list)
print(num_list)
```

## • Insertion Sort ( $\Omega(n), O(n^2)$ )<sup>۶۵</sup>

ما باید برخلاف Bubble Sort که ماکزیموم می‌بردیم آخر، minimum رو بیاریم اول.<sup>۶۶</sup> یعنی:

input:

5 4 3 2 1

First time:

5 4 3 2 1

4 5 3 2 1

---

<sup>۶۵</sup> این چیه؟ اگر نمی‌دونین چیه، نیازی نیست بدونین! صرفاً آوردم برای کسانی که می‌دونن. بهش میگن time complexity که توی درس تحلیل الگوریتم و اینا می‌خونین. ولی به طور کلی بدونین درباره تایم کامپلکسیتی هست.

<sup>۶۶</sup> More info: <https://www.geeksforgeeks.org/insertion-sort/>



Second time:

4 5 3 2 1

4 3 5 2 1

3 4 5 2 1

Third time:

3 4 5 2 1

3 4 2 5 1

3 2 4 5 1

2 3 4 5 1

Fourth time:

2 3 4 5 1

2 3 4 1 5

2 3 1 4 5

2 1 3 4 5

1 2 3 4 5

Fifth time:

1 2 3 4 5

• یعنی بار اول، ایندکس ۱ با ایندکس ۰

• بار دوم، ایندکس ۲ با ۱ و بعدش ۱ با ۰

همینطور ادامه پیدا می‌کنه.

بار پنجم عملاً کاری انجام نمیشه. پس نیازی نیست ۵ بار تکرار کنیم. ۴ بار (یکی از تعداد اعضا کمتر)،

انجامش بدیم، یعنی درست شده!

پس درواقع باید به `for` بزنی به تعداد `len - 1` و داخلش به سری کار کنیم. اینکه بار اول، ایندکس ۱

با صفر چک شه.

به نظرتون بهتر نیست `for` رو از ۱ شروع کنیم؟ ۱ تا قبل `len` (که میشه `len - 1` بار).

این به این درد می‌خوره که بار اول ایندکس ۱ هست و دقیقاً باید ایندکس ۱ رو با ایندکس ۰ مقایسه

کنیم. یعنی هر بار ایندکس با ایندکس یکی قبل‌تر از خودش. اما اگر `for` رو از ۰ شروع کنیم، به علاوه

یک نیازه هی حساب کنیم.

هر بار با قبلش چک میشه و اگر نیاز بود، هی میره عقب‌تر و هی `swap` می‌کنه.

- نیاز بود یعنی چی؟

+ یعنی اگر یه مقایسه کردیم که دیدیم عنصرمون از عقبی کوچکتتر نیست، دیگه نیاز نیست قبلش چک کنیم. چون گفتیم هر بار مینیموم میره ته. پس نیازی به چک کردن اونا نیست!

```
def insertion_sort(l):  
    for i in range(1, len(l)):  
        j = i  
        while l[j] < l[j - 1]:  
            l[j], l[j - 1] = l[j - 1], l[j] # swap  
            j -= 1  
    return l
```

یعنی هی میریم عقب تا وقتی که هی کوچکتتر باشه و باید بره ته. اگر دیدیم کوچکتتر نیست، پس قبلش هم درست و نیاز نیست بره ته. (چون هر بار مینیموم می‌رفت ته)

اما یه چیزی ممکنه درست نباشه! زهی داره کم میشه و یه جایی صفر میشه عملاً میشه ایندکس منفی! الکی داره میچرخه! به خاطر  $j - 1$  توی `while`، عملاً وقتی  $j = 1$  بشه، ایندکس صفر توی `l[j - 1]` بررسی میشه و اوکیه!

پس یه شرط دیگه اضافه می‌کنیم که تا وقتی که  $j$  بزرگ‌تر از صفره نیازه توی `while` بمونه:

```
def insertion_sort(i):  
    for i in range(1, len(l)):  
        j = i  
        while j > 0 and l[j] < l[j - 1]:  
            l[j], l[j - 1] = l[j - 1], l[j] # swap  
            j -= 1  
    return l
```

شرط  $j > 0$  هم اول می‌گذاریم که `out of range` نشه. چون یادتونه که توی `and` اگر اولی درست نشه، دومی هم چک نمی‌کنه.

حالا پایتون ایندکس منفی داریم و ارور نمی‌خورین ولی خیلی از زبونا نداریم. اونجا به مشکل می‌خورین.

## • Selection Sort<sup>67</sup>

قضیه `selection sort` اینه که ما از ایندکس ۰ شروع می‌کنیم و میریم تا آخر و مینیموم رو پیدا می‌کنیم و جای ایندکس صفر رو با مینیموم عوض می‌کنیم. بعدش میریم ایندکس ۱ و میریم باز تا آخر و مینیموم رو پیدا می‌کنیم و جاش رو با ایندکس ۱ عوض می‌کنیم. یعنی هی داریم کوچکتترین رو میاریم اول.

67 More info: <https://www.geeksforgeeks.org/selection-sort/>

مثال (در هر مرحله مینیمومی که از مرحله قبل پیدا شده، با رنگ آبی هایلایت شده).  $\text{min\_i}$  هم ایندکس عنصر مینیموم هست:

Input:

12 15 7 9 2

First time ( $i = 0$ ):

12 15 7 9 2  $\rightarrow$  Compare 12 and 15  $\rightarrow$  12 is less  $\rightarrow \text{min\_i} = 0$

12 15 7 9 2  $\rightarrow$  Compare 12 and 7  $\rightarrow$  7 is less  $\rightarrow \text{min\_i} = 2$

12 15 7 9 2  $\rightarrow$  Compare 7 and 9  $\rightarrow$  7 is less  $\rightarrow \text{min\_i} = 2$

12 15 7 9 2  $\rightarrow$  Compare 7 and 2  $\rightarrow$  2 is less  $\rightarrow \text{min\_i} = 4$

12 15 7 9 2  $\rightarrow$  Reach to the end. Swap( $l[0]$ ,  $l[\text{min\_i}]$ )

2 15 7 9 12

Second time ( $i = 1$ ):

2 15 7 9 12  $\rightarrow$  Compare 15 with 7  $\rightarrow$  7 is less  $\rightarrow \text{min\_i} = 2$

2 15 7 9 12  $\rightarrow$  Compare 7 with 9  $\rightarrow$  7 is less  $\rightarrow \text{min\_i} = 2$

2 15 7 9 12  $\rightarrow$  Compare 7 with 12  $\rightarrow$  7 is less  $\rightarrow \text{min\_i} = 2$

2 15 7 9 12  $\rightarrow$  Reach to the end. Swap( $l[1]$ ,  $l[\text{min\_i}]$ )

2 7 15 9 12

Third time ( $i = 2$ ):

2 7 15 9 12  $\rightarrow$  Compare 15 with 9  $\rightarrow$  9 is less  $\rightarrow \text{min\_i} = 3$

2 7 15 9 12  $\rightarrow$  Compare 15 with 9  $\rightarrow$  9 is less  $\rightarrow \text{min\_i} = 3$

2 7 15 9 12  $\rightarrow$  Reach to the end. Swap( $l[2]$ ,  $l[\text{min\_i}]$ )

2 7 9 15 12

Fourth time ( $i = 3$ ):

2 7 9 15 12  $\rightarrow$  Compare 15 with 12  $\rightarrow$  12 is less  $\rightarrow \text{min\_i} = 4$

2 7 9 15 12  $\rightarrow$  Reach to the end. Swap( $l[3]$ ,  $l[\text{min\_i}]$ )

2 7 9 12 15

خب آیا نیازه برای  $i = 4$  هم بریم؟ نه! چون آخری با چی نیازه چک شه؟ هیچی! همه چی درسته!  
خب حالا کدش بزنیم!

پاسخ:

```
def selection_sort(l):  
    length = len(l)
```

```

min_i = 0
for i in range(length):
    min_i = i
    j = i + 1
    while j < length:
        if l[j] < l[min_i]:
            min_i = j
        j += 1
    l[min_i], l[i] = l[i], l[min_i]
return l

```

```

mylist = [64, 25, 12, 22, 11]
print(selection_sort(mylist))

```

هر بار چک میشه که مینیموم پیدا شه و اگر کوچکتر شه، min\_i عوض میشه.

## • More List Comprehensions

یه مقداری بیشتر وارد for درون List شیم:

```

l1 = ["watermelon", "apple", "banana", "orange"]
l2 = [fruit for fruit in l1 if fruit != "banana"]
print(l2)

```

میگه یه سری fruit (اسم متغیر بامعنی انتخاب کردم) بذار توی لیست. کدوما؟ اونایی که توی l1 هستن ولی چک کن ببین برابر «banana» نباشه.

**تمرین!** همش با list comprehension انجام بدین!

۱- یه لیست شامل اعداد اول کوچکتر از ۱۰۰.

۲- به تعداد عناصر list1، توی list2 کلمه «Python» رو بذارین.

۳- روی اعداد ۱ تا قبل ۱۰ حرکت کنین و اگر عدد زوج بود، خود i رو بذارین توی لیست وگرنه، صفر رو بذارین.

۴- یه ماتریس ۳ در ۳ رو بکشین. بعدش سعی کنین ماتریس زیر رو بسازین:

```

matrix = [[1, 2, 3],
           [4, 5, 6],
           [7, 8, 9]]

```

۴- یه مثال جالب که توی لینک زیره (بعد مبحث بعدی که any و all هست بخونیدش):

<https://stackoverflow.com/questions/2522503/advanced-python-list-comprehension>

## پاسفنامه:

-۱

```
prime_list = [num for num in range(2, 100) if is_prime(num)]
```

میگه num رو بذار توی لیست. خب این num ها از کجا میان؟ از ۲ تا قبل ۱۰۰. خب اما یه شرط هم باید برقرار باشه. اگر `is_prime(num)` مقدارش True شه.

-۲

```
list1 = ["watermelon", "apple", "banana", "orange"]  
lsit2 = ['Python' for i in range(len(list1))]
```

حواستون هست که زبون پایتون case sensitive هست! من گفتم «Python» رو چاپ کنین! پس حواستون باشه که «P» تون بزرگ باشه!

-۳

```
l = [i if i % 2 == 0 else 0 for i in range(1, 10)]
```

عه اینو نگفته بودی! else رو نگفته بودی توش!  
آره نگفته بودم ولی می خواستم کم کم روی پای خودتون بایستین و با امتحان و خطا syntax (نحوه نوشتار) رو پیدا کنین.  
میگه i رو بذار توی لیست، اگر rrr i زوج بود. اگر نه ۰ رو بذار. i ها هم در رنج ۱ تا ۱۰ هستن.  
- باز متوجه نشدم!  
+ شاید اگر اینطوری بنویسمش بهتر متوجه شین:

```
l = [  
    if i % 2 == 0  
    else 0  
    for i in range(1, 10)]
```

-۴

همیشه سعی کنین مسأله‌های سخت رو بشکونین به مسئله‌های کوچکتر. کاری که من اینجا می‌کنم اینه که از دید سطح بالا شروع می‌کنم و عناصر اصلی رو می‌سازم و بعدش جزئیات رو کامل می‌کنم. یهو نمی‌پریم توی جزئیات!  
خب قبول دارین که ماتریکس، یه لیست دوبعدیه؟ خب باید بگیم یه لیست می‌خوام که توش یه سری لیستن:

```
mat = [[]]
```

حالا مگه نباید تعداد تکرار درونی ۳ تا بشه؟ خب می‌گیم این لیستای خالی رو ۳ بار تکرار کن:

```
mat = [[] for j in range(3)]
```

```
mat: [[], [], []]
```

حالا توی این لیستای درونی چی باشه؟ خب ما باید ۳ عدد قرار بدیم (چون ماتریس ۳ در ۳ هست):

```
mat = [[i for i in range(3)] for j in range(3)]
```

یا تمیزتر بنویسیم:

```
mat = [[i for i in range(3)]  
        for j in range(3)]
```

```
mat: [[0, 1, 2], [0, 1, 2], [0, 1, 2]]
```

توضیح:

۱- من یه لیست می‌خوام.

۲- این لیست باید یه سری لیست رو درون خودش جا بده. این لیست‌های درونی ۳ بار باید تکرار شن.

۳- خب داخل لیست درونی چی هست؟ یه سری *i* که از ۰ تا ۲ وجود دارن.

خب حالا بیایم یکم بهترش کنیم. می‌خوایم ماتریس زیباتری بسازیم که لیستای درونی متفاوت شن.

برای همین از *j* کمک می‌گیرم که بتونم لیست درونی رو متغیر کنم و ثابت نباشه:

```
mat = [[(i + j) for i in range(3)]  
        for j in range(3)]
```

```
mat: [[0, 1, 2], [1, 2, 3], [2, 3, 4]]
```

خب دیدین؟ بهتر شد! حالا اینجا به بعد باید بازی کنیم باهاش که اون ماتریسی که سؤال گفته ساخته

شه. من خودمم نشستم یکم ور رفتم باهاش تا تونستم بسازمش:

```
mat = [[(i + j) for i in range(1, 4)]  
        for j in range(0, 7, 3)]
```

```
mat: [[1, 2, 3],  
       [4, 5, 6],  
       [7, 8, 9]]
```

## • Any() all()

گاهی ما مثلاً یه `for` می‌زنیم و حاصل یه تابع رو می‌ریزیم توش. یعنی مثلاً ۵ بار `for` اجرا میشه و ۵

خروجی تابع ریخته میشه توش. حالا می‌خوایم ببینیم که مثلاً آیا توش `True` وجود داره یا نه؟

به این درد می‌خوره که ببینیم مثلاً اگر حتی یه `True` بود، چیزی رو چاپ کنیم.

میگیم:

```
l = [True, False, True, False]
if any(l):
    print('yes')
```

all می‌گه اگر همش True بود، چاپ کنه:

```
l = [True, False, True, False]
if all(l):
    print('yes')
```

این به درد مسئله‌ها می‌خوره که خروجی رو به جا ذخیره می‌کنیم و در صورت درست بودن حتی یکیشون، به چیزی یا به کاری رو انجام بده.  
این جدول درستیش (Truth Table):

<https://www.geeksforgeeks.org/any-all-in-python/>

مثال ۱:

```
l = [-1]
if all(l) >= 3:
    print('Yes')
```

به نظرتون چرا این «Yes» رو چاپ می‌کنه؟ عدد داخلش که ۱- هست! عناصر بزرگ‌تر یا مساوی ۳ هم نیستن که! چرا پس Yes چاپ میشه؟  
راهنمایی: all رو به شکل یه تابع ببینین. میره اول نگاه می‌کنه که آیا حداقل یه مقدار لیست ما، True هست؟ اگر هست به جای all(l) مقدار True قرار می‌گیره.  
همونطور که می‌دونیم، تنها مقدار عدد صفر False هست و هر عدد دیگه‌ای (چه منفی چه مثبت)، True هست. پس اینطوری میشه:

```
True >= 3
```

عدد ۳ هم باید تبدیل به boolean شه و درواقع به جاش boolean و True قرار می‌گیره. پس درواقع اینطور میشه:

```
if True >= True:
```

آیا True بزرگ‌تر یا مساوی True هست؟ بله! پس چاپ میشه!  
چه موقع چاپ نمیشه؟ زمانی که درواقع توی لیست ما همه چی False باشه. مثلاً عدد صفر که به صورت پیشفرض وقتی به boolean تبدیل میشه، False هست.

```
l = [0]
if all(l) >= 1:
    print('Yes')
```

هیچی توش True نیست پس False ریترن میشه. عدد یک هم True هست. پس درواقع اینطوری میشه:

```
if False >= True:
```

نکته! توی String ها و List، مقدار خالی False هست و هرچی دیگه True.

## مثال ۲:

یه لیست شامل یه سری آدم‌ها. حالا اگر یه دونه اسم هم اسمش amir بود، چاپ کنه Hi.

```
l = ['james', 'jack', 'hannah', 'amir']
if any(name == 'amir' for name in l):
    print('Hi')
```

## • لیست دو بعدی!

همونطور که گفتیم، یه لیست می‌تونه شامل چند نوع متغیر باشه. (حتی یه متغیر به نام num تعریف کنین و اون رو توش قرار بدین!) حتی یه لیست رو توی خودش نگه داره!

```
mylist = [1.28, 'John', -256, ['car', 'bus', 152], False]
```

خب حالا چه جور به عناصر دسترسی پیدا کنیم؟ مثلاً string بود که index داشت، لیست هم index داره. یعنی عنصر اول `index = 0`. فانکشن `len` هم می‌تونین روش صدا بزنین که طول رو میده بهتون.

مثلاً `len` برای مثال بالا، بهمون ۵ رو میده. چون ۵ عنصر توشه.

- خب یکیش لیسته که توش ۳ تا عنصر هست. چرا نمیره داخلش رو بشمره؟!

+ آره نمیره! صرفاً می‌گه ۵ تا عنصر دارم. اولیش `float`، دومیش `string`، سومیش `integer`، چهارمی یه لیست، پنجمی یه `boolean`. اگر می‌خوانین سائز لیست درونی رو بدونین، باید بگین `len` اون ایندکس خاص رو می‌خوام. (ایندکسش ۳ هست) مثلاً:

```
mylist = [1.28, 'John', -256, ['car', 'bus', 152], False]
print(len(mylist[3]))
```

مثلاً اگر بخوایم به ۱۵۲ دسترسی داشته باشیم:

```
mylist = [1.28, 'John', -256, ['car', 'bus', 152], False]
print(mylist[3][2])
```

یعنی اول برو ایندکس سوم. بعد برو ایندکس دومش.

اصطلاحاً به اینا می‌گن لیست دو بعدی!

تازه مثلاً اگر بخوایم به «۲» توی لیست درونی دسترسی داشته باشیم:

```
mylist = [1.28, 'John', -256, ['car', 'bus', 152], False]
print(mylist[3][0][2])
```

- این به چه درد می‌خوره؟

+ فرض کنین می‌خواهیم یه ماتریس بسازیم:

```
mylist = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```



یا همونطور که قبلاً گفتیم، می‌تونیم بعد کاما، یه اینتر بزنینم که تمیزتر شه:

```
mylist = [[1, 2, 3],  
          [4, 5, 6],  
          [7, 8, 9]]
```

حالا تمیزتر شد و دقیقاً شبیه ماتریس شد.

نظرتون چیه با `while` همین `list` رو بسازیم؟

```
matrix3 = []  
i = 1  
while i < 10:  
    matrix3.append([i, i + 1, i + 2])  
    i += 3  
print(matrix3)
```

خب اگر بخوایم از خود کاربر بگیریم چی؟

خط به خط می‌گیریم و هر خط رو یه `split` به لیست تبدیل می‌کنیم و لیست رو توی یه لیست دیگه می‌ریزیم. دقیقاً مثل شکل بالا:

```
row1 = input().split()  
row2 = input().split()  
row3 = input().split()  
matrix = [row1, row2, row3]  
print(matrix)
```

- عه مگه میشه `input` رو همونجا `split` کنی؟

+ بله! درواقع میگی همون ورودی که توی صف هستا که داره میاد، همونو `split` کن بریز توی `row1`. در آخر هم توی ماتریس قرارش دادیم.

خب نظرتون چیه یه ماتریس‌طور با `list comprehension` بسازیم؟

```
matrix3 = [[i for i in range(1, 4)] for j in range(3)]  
print(matrix3)
```

میگیم یه سری لیست (لیست بنفش) توی لیست اصلی ما باشن. ۳ بار (`for j`...) این لیستا رو بذار. حالا چه لیستی؟ لیستی که عناصرش از ۱ تا قبل ۴ هستن. یعنی:

```
[[1, 2, 3], [1, 2, 3], [1, 2, 3]]
```

یکم با این بازی کنین و لیست‌های تو در تو بسازین.

مثال:

```
matrix3 = [[i+j for i in range(1, 4)] for j in range(3)]
```

سه بار یه لیست رو بذار توی لیستم.

اون لیستایی که باید بذاره اینطورین:

$i + j$  رو برای  $i$  های از رنج ۱ تا قبل ۴ بذار. بار اول چون  $j$  برابر صفره، پس میشه  $i + 0$  که خود  $i$  هم از ۱ تا قبل ۴ میره. بار بعدی  $j$  برابر ۱. بار بعدی برابر ۲.

```
[i + 0, i + 0, i + 0],  
[i + 1, i + 1, i + 1],  
[i + 2, i + 2, i + 2]
```

پس یعنی چاپ می‌کنه:

```
[1, 2, 3],  
[2, 3, 4],  
[3, 4, 5]
```

قسمت کاپی گرفتن یه `list` رو یادتونه؟ حالا اگر لیست دوبعدی باشه باز نمی‌تونیم از کاپی عادی استفاده کنیم. چون `copy` عادی فقط یه بعد و لایه رو کاپی می‌کنه. اگر لایه درونی‌تر رو تغییر بدیم، باز قبلی هم تغییر می‌کنه:

```
l1 = [[1, 2, 3],  
      [2, 3, 4],  
      [3, 4, 5]]  
l1_copy = l1.copy()  
l1_copy[0][0] = 'a'  
print(f'l1_copy: {l1_copy}')  
print(f'l1: {l1}')
```

پس چیکار کنیم؟

اینجا یه لایبرری هست که گفته نیاز نیست تو کاری کنی! من کدشو زدم! شما صرفاً از تابعی که من قبلاً ساختم استفاده کن! اسم لایبرری «`copy`» هست.

```
import copy  
  
l1 = [[1, 2, 3],  
      [2, 3, 4],  
      [3, 4, 5]]  
l1_copy = copy.deepcopy(l1)  
l1_copy[0][0] = 'a'  
print(f'l1_copy: {l1_copy}')  
print(f'l1: {l1}')
```

درواقع می‌گیم که از `copy`، برو `deepcopy` رو صدا بزن و بهش `l1` رو پاس میدیم که یه کاپی عمیق (`deep copy`) انجام بده.

شاید براتون سخت باشه که هر دفعه بگین از `copy` برو `deepcopy` رو صدا بزن. (بینش نقطه می‌گذاریم). خب می‌تونین صرفاً بگین که اون تابعی هست که توی `copy` هست که اسمش `deep copy` هست! برو صرفاً اون رو اضافه کن به کدم:

```
from copy import deepcopy
```

```
l1 = [[1, 2, 3],  
      [2, 3, 4],  
      [3, 4, 5]]  
l1_copy = deepcopy(l1)  
l1_copy[0][0] = 'a'  
print(f'l1_copy: {l1_copy}')  
print(f'l1: {l1}')
```

وقتی صداش بخوایم بزنیم، چون خود تابع اومده و نه کل لایبرری، صرفاً اسم خود تابع رو صدا می‌زنیم.

**نکته:** این کار کد رو هم سریع‌تر می‌کنه تا بخوایم نقطه بذاریم!<sup>۶۸</sup>

حتی می‌تونیم اسم هم بدین بهش. بگین وقتی اضافه‌اش کردی، اسمش مثلاً بذار `deepc`. من هر وقت بخوام صداش بزنم، برام راحت‌تره که بنویسم `deepc`:

```
from copy import deepcopy as deepc
```

```
l1 = [[1, 2, 3],  
      [2, 3, 4],  
      [3, 4, 5]]  
l1_copy = deepc(l1)  
l1_copy[0][0] = 'a'  
print(f'l1_copy: {l1_copy}')  
print(f'l1: {l1}')
```

اینطوری ترجمه میشه: از لایبرری `copy`، برو `deepcopy` رو اضافه کن ولی به اسم `deepc`.

**کدت رو سریع‌تر کن!**

به نظرتون کدوم کد سریع‌تره؟

Code 1:

```
def getSqrt(num):  
    from math import sqrt  
    return sqrt(num)  
  
for i in range(1_000_000_000):  
    getSqrt(i)
```

<sup>۶۸</sup> درواقع این نقطه گذاشتن بازم منجر به dictionary lookup میشه. درواقع هر بار که `math.sqrt` اجرا میشه، پایتون می‌گه خب از `math` چه فانکشنی رو بردارم اجرا کنم؟ عه بذار توی فانکشن‌ها بگردم ببینم `sqrt` کدومه؟ این گشتنه یکم طول می‌کشه.

Code 2:

```
from math import sqrt

def getSqrt(num):
    return sqrt(num)

for i in range(1_000_000_000):
    getSqrt(i)
```

- خب معلومه! کد ۱! چون من یادمه که چیزایی که داخل فانکشن هستن و local هستن، یافتنشون برای استفاده ازشون سریع‌تره! اما کد ۲ هر بار باید sqrt گلوبال رو پیدا کنه که کندتره! + بله یافتن local ها سریع‌تر از global هاست، اما اینجا صرفاً این اتفاق نیوفتاده که بگیم کد ۱ سریع‌تره! خیر! اینجا یه اتفاق مهم‌تر و هزینه‌بردارتر افتاده. اونم اینکه تو کد ۱، هر بار که تابع صدا زده میشه، یه بار import صورت می‌گیره. خود فرایند import کد به کد شما، فرایندی زمان‌بر هست و زمان‌برتر از یافتن و lookup کردن local یا global هست.<sup>۶۹</sup> درواقع درسته کد ۲ داره lookup یافتن تابع sqrt رو توی گلوبال‌ها انجام میده و زمان‌برتره، اما کد ۱ به دلیل اینکه عملیات import زیادی داره انجام میده و خود import کندتر از lookup کردنه، کد ۲ سریع‌تره! کدها رو اجرا کردم و نتیجه:

Code 1: 54 s

Code 2: 9 s

نتیجه: حواستون باشه که اگر for می‌زنین روی یه تابعی که خود اون تابع یه چیزی رو ایمپورت می‌کنه، این باعث میشه که کدتون به شدت کند شه!

تمرین!

تمرین ۱ (ضرب ماتریس‌ها):

<https://quera.org/problemset/607/>

راهنمایی:

وبسایت زیر رو بخونین تا بدونین ضرب ماتریس‌ها چه جور انجام میشه:

<https://blog.faradars.org/how-to-multiply-matrices/>

تمرین ۲ (مثث فیام):

69 Import Statement Overhead → <https://wiki.python.org/moin/PythonSpeed/PerformanceTips>, <https://moinmo.in/GPL>

<https://quera.org/problemset/3410/>

راهنمایی:

برای تبدیل یه لیست به استرینگ، از متد join استفاده کنین. مثلاً:

```
l = ['1', '2', '3']
string = ' '.join(l)
print(string)
```

میگه به وسیله فاصله، اعضای l رو به هم وصل کن. (join کن)  
توجه کنین که اعضا باید string باشه.

بخوایم لیست زیر رو به صورت string پرینت کنیم:

```
l = [['1'], ['1', '2'], ['1', '2', '3']]
for i in range(len(l)):
    print(' '.join(l[i]))
```

میگیم برو هر بار ایندکس i ام (که خودش یه لیست هست)، اجزاشو با فاصله پیوند بده و بذار توی string. یعنی روی لیستای درونی حرکت می‌کنیم.

تمرین ۳:

یه سری لیست مرتب و یه عدد بهتون میدن که باید عدد رو در جای مناسبی از لیست قرار بدین که باز هم لیست مرتب بمونه.

input:

1 2 4

3

output:

1 2 3 4

-----

input:

7 11

12

output:

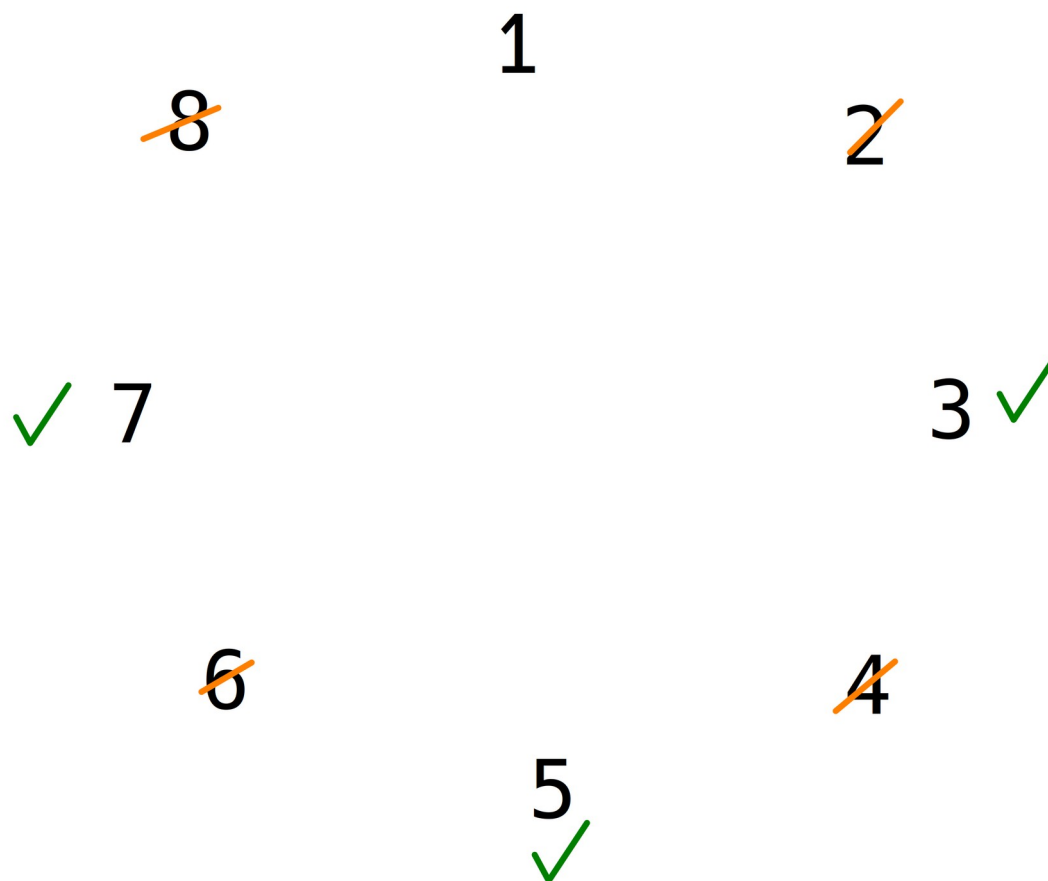
7 11 12

تمرین ۴ (انتقابات دور میز):<sup>۷۰</sup>

<https://quera.org/problemset/604/>

70 Josephus Problem: [https://en.wikipedia.org/wiki/Josephus\\_problem](https://en.wikipedia.org/wiki/Josephus_problem)

مثلاً ۸ نفر دور میز نشستن. بار اول از ۲ شروع میشه و ۲ خط می خوره. بعدش ۴. بعدش ۶ بعدش ۸. یعنی ۳ و ۵ و ۷ بین بودن و نیاز نبوده خط بخورن.



بارو دوم: دفعه قبل ۸ خط خورد. پس از ۸ شروع می کنیم و افراد باقی مونده رو یکی در میون خط می زنیم. ۱ نیاز نیست خط بخوره. پس ۳ خط می خوره. (دوبار خطش زدم که مشخص باشه بار دوم خط خورده و افراد کوررنگ بهتر بتونن تشخیص بدن)  
بعد ۳، ۷ خط می خوره. (۵ خط نمی خوره چون قرار بود یکی در میون از افراد باقی مونده خط بزنینم)

✓✓  
1

~~8~~

~~2~~

~~7~~

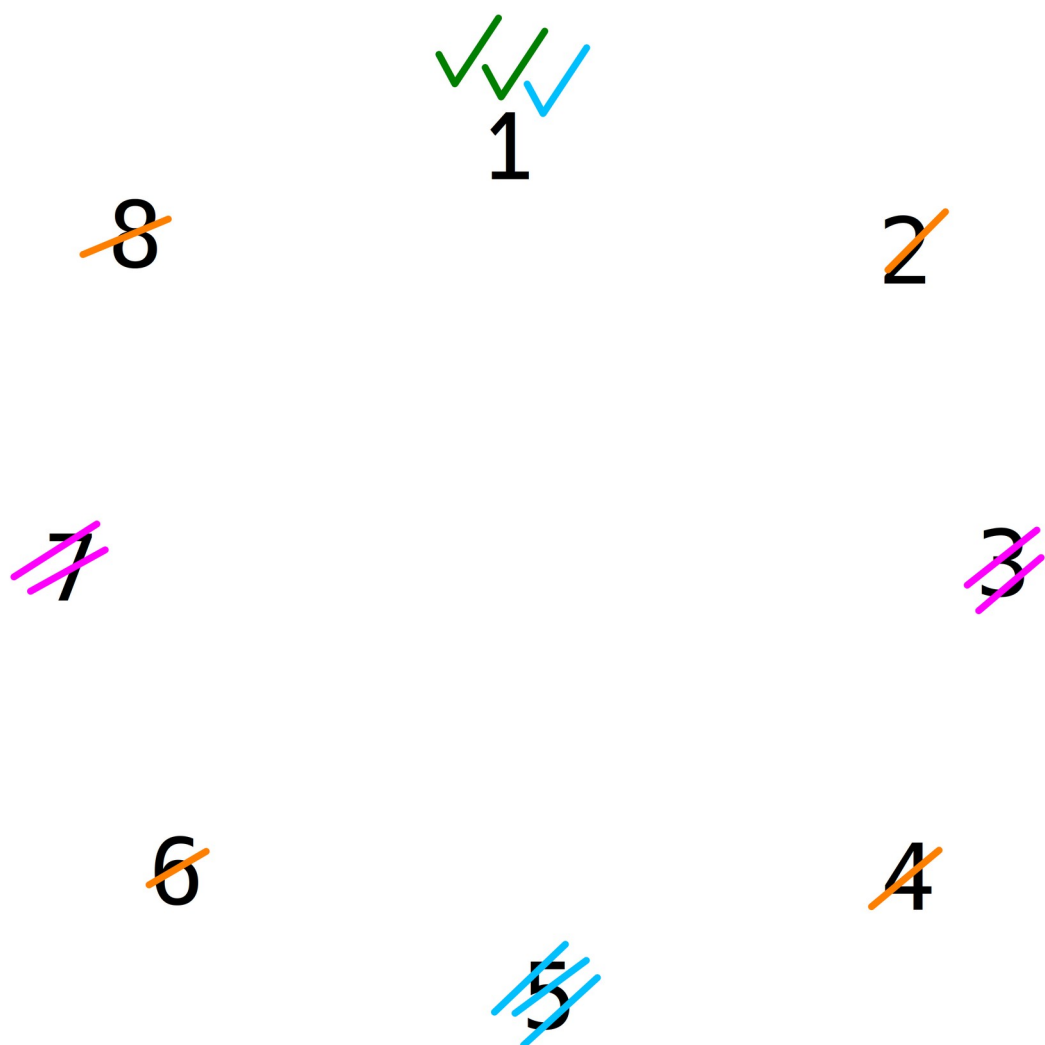
~~3~~

~~6~~

~~4~~

5  
✓✓

بار سوم: صرفاً به ترتیب ۷ و ۱ و ۵ مونده. دفعه قبل ۷ خط خورده بود. پس از ۷ شروع می‌کنیم. ۱ که  
بینشونه خط نمی‌خوره و ۵ خط می‌خوره.



خب باقی مونده ۱ بود. پس ۱ باید چاپ شه.

پاسفنامه:

پاسخ ا:

می دونیم ضرب ماتریس اینطوریه که اول عنصرهای row1 با عناصر col1 نظیر به نظیر ضرب میشن.

و

`res[0][0]`

رو می سازن.

بعدش عناصر row1 با عناصر col2 نظیر به نظیر ضرب میشن و



```
res[0][1]
```

رو می‌سازن.

بعدش میریم سراغ row2 و همین عمل ادامه پیدا می‌کنه.

پس نیاز به اولین for من، روی سطرهای row1 باشه. یه تابع می‌نویسم که ضرب رو انجام بده. چیا نیاز به بگیره؟ قاعدتاً ماتریس اول و دومی و تعداد سطر ۱ و تعداد ستون ۲. چون در ضرب ماتریس نیاز به اینا داریم (اگر همون اول متوجه اینا نشدین، ایراد نداره. کد رو بنویسین و هر جا نیاز به چیزی بود، به لیست متغیرهایی که تابع می‌گیره. اضافه کنین):

```
def mat_product(mat1, mat2, row1, col2):
```

```
    res = []
```

```
    for r1 in range(row1):
```

خب بعدش باید بگیریم که من باید بار اول روی ستون ۰ ماتریس ۲ حرکت می‌کنم. بار دوم روی ستون ۱ ماتریس ۲. همینطور تا آخر. پس:

```
def mat_product(mat1, mat2, row1, col2):
```

```
    res = []
```

```
    for r1 in range(row1):
```

```
        for c2 in range(col2):
```

بعدش حالا باید ضرب رو انجام بدم. که چون ماتریس ما دو بعدی و شکل زیره:

```
[[1, 2, 3],
```

```
 [4, 5, 6]]
```

باید حالا بگم که به ترتیب اینا ضرب میشه:

```
mat1[0][0] * mat2[0][0]
```

```
mat1[0][1] * mat2[1][0]
```

```
mat1[0][2] * mat2[2][0]
```

یعنی اگر دقت کنین ما داریم روی ستون اولی و سطر دومی حرکت می‌کنیم. پس درواقع یه for می‌زنیم روی col1:

```
def mat_product(mat1, mat2, row1, col2):
```

```
    res = []
```

```
    for r1 in range(row1):
```

```
        for c2 in range(col2):
```

```
            for c1 in range(col1): # c1 = r2
```

همونطور که می‌بینین توی کامنت هم نوشتم که ستون ۱، متناظر با سطر ۲ هست.

خب قبول دارین که باید توی for سومی ضرب رو انجام بدیم؟ یعنی قبول دارین که برای هر مرحله که وارد for سومی میشه، قبلش باید یه متغیر در نظر بگیریم که سه تا جمع می‌کنیم رو هر مرحله انجام میده با هم جمع کنه؟ یعنی اینطوری:

```
product = 0
```

```
product += mat1[0][0] * mat2[0][0]
```

```
product += mat1[0][1] * mat2[1][0]
```

```
product += mat1[0][2] * mat2[2][0]
```

خب پس قبل از اینکه هر بار وارد for شم متغیر رو تعیین می‌کنم:

```
def mat_product(mat1, mat2, row1, col2):  
    res = []  
    for r1 in range(row1):  
        for c2 in range(col2):  
            product = 0  
            for c1 in range(col1): # c1 = r2  
                product += mat1[r1][c1] * mat2[c1][c2]
```

خب قبول داریم که بعد for سومی، من یه قسمت از ردیف res رو درست کردم؟ یعنی مثلاً row[0] رو درست کردم. اما کل ردیف رو نساختم. کل ردیف زمانی ساخته میشه که for دومی تموم شه! پس من نیازه که قبل از ورود به for دومی، یه متغیر بسازم به عنوان ردیف که هر بار که for دومی اجرا میشه و یه قسمت از ردیف ساخته میشه، اون عدد به ردیف اضافه شه: اسمش میذارم rowl که نشون بده یه لیست هست.

```
def mat_product(mat1, mat2, row1, col2):  
    res = []  
    for r1 in range(row1):  
        rowl = []  
        for c2 in range(col2):  
            product = 0  
            for c1 in range(col1): # c1 = r2  
                product += mat1[r1][c1] * mat2[c1][c2]  
  
        rowl.append(product)
```

وقتی for سومی تموم شد، باید اون لیست رو اضافه کنم به rowl. حالا دوباره میره بالا توی for دومی و باز میاد پایین و توی for سومی میاد ستون دوم row ما رو می‌سازه و همینطور ادامه پیدا می‌کنه.

حالا قبول داریم که وقتی که for دوم تموم شد و پرید بیرون، درواقع یه ردیف ما ساخته شده؟ یعنی ردیف اول res ما ساخته شده. پس نیازه که این ردیف (که یه لیست هست رو) به res اضافه کنم:

```
def mat_product(mat1, mat2, row1, col2):  
    res = []  
    for r1 in range(row1):  
        rowl = []  
        for c2 in range(col2):  
            product = 0
```

```

        for c1 in range(col1): # c1 = r2
            product += mat1[r1][c1] * mat2[c1][c2]

    row1.append(product)

res.append(row1)
return res

```

یعنی درواقع for سوم ما هر بار یه بار ضربا رو انجام میدی و جمع می‌کنی و یه عدد می‌سازی.

For دوم ما میاد هر بار یکی از این عدد رو به لیست ردیف اضافه می‌کنی تا ردیف تکمیل شه:

```

[] # Before appending product
[22] # After appending first product series
[22, 28] # After appending second product series

```

حالا که چند بار داخل for دوم ما چرخید و ردیف ساخته شد، از for دوم می‌پره بیرون و میاد توی for اول. حالا توی for اول هر بار این ردیف‌های ساخته شده، به لیست نهایی res ما append میشن (یه ردیف که به صورت لیسته، به لیست نهایی اضافه میشه):

```

[] # Before appending

[[22, 28]] # After appending first row

[[22, 28],
 [49, 64]] # After appending second row

```

حالا بیایم کد برنامه رو تکمیل کنیم و ورودی‌ها رو بگیریم و خروجی کنیم:

```

def mat_product(mat1, mat2, row1, col2):
    res = []
    for r1 in range(row1):
        row1 = []
        for c2 in range(col2):
            product = 0
            for c1 in range(col1): # c1 = r2
                product += mat1[r1][c1] * mat2[c1][c2]

        row1.append(product)

    res.append(row1)

return res

```

```

line1_inp = input().split()
row1 = int(line1_inp[0])
col1 = int(line1_inp[1])
row2 = col1
col2 = int(line1_inp[2])

mat1 = []
mat2 = []

for i in range(row1): # Get matrix 1
    row1 = input().split()
    row1 = [int(i) for i in row1]
    mat1.append(row1)

for i in range(row2): # Get matrix 2
    row1 = input().split()
    row1 = [int(i) for i in row1]
    mat2.append(row1)

product = mat_product(mat1, mat2, row1, col2)

for i in range(len(product)):
    for j in range(len(product[i])):
        print(product[i][j], end='')
        if j != len(product[i]): # if not last the line, print space
            print(' ', end='')
    print()

```

موقع پرینت یکم نیاز به فکر کنیم تا دقیقاً طبق فرمت judge پرینتش کنیم. یعنی تا وقتی که هر ردیف پرینت میشه، باید بین اعداد فاصله باشه و بعد هر ردیف یه `enter`. من اینو اینطور انجام دادم که هر بار که عددی چاپ میشه، `end` (بعد پرینت)، چیزی چاپ نکنه. نه فاصله و نه `enter`. بعدش چک می‌کنم که اگر هنوز به ته ستون نرسیدم (یعنی هنوز عناصر دیگه‌ای درون ستون وجود دارن)، یه فاصله با `end` خالی چاپ کنه. یعنی بعد فاصله اینتر نزنه. (اگر `end` نمی‌گذاشتم، بعد هر پرینت مثل همیشه خودش یه اینتر می‌زد)

در واقع این کار از اینکه بعد چاپ کردن ستون آخر ما، به فاصله اضافه چاپ شه جلوگیری می‌کنه. کاری که اگر کد رو اینطور می‌نوشتیم پیش می‌ومد:

```
for i in range(len(product)):
    for j in range(len(product[i])):
        print(product[i][j], end=' ')
    print()
```

چون اینطوری فارغ از اینکه ستون آخری چاپ شده یا نشده، به فاصله میزد الکی. ولی خب ستون آخر که چاپ شد، نیاز به فاصله نیست! بلکه نیاز به `enter` بزنیم. در پایان هر باری که `for` داخلی اجرا میشه، به ردیف چاپ میشه. پس نیاز به برم خط بعدی. پس به `print` چاپ می‌کنم. (که به صورت پیشفرض `enter` میزنه).

کد رو می‌تونستیم بهینه‌تر هم کنیم. به این صورت که در `for` آخری، نیازی به محاسبه تعداد ردیف و ستون `result` نبود! اگر دانش ریاضی داشته باشیم، می‌دونیم که تعداد ردیف `result`، برابر تعداد ردیف ماتریس اول.

تعداد ستون هم برابر تعداد ستون دومی هست.

البته بهتره توی کامنت این رو بنویسیم که کسی که کد رو می‌خونه بهتر متوجه شه. یعنی:

```
# Print result of product
# Result row = row1, Result col = col2
for i in range(row1):
    for j in range(len(col2)):
        print(product[i][j], end=' ')
    print()
```

پاسخ ۲:

```
def khayam_triangle(n):
    l = []
    if n == 1:
        return [[1]]
    if n == 2:
        return [[1, 1]]

    l = [[1], [1, 1]]
    n -= 2
    while n != 0:
        length = len(l)
        newl = [1] # each time we need an 1 at the left.
```

```

        # -1 because we don't need to check the last one.
        for i in range(len(l) - 1):
            # append sum of two numbers next to each other.
            newl.append(l[length-1][i] + l[length-1][i + 1])

        newl.append(1) # at the end of all rows, there is an 1.
        l.append(newl)
        n -= 1

    return l

```

```

row_count = int(input())
khayam_triangle_list = khayam_triangle(row_count)
for row in khayam_triangle_list:
    print(' '.join([str(i) for i in row]))

```

من هر خط رو توی یه لیست می‌سازم و در آخر لیست رو اضافه می‌کنم به لیست اصلی.  
در آخر باید اعضا اول string شن - که با cast کردن انجام دادم-، بعد باید join بشون کنم که پرینت بشون کنم.

### پاسخ ۳:

باید روی اعضای لیست دومی که می‌خواد اضافه شه، حرکت کنیم. و دونه دونه به لیست اولی اضافه کنیم.  
خب برای اضافه کردن هر عنصر، باید دونه دونه روی اعضای لیست دومی حرکت کنیم و بگیریم خب کجا از عنصر لیست اول کوچکتر یا مساویه که اضافه کنیم به قبل عنصر لیست. یعنی درواقع لیست جدید، از ترکیب عنصرای قبلی + عنصر جدید + ادامه لیست تشکیل میشه:

```

mainl = input().split()
numbersl = input().split()
mainl = [int(i) for i in mainl]
numbersl = [int(i) for i in numbersl]

for i in range(len(numbersl)):
    for j in range(len(mainl)):
        if numbersl[i] <= mainl[j]:
            mainl = mainl[:j] + [numbersl[i]] + mainl[j:]
            break

```

وقتی اضافه‌اش کردیم، کارمون تمومه! پس break می‌کنیم که الکی نره جلوتر.

اما این یه مشکل داره. یه سری تست کیس بنویسین ببینین کجا به مشکل بر می‌خورین! مثلاً:

input:

1 2 3

4

output:

1 2 3 4

-----

input:

1 2 3

0

output:

0 1 2 3

-----

input:

1 2 3

3

output:

1 2 3 3

-----

input:

1 2 3

1

output:

1 1 2 3

-----

input:

1 2 3

2

output:

1 2 2 3

خب مشکل رو یافتین؟ مشکل اینجاست که من میگم اگر کوچکتر یا مساوی بود، اضافه کن. اما اگر عنصر بزرگ‌تر از عنصر آخر باشه، باید به آخر اضافه شه. اما اینجا اضافه نمیشه. (اصلاً شرطی نیست که چکش کنه!) پس کدمون رو درست کنیم:

```

mainl = input().split()
numbersl = input().split()
mainl = [int(i) for i in mainl]
numbersl = [int(i) for i in numbersl]

for i in range(len(numbersl)):
    for j in range(len(mainl)):
        if numbersl[i] <= mainl[j]:
            mainl = mainl[:j] + [numbersl[i]] + mainl[j:]
            break
    # if the number is bigger than all the numbers in mainl
    elif j == len(mainl)-1:
        mainl = mainl + [numbersl[i]]
        break

mainl = [str(i) for i in mainl]
print(' '.join(mainl))

```

میشه گفت که لیست مهم‌ترین data type (نوع داده؛ چیزایی مثل string و integer و...) هست.

سعی کنین خیلی دربارش تمرین حل کنین. درواقع توی زبون‌های دیگه حتی ممکنه string نداشته باشین! بله! درست شنیدین! به جاش یه لیست شامل یه سری کرکتر دارین که پشت هم قرار گرفتن.<sup>۷۱</sup> یعنی:

```

s = 'hello'
l = ['h', 'e', 'l', 'l', 'o']

```

پس خیلی مهمه که به خوبی یادش بگیرین!

### پاسخ ۴:

یه راه اینه که من پیام بگم هر بار اونایی که خط می‌خورن رو از لیست بندازم بیرون و هر بار لیستی جدید پاس بدم بهش. این به نظرم سخته. چون هر بار لیست سایشش تغییر می‌کنه. هر بار باید حواسم باشه دفعه قبلی چی خط خورده و ایندکس بعدیش چیه؟ کجاها خط خوردن و بعدیش چیه و کلی مشکل دیگه!

۷۱ در اصل بهشون میگن Array (آرایه)



راهی که من استفاده کردم اینه که میگم یه لیست دارم و هر بار که کسی خط خورد، جاش «'0'» می‌گذارم و در آخر هم یه عنصر که «'0'» نیست باقی می‌مونه و صرفاً همونو ریترن می‌کنم. اینطوری بهتر می‌تونم روند رو دنبال کنم.

بار اول همه زوجا خط می‌خورن قبول دارین؟ پس من تابع رو با این شروع می‌کنم که به صورت پیشفرض همه زوجا خط بخورن.

```
def find_last(l, length):
    # change even elements (odd indexes) to '0'
    l = ['0' if i % 2 == 1 else l[i] for i in range(length)]
```

یعنی میگم کرکتر string صفر رو بذار. اگر ایندکس فرد بود (عنصر زوج) وگرنه، خود  $l[i]$  رو بذار. یعنی درواقع اومدم ایندکسای فرد (عناصر زوج) رو با «'0'» عوض کردم.

این دور اول خط زدن بود. حالا بریم سراغ ادامهش.

قبول دارین که باید یادم باشه دفعه قبل کدوم عنصر خط خورد؟

حالا من با نوشتن ۲ تا ۷ و ۲ تا ۸ (تست کیسی که آخری زوج باشه و آخری فرد)، دیدم که آخرین ایندکسی که برای زوجا خط می‌خوره، برابر  $len - 1$  و برای فردا،  $len - 2$  هست:

```
1 2 3 4 5 6 7 → last one i = 5 → i = len - 2
1 2 3 4 5 6 7 8 → last one i = 7 → i = len - 1
```

حالا پس برای اینکه بدونم آخری چی بوده، اونو توی یه متغیر ذخیره می‌کنم:

```
def find_last(l, length):
    # change even elements (odd indexes) to '0'
    l = ['0' if i % 2 == 1 else l[i] for i in range(length)]
```

```
    if length % 2 == 0:
        i = length - 1
    else:
        i = length - 2
```

خب تا اینجا همه چیز داره اوکی پیش میره.

خب روند رو باید چه جور پیش ببریم؟ قبول دارین هی باید خط بزنینم تا فقط یک عنصر باقی بمونه؟ یا درواقع صرفاً یک عنصر که «'0'» نیست باقی‌بمونه. همه عناصر لیست ما بشن «'0'» به جز یکیش. یعنی اینو اینطور می‌تونیم بنویسیم:

```
def find_last(l, length):
    # change even elements (odd indexes) to '0'
    l = ['0' if i % 2 == 1 else l[i] for i in range(length)]

    if length % 2 == 0:
        i = length - 1
```

```
else:
```

```
    i = length - 2
```

```
while l.count('0') != length - 1:
```

یعنی درواقع تا وقتی که تعداد صفرا برابر یکی کمتر از تعداد کل نشده، پیش برو و خط بزن. حالا بریم خط بزنیم!

من اینطوری فکر کردم که چون برای بار اول که یکی در میون پیش میرفت و چیزی خط نخورده بود،  $i = 2$  بود و هی ۲ تا دوتا اضافه می کردیم و خط می زدیم که یکی در میون خط بخوره. اما از بار دوم به بعد، یه سری چیزا خط خورده و نمی تونیم بگیریم دوتا دوتا برو جلو و خط بزن! چون ممکنه دوتا بره جلو و اونم خط خورده باشه.

پس فکر درست تر اینه که بگیریم از اینجایی که هستی، برو جلو و دونه دونه چک کن. هر جا «0» نبود (عادی بود)، اولیشو بذار کنار و کاریش نداشته باش. برو دومی که «0» نیست رو پیدا کن و دومی رو خط بزن.

این رو اینطوری تعریف می کنم که یه فلگ قرار میدم و میگم هی برو جلو و هی ایندکس اضافه کن. تا وقتی که که ببینی «0» نیست و همچنین ببینی که flag ما عوض نشده. اولین چیزی که «0» نبود، flag رو عوض کن. حالا یکی از شروط برقرار شد. یعنی flag عوض شد. پس حالا نیازه اولین چیزی که «0» نبود رو دیدی، بپری بیرون و عوضش کنی به «0».

این رو اینطوری می نویسم:

```
def find_last(l, length):
    # change even elements (odd indexes) to '0'
    l = ['0' if i % 2 == 1 else l[i] for i in range(length)]

    if length % 2 == 0:
        i = length - 1
    else:
        i = length - 2

    while l.count('0') != length - 1:
        i += 1
        first_finded = False
        while first_finded == False or l[i] == '0':
            if l[i] != '0':
                first_finded = True
        i += 1
```

یعنی میگم اول از آخری که خط خورد یه ایندکس برو جلو. بعد یه فلگ به نام first\_finded رو بساز و چون هنوز اولی پیدا نشده، False اش کن. بعدش تا وقتی که اولی پیدا نشده و هنوز «0» هست برو جلو. اول چک می کنم که اگر برابر «0» نبود، یعنی اولی پیدا شد. حالا باز دونه دونه ایندکس میریم

جلو تا وقتی که به اولین چیزی که «'0'» نیست برسیم و خب از while می‌پریم بیرون تا اون رو خط بزنیم:

```
def find_last(l, length):  
    # change even elements (odd indexes) to '0'  
    l = ['0' if i % 2 == 1 else l[i] for i in range(length)]  
  
    if length % 2 == 0:  
        i = length - 1  
    else:  
        i = length - 2  
  
    while l.count('0') != length - 1:  
        i += 1  
        first_finded = False  
        while first_finded == False or l[i] == '0':  
            if l[i] != '0':  
                first_finded = True  
            i += 1  
  
    l[i] = '0'
```

اما یه مشکلی هست! هی میریم ایندکس جلو و هیچ‌وقت چک نمی‌کنیم که آیا out of range میشه یا نه؟!

جاهایی که ایندکس اضافه می‌کنیم، باید یه چک بشه ببینیم out of range نشه یه وقت!

```
def find_last(l, length):  
    # change even elements (odd indexes) to '0'  
    l = ['0' if i % 2 == 1 else l[i] for i in range(length)]  
  
    if length % 2 == 0:  
        i = length - 1  
    else:  
        i = length - 2  
  
    while l.count('0') != length - 1:  
        i += 1  
        first_finded = False  
        if i >= length:  
            i -= length  
        while first_finded == False or l[i] == '0':  
            if l[i] != '0':  
                first_finded = True
```

```

        i += 1
    if i >= length:
        i -= length
    l[i] = '0'

```

خب حالا همه چی مرتبه!

صرفاً نیازه اون عنصری در لیست که خط نخورده (یعنی برابر «0» نیست) رو ریترن کنم. این کار رو با list comprehension انجام میدم:

```

def find_last(l, length):
    # change even elements (odd indexes) to '0'
    l = ['0' if i % 2 == 1 else l[i] for i in range(length)]

    if length % 2 == 0:
        i = length - 1
    else:
        i = length - 2

    while l.count('0') != length - 1:
        i += 1
        first_finded = False
        if i >= length:
            i -= length
            while first_finded == False or l[i] == '0':
                if l[i] != '0':
                    first_finded = True
            i += 1
        if i >= length:
            i -= length
        l[i] = '0'

    return [element for element in l if element != '0']

```

یعنی میگم یه لیست رو پاس بده. توش element هایی رو بذار. کدوم element ها؟ اونایی که تو لیست هستن و برابر «0» نیستن. یعنی درواقع یه لیست صرفاً شامل همون عنصر باقی‌مونده رو ریترن کن.

حالا برای کامل کردن کد، خارج تابع نیازه تعداد آدم‌ها رو بگیریم و همچنین یه لیست شامل عدد اون آدم‌ها بسازیم. در آخر هم نیازه خروجی‌مون رو چاپ کنیم. چون تابع یه لیست یه عنصری خروجی میده، من می‌تونم بگم خروجی تابع که لیسته (هایلایت صورتی)، ایندکس صفرمشو پرینت کن:

```

people_count = int(input())
l = [i for i in range(1, people_count + 1)]

```

```
print(find_last(1, people_count)[0])
```

تمام!

ببین برنامه‌نویسی سخت نیست. صرفاً نیاز به فکر کنین و راه‌حل پیدا کنین و سعی کنین قدم به قدم راه‌حلتون رو تبدیل به کد کنین. برای همین هم من این مسأله رو قدم به قدم با توضیح براتون توضیح دادم.

+ حل با زبون C (اگر نمی‌دونین، نخوندیش):

ایده: هر بار عنصر مورد نظر رو پیدا می‌کنیم. حذف می‌کنیم. بعد حذف، باید عناصر آرایه، یه شیفت پیدا کنن. یعنی اگر عنصر دوم پاک شد، عنصر سوم به بعد، یه شیفت پیدا کنن و جای خالی عنصر دوم رو پر کنن.

راه ۱ (تتا  $n^2$ )

```
#include <stdio.h>
#include <stdlib.h>

int josephusProblem(int *arr, int n, int pdelete, int k);
int *deleteShift(int *arr, int n, int index);

int main(int argc, char const *argv[])
{
    int n;
    scanf("%d", &n);
    int *arr = (int *)malloc(sizeof(int) * n);
    for (int i = 0; i < n; i++)
    {
        arr[i] = i + 1;
    }

    int res = josephusProblem(arr, n, -1, 2);
    printf("%d", res);
    return 0;
}

int josephusProblem(int *arr, int n, int pdelete, int k)
{
    if (n == 1)
    {
```

```

        return arr[0];
    }

    int *newArr = (int *)malloc(sizeof(int) * (n - 1));
    int newDelete = (pdelete + k) % n;
    newArr = deleteShift(arr, n, newDelete);
    josephusProblem(newArr, n - 1, newDelete - 1, k);
}

int *deleteShift(int *arr, int n, int index)
{
    int *newArr = (int *)malloc(sizeof(int) * (n - 1));
    for (int i = 0; i < index; i++)
    {
        newArr[i] = arr[i];
    }
    for (int i = index; i < n - 1; i++)
    {
        newArr[i] = arr[i + 1];
    }
    return newArr;
}

```

راه دو:  
استفاده از لینکد لیست

```

#include <stdio.h>
#include <stdlib.h>

typedef struct CirNode
{
    int data;
    struct CirNode *next;
} CirNode;

int josephusProblem(CirNode *start, int n);

int main(int argc, char const *argv[])
{
    int n;
    scanf("%d", &n);

```

```

    CirNode *first = (CirNode *)malloc(sizeof(CirNode) * n);
    for (int i = 0; i < n; i++)
    {
        if (i == n - 1)
            first[i].next = &first[0];
        else
        {
            first[i].next = &first[i + 1];
        }
        first[i].data = i + 1;
    }

    printf("%d", josephusProblem(first, n));

    return 0;
}

int josephusProblem(CirNode *start, int n)
{
    if (n == 1)
        return start->data;

    start->next = start->next->next;
    josephusProblem(start->next, n - 1);
}

```

## • Recursive Function (تابع بازگشتی)

قبل اینکه «تابع بازگشتی» رو بهتون معرفی کنم، بیایم دو مثال رو ببینیم که درک تابع بازگشتی رو براتون ساده‌تر کنه.

مثال ۱:

```

def increment(num):
    return num + 1

def f(num):
    return increment(num)

```

```
print(f(5))
```

خب من تابع  $f$  رو توی پرینت صدا زدم که مقدار ریترن شدشو چاپ کنه. خب میره سراغ تابع  $f$ . توی تابع  $f$  نوشته که حاصل یه تابع رو ریترن کن!  
- عه؟! مگه میشه؟! میشه بگیم که حاصل یه تابع رو ریترن کن؟!  
+ آره میشه! اینجا هم همین گفته. پس میره اول حاصل تابع `increment` رو حساب کنه. میره اونجا. اونجا نوشته که  $num + 1$  رو ریترن کن. پس  $5 + 1$  یعنی  $6$  رو ریترن می کنه. حالا برمی گرده توی تابع  $f$  و حاصل تابع `increment` رو جاش قرار میده:

```
def f(num):  
    return 6
```

الآن  $6$  رو ریترن می کنه. پس  $6$  به سمت پرینت پاس داده میشه و پرینت هم برامون  $6$  رو چاپ می کنه.

## مثال ۲:

```
def square(num):  
    return num * num
```

```
def f1(num):  
    return square(num) * 2
```

```
def f2(num):  
    return f1(num) + 1
```

```
print(f2(2))
```

خب اینجا میاد میگه خب حاصل تابع  $f2$  رو خواستی چاپ کنم. پس میرم توی تابع  $f2$ . توی تابع  $f2$  میگه حاصل یه تابع دیگه رو حساب کن؛ بعلاوه یک کن؛ بعدش ریترنش کن.  
خب پس باید بره ببینه حاصل تابع  $f1(2)$  چیه. میره اونجا. اونجا هم میگه که ریترن این تابع، حاصل یه تابع، ضربدر  $2$  هست. پس میره توی تابع  $square(2)$  که ببینه حاصل چیه.  
اونجا میبینه که باید عددی که بهش پاس داده شده رو ضربدر خودش کنه و ریترن کنه. پس  $2$  رو ضربدر دو میکنه. میشه چهار. حالا  $4$  رو ریترن می کنه. حواسش هست که قبلاً کجای  $f1$  بود. حالا بازگشت میزنه و بر میگردد به دقیقاً جایی از تابع  $f1$  که از اونجا پرش کرده بود به یه جای دیگه:

```
def f1(num):  
    return 4 * 2
```



```
def f2(num):
    return f1(num) + 1
```

پس ۴ رو ضربدر ۲ می‌کنه و ۸ رو ریترن می‌کنه. حالا بر می‌گرده به عقب میره توی جایی که f1 بود:

```
def f2(num):
    return 8 + 1
```

خب اینجا هم همه چی به دست اومده و ۹ رو ریترن می‌کنه و توی پاسخ هم ۹ چاپ میشه. درواقع هر جا که به تابعی بر خورد، میره اول حاصل اون تابع رو حساب کنه. نمیتونه که تابع رو ریترن کنه! بلکه باید حاصل تابع رو اول به دست بیاره و یه چیز درستی که به دست اومد، جایگذاری کنه و ریترن کنه.

اصطلاحاً هی میره تو در تو و بعد باز می‌گرده. (بازگشت پیدا می‌کنه) حالا که فهمیدین چه جوریه، میریم سراغ توابع بازگشتی (:)

به طور کلی، هر تابعی که خودش توی خودش صدا بزنه، بهش می‌گیم «تابع بازگشتی».

- عه چه عجیب! مگه میشه؟!

+ بله! همراه باشین بهتون یاد میدم (:)

فرض کنین من می‌خوام یه عددی رو به توان یه عدد دیگه برسونم. بدون استفاده از توان یه راهش اینه:

```
def power(a, b):
    res = 1
    while b > 0:
        res *= a
        b -= 1
    return res
```

هر بار یکی از b کم میشه و تا وقتی b بزرگ‌تر از صفره، یه a در res ضرب میشه. درواقع b برابر صفر شه، می‌پره بیرون.

اما یه راه دیگش اینکه من بگم:

```
def power(a, b):
    res = 1
    if b == 0:
        return 1
    return a * power(a, b-1)
```

میگه اگر b برابر صفر بود، ۱ رو ریترن کن. وگرنه میاد خط بعدی، میگه a رو ضربدر تابع power کن اما ایندفعه متغیر دوم به جای اینکه b باشه، b-1 هست. درواقع یه مرحله انجام شده برای همین b-1 شده. درواقع اینطوری انجام میشه:

**2\*\*5:**

**power(2, 5) → return 2 \* power(2, 4)**

خب می‌گه توی **power(2, 5)** من باید ۲ رو ضربدر یه تابع کنم. تابع **power(2, 4)** اما خب من اول باید برم **power(2, 4)** رو حساب کنم که ببینم حاصلش چی میشه که جاش بذارم:

**power(2, 4) → return 2 \* power(2, 3)**

خب **power(2, 4)** هم حاصل ۲ ضربدر یه تابع که **power(2, 3)** هست رو ریترن می‌کنه! خب پس بریم این یکیو به دست بیاریم:

**power(2, 3) → return 2 \* power(2, 2)**

عه این هم همینطوره! پس بریم باز تابع رو حساب کنیم:

**power(2, 2) → return 2 \* power(2, 1)**

**power(2, 1) → return 2 \* power(2, 0)**

**power(2, 0) → return 1**

همینطور روند ادامه پیدا می‌کنه تا وقتی که می‌گه **power(2, 1)** مقدار **power(2, 0)** رو ریترن می‌کنه. خب حالا اینجا مقدار **power(2, 0)** یه چیز مشخصه و میشه عادی ریترنش کرد! چون توی **if** می‌بینیم که **b** برابر صفر هست و مقدار ۱ رو باید ریترن کنه. پس بازگشت می‌زنه و هی جایگذاری می‌کنه:

**power(2, 0) → return 1**

**power(2, 1) → return 2 \* power(2, 0) → return 2 \* 1 → 2**

**power(2, 2) → return 2 \* power(2, 1) → return 2 \* 2 → 4**

**power(2, 3) → return 2 \* power(2, 2) → return 2 \* 4 → 8**

**power(2, 4) → return 2 \* power(2, 3) → return 2 \* 8 → 16**

**power(2, 5) → return 2 \* power(2, 4) → return 2 \* 16 → 32**

**print(power(2,5))**

و در آخر ۳۲ برمیگرده و چاپ میشه (:

- عه چه کار سختی! خب عادی بنویسیم دیگه!  
+ اولش شاید سخت باشه ولی راه بازگشتی، یه راه جالب برای حل مسأله هست که بعضی از مسأله‌ها رو به شدت ساده حل می‌کنه. حالا می‌رسیم بهش. فعلاً باید یکم تمرین کنیم که بهتر بتونیم درکش کنیم.

مثال! تمام توابع رو به کمک توابع بازگشتی بنویسین!

## مثال ۱:

تابعی بنویسین که دو عدد بزرگ‌تر از صفر بگیره و بگه کدوم بزرگ‌تره؟  
راهنمایی: هی یکی کم کنین و هروقت a برابر صفر شد، پس b بزرگ‌تره. اگر b برابر صفر شد، پس a بزرگ‌تره. اگر هر دو برابر بودن، که مساوین.

```
def bigger(a, b):  
    if a == b:  
        return "They are equal"  
    if a == 0:  
        return "Second one is bigger"  
    if b == 0:  
        return "First one is bigger"  
    return bigger(a-1, b-1)
```

اول چک‌ها رو انجام میدم و در آخر تابع رو دوباره صدا می‌زنم با a-1 و b-1.  
در واقع تمام توابع بازگشتی هی چیزا رو محدود می‌کنن و به یه base case یا یه حالت پایه و یه متوقف‌کننده می‌رسن. اینجا هم ما سه نوع متوقف‌کننده داریم که یه مقداری رو ریترن کنن و نه یه تابع. در واقع هنر اصلی شما، یافتن متوقف‌کننده‌هاست. اگر درست انتخاب نکنین، ممکنه بی‌نهایت بار تکرار شه! مثل loop بی‌نهایت.

## مثال ۲:

به صورت بازگشتی، اعضای یه لیست رو پرینت کنین.

پاسخ:

```
def print_list(l):  
    if len(l) == 1:  
        print(l[0])  
        return  
    print(l[0], end=' ')  
    return print_list(l[1:])
```

```
l = [1, 2, 3, 4]  
print(print_list(l))
```

هر بار یه دونه عنصر رو پرینت می‌کنم و بعد برای ایندکس ۱ به بعد لیست، تابع رو صدا می‌زنم.  
موقف‌کننده یا همون base case من هم زمانی که یه عنصر توی لیست باشه. صرفاً همون رو پرینت می‌کنم و یه return خالی می‌نویسم که None رو ریترن می‌کنه.

```
l = [1, 2, 3, 4]  
print(1)
```

```

return print_list(l[1:]) # Return None
l = [2, 3, 4]
print(2)
return print_list(l[1:]) # Return None
l = [3, 4]
print(3)
return print_list(l[1:]) # Return None
l = [4]
print(4)
return None

```

فلش رنگ بنفش (رو به پایین)<sup>۷۲</sup> رفت و رنگ آبی (رو به بالا) برگشت رو نشون میده. (اول کد مسیر هایلایت‌های صورتی رو طی می‌کنه و بعد به وسیله فلش آبی برمی‌گرده)

حالا که رسید به تهش که **None** ریترن میشه، بر می‌گرده و جای توابع **None** می‌ذاره. میره بالا. بالا هم **return** رو پاس میده به قبلی. هی میره عقب تا وقتی که به تابع اصلی برسه. تابع اصلی، **None** رو از تابع جلوییش گرفته و حالا **None** رو به **print** پاس میده.

پس درواقع **None** که چاپ میشه، توسط تابع **اولی** به پرینت پاس داده شده. ولی توسط داخلی‌ترین تابع ساخته شده و هی دست به دست شده تا رسیده به تابع اولی.

به نظرتون اگر نخواهیم **None** چاپ شه، باید چیکار کنیم؟

+ به سادگی، صرفاً نیازه که تابع رو صدا بزنیم. مقادیر چاپ میشه ولی چیزی که ریترن میشه، چون توی پرینت نیست، مقدار ریترن‌شده (یعنی **None**) چاپ نمیشه! ریترن میشه ولی چیزی رخ نمیده. چیزی چاپ نمیشه. چون پرینت مقدار ریترن‌شده رو چاپ می‌کرد.

یعنی:

```

l = [1, 2, 3, 4]
print_list(l)

```

مثال ۳:

کتابخونه رندوم، کتابخونه‌ای هست که برای تولید اعداد نیمچه رندوم استفاده میشه. (چرا نیمچه رندوم؟ چون کامپیوتر بلد نیست عدد رندوم خالص بسازه. کامپیوتر مثل یه روباته! شما نمی‌تونین بهش بگین که یه عدد رندوم بده. نمی‌فهمه! باید یه چیز مشخص باشه. مثلاً ۲ ضربدر ۲. اینو می‌تونه جواب بده.

<sup>۷۲</sup> جهت هم مشخص کردم که اگر یکی سیاه سفید پرینت گرفته یا کوررنگ هست، بهتر بتونه تشخیص بده (:

چون مراحلش مشخصه. ولی یه عدد رندوم بده مشخص نیست. برای همین تولید اعداد رندوم یکی از چیزای سخت توی کامپیوتر هست.

```
import random
```

```
print(random.randint(1, 100))
```

میگیم از کتابخونه رندوم، تابع randint رو صدا بزن که یه عدد بین ۱ تا ۱۰۰ بده.

یه بازی در نظر بگیرین که یه عدد رندوم از ۱ تا ۱۱ پایتون در نظر میگیره (با تابع randint). بعد یه عدد از من میگیره. اگر برابر اون عدد دلخواه بود، پرینت کنه که درست حدس زدم. اگر عددی که حدس زدم، بزرگتر بود، چاپ کنه که عددم بزرگ بوده و باید یه عدد کوچکتري وارد کنم. اگر هم عددم کوچکتر بود، بگه اشتباهه و بزرگتر باید حدس بزنی.

پاسخش در وبسایت زیر قسمت «Example of a number guessing program using recursion»:

<https://pythongeeks.org/recursion-in-python/>

مثال ۴:

جمع عناصر یه لیستی که شامل اعداد integer هست رو حساب کنین.

پاسخ:

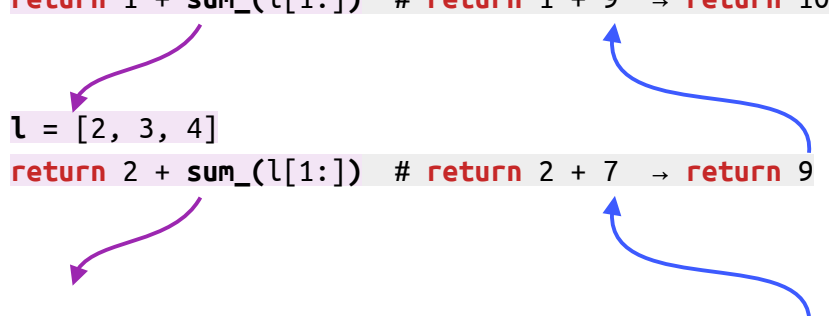
```
def sum_(l):  
    if len(l) == 1:  
        return l[0]  
    return l[0] + sum_(l[1:])
```

اسم تابع رو sum\_ نوشتم که با اسم sum خود تابع پایتون قاطی نشه. مروسومه که وقتی یه اسمی میخوان استفاده کنین که با اسم چیزای خود پایتون خاطی نشه، آخرش یه «underscore» میذارین.

جمع عنصر اول + جمع باقی عناصر.

Base case چیه؟ اگر یه عنصر باقی مونده بود، خودش رو ریترن کن.

```
l = [1, 2, 3, 4]  
return 1 + sum_(l[1:]) # return 1 + 9 → return 10  
  
l = [2, 3, 4]  
return 2 + sum_(l[1:]) # return 2 + 7 → return 9
```



```
l = [3, 4]
return 3 + sum_(l[1:]) # return 3 + 4 → return 7
```

```
l = [4]
return 4
```

رنگ بنفش رفت و آبی برعکس رو نشون میده. (اول کد هی میره تو که مسیر صورتیه و بعد به وسیله فلش آبی بر می‌گرده)

نقطه ضعف توابع بازگشتی

نقطه ضعف توابع بازگشتی اینه که چون از یه جایی می‌پره به یه جای دیگه، هی نیازه یه سری چیزا به خاطر داشته باشه. یعنی نیازه هی یادش باشه که کجا بود که بتونه برگرده. هی متغیرهای اونجا رو یادش باشه. هی میره پایین‌تر و هر مرحله باید یادش باشه مرحله قبل چیا داشت و کجاش بود. یعنی نیازه متغیرهای اون محدوده قبلی رو حفظ کنه و یادش باشه کدوم خط بود. به همین دلیل حافظه بیشتری اختیار می‌کنه. یه جایی به بعد هم نمی‌تونه جواب بده و ممکنه ارور بده. مثلاً تمرین ۱ رو ببینین.

**تمرین ۱!** همش با کمک توابع بازگشتی بنویسین!

**تمرین ۱:** تابع فاکتوریل اعداد رو پیاده‌سازی کنین.

**تمرین ۲:** تابع برعکس کردن یه string رو پیاده‌سازی کنین.

**تمرین ۳:** طول یه string رو به دست بیارین.

**تمرین ۴:** لگاریتم بر مبنای ۲ برای ورودی‌هایی که توان‌های ۲ هستن رو حساب کنین.

**تمرین ۵:** یه استرینگ palindrome، استرینگیه که چه از چپ و چه از راست خونده شه، یه چیزیه. مثلاً:

```
aba
abcba
aaaaa
```

تابعی بنویسین که تشخیص بده یه چیز palindrome هست یا نه؟

**تمرین ۶:** توی دنباله فیبوناچی، هر عدد، از مجموع دو عدد قبلیش به دست میاد:

```
1, 1, 2, 3, 5, 8, 13, 21, ...
```

تابعی بنویسین که جمله n ام دنباله فیبوناچی رو چاپ کنه.

**تمرین ۷:** تبدیل از باینری به دسیمال رو با یه تابع انجام بدین. ورودی یه string که عدد باینریه و خروجی یه integer که مقدارش به decimal هست.

## تمرین ۸: تبدیل دسیمال به باینری:

راهنمایی:

الگو پیدا کنین که چه جوریه.

قاعدتاً هر بار باقی‌مونده در سمت راست استرینگ باینری ما قرار می‌گیره و خارج‌قسمت میره برای تقسیم بعدی.

کی متوقف میشه؟ زمانی که آخرین تقسیم ۱ یا صفر باشه. پس اون موقع متوقف میشه و خودش قرار می‌گیره.

## تمرین ۹:

توی این تمرین ما می‌خوایم که صرفاً تفکر ریکرسیو رو یاد بگیریم و نمی‌خوایم کد رو کامل بنویسیم. صرفاً خط فکریشو می‌خوایم پیاده کنیم.

برج هانوی<sup>۷۳</sup> یه مثال خیلی معروفیه که می‌گه فرض کنین ما سه تا برج داریم (لینک پایین صفحه رو نگاه بندازین که بهتر درک کنین). حالا ما می‌خوایم که از یه ستون که پر دیسک هست، دیسکا رو به همین ترتیب که هستن (به ترتیب بزرگ به کوچک از پایین به بالا) توی یه برج دیگه قرار بدیم. اما این وسط دو شرط وجود داره:

۱- هر بار یه دیسک رو می‌تونین جابه‌جا کنین.

۲- هر بار صرفاً می‌تونین دیسک کوچکترو روی دیسک بزرگ‌تر بذارین و دیسک بزرگ‌تر رو نمی‌تونین روی دیسک کوچکترو بذارین.

## تمرین ۱۰:

جمع عناصر داخل یه لیست رو به دست بیارین. البته لیست ما می‌تونه داخل خودش لیست داشته باشه.

```
l = [0, [1, 2], [3, [4, 5]]]
# 0 + 1 + 2 + 3 + 4 + 5 = 15
```

راهنمایی:

اگر می‌خوایم بدونیم که یه متغیر از چه نوعیه (یعنی integer هست یا float یا لیست یا...)، می‌تونیم از تابع type که مال خود پایتونه استفاده کنیم. مثلاً:

```
num = 2
if type(num) is int:
    print("num is int")
```

```
l = [1, 2, 3]
```

73 English: [https://en.wikipedia.org/wiki/Tower\\_of\\_Hanoi](https://en.wikipedia.org/wiki/Tower_of_Hanoi)

Persian: [https://fa.wikipedia.org/wiki/%D8%A8%D8%B1%D8%AC\\_%D9%87%D8%A7%D9%86%D9%88%DB%8C](https://fa.wikipedia.org/wiki/%D8%A8%D8%B1%D8%AC_%D9%87%D8%A7%D9%86%D9%88%DB%8C)

```
if type(l) is int:
    print("l is int")
elif type(l) is list:
    print("l is a list")
```

output:

```
num is int
l is a list
```

میگه اگر `type(num)` اینتیجر است...  
is مثل انگلیسی ترجمش کنین! اگر فلان چیز ... است.

### تمرین ۱۱:

Bubble sort رو به صورت بازگشتی بنویسین.

راهنمایی:

for بیرونی رو می‌تونین به صورت بازگشتی پیاده‌سازی کنین.

### تمرین ۱۲:

<https://quera.org/problemset/608/>

توجه ۱: نمره ۸۰ و درست جواب‌دادن ۱ و ۳ و ۴ و ۵ برای این سؤال کافیه. دومی تایم لیمیت می‌خورین و ایراد نداره.

توجه ۲: برای محدود کردن تعداد اعشار، از تابع `format` استفاده کنین. مثلاً توی این سؤال که باید به ۲ اعشار محدود کنین، اینطوری استفاده میشه:

```
print(format(3, '.2f'))
print(format(12.9483, '.2f'))
```

یعنی بعد اعشارت حتماً دو رقم باشه.

راهنمایی:

<https://blog.faradars.org/determinant-of-a-matrix/>

### پاسفنامه:



پاسخ ۱:

```
def fac(n):  
    if n == 1 or n == 0:  
        return 1  
    return n * fac(n-1)
```

```
n = 5  
return 5 * fac(4) # return 5 * 24 → return 120  
n = 4  
return 4 * fac(3) # return 4 * 6 → return 24  
n = 3  
return 3 * fac(2) # return 3 * 2 → return 6  
n = 2  
return 2 * fac(1) # return 2 * 1 → return 2  
n = 1  
return 1
```

اگر بگیم مثلاً فاکتوریل ۱۰,۰۰۰ رو حساب کن، می‌نویسه:

```
RecursionError: maximum recursion depth exceeded in comparison
```

یعنی اینقدر تو در تو شده که نمی‌تونه! <sup>۷۴</sup>

پاسخ ۲:

```
def rev(s):  
    if len(s) == 1:  
        return s  
    return s[-1] + rev(s[:-1])
```

نفهمیدین چی شد؟ اون شکل اینکه چه اتفاقی رخ میده رو بکشین برای خودتون و فلش بزنین که هر بار چه اتفاقی میوفته.

پاسخ ۳:

---

۷۴ البته سیستمای مختلف (کامپیوترهای) تفاوت دارن. ممکنه یکی با چیزای کمتر ارور بده.

```
def lenght(s):
    if not s:
        return 0
    return 1 + lenght(s[1:])
```

قسمت if not s یعنی اگر s خالی بود. وقتی خالی باشه، boolean اش میشه False و درواقع یعنی اگر not False ....

پاسخ ۴:

```
def my_log(n):
    if n == 1:
        return 0
    return my_log(n//2) + 1
```

لگاریتم بر مبنای ۲ یعنی ۲ به توان چه عددی میشه اون عدد؟ پس هر بار تقسیم می‌کنیم.

پاسخ ۵:

هر بار باید ایندکس اول و آخر رو چک کنیم ببینیم یکسان یا نه. و این شرط باید برای تمام ایندکسها باشه. یعنی هی and باشه. حالا می‌تونیم اینطوری پیاده‌سازیش کنیم:

```
def is_pal(s):
    if len(s) == 0:
        return True
    return s[0] == s[-1] and is_pal(s[1:-1])
```

البته هم میشه len رو اینطوری نوشت:

```
if not s:
    return True
```

یعنی اگر not خالی.

```
s = 'abcba'
return 'a' == 'a' and is_pal('bcb') → return 'a' == 'a' and True → True
```

```
s = 'abcba'
return 'b' == 'b' and is_pal('c') → return 'b' == 'b' and True → True
```

```
s = 'abcba'
return 'c' == 'c' and is_pal('') → return 'c' == 'c' and True → True
```

```
s = ''  
return True
```

پاسخ ۶؛

هر جمله از جمع دو جمله قبلی به دست میاد. پس:

```
def fib(n):  
    return fib(n-1) + fib(n-2)
```

یعنی هر جمله، جمع دو تایی قبلیه. دقیقاً همین رو نوشتیم. اما خب همینطور باید بره عقب تا به یه **base case** و یه چیز مشخص عددی بخوره. جمله اول یک هست و مشخصه. پس می‌تونیم اینطوری بنویسیمش:

```
def fib(n):  
    if n == 1:  
        return 1  
    return fib(n-1) + fib(n-2)
```

اما یه سری تست کیس بنویسیم به ارور بر می‌خوریم:

```
print(fib(1))  
print(fib(2))  
print(fib(3))  
print(fib(4))  
print(fib(5))
```

ارور می‌گه: `RecursionError: maximum recursion depth exceeded in comparison`

پس احتمالاً هیچ‌وقت تموم نمیشه. یعنی **base case** رو درست به دست نیوردیم. بیایم یه عدد مثال بزنیم. مثلاً ۲ تا ببینیم چرا درست نیست.

خب ۲ میاد توی تابع. برابر ۱ نیست پس از `if` عبور می‌کنه و میره خط بعدی. خط بعدی می‌گه:

```
return fib(2-1) + fib(2-2)
```

خب برای `fib(1)` که میره و توی `if` عدد هم به دست میاد و ریترن میشه و مشکلی نداریم. اما برای `fib(0)` میره صداس میزنه و هیچ **base case** ای نیست که جواب بده! (در واقع از **base case** ما پریده) پس هی منفی‌تر و منفی‌تر میشه و هیچ‌وقت متوقف نمیشه! برای همین ارور داد که هرچی میرم توش تموم نمیشه.

```
fib(0):  
return fib(-1) + fib(-2)  
fib(-1):  
return fib(-2) + fib(-3)
```

پس نیاز به **base case** رو تغییر بدیم که وقتی که کمتر از ۱ شد هم جواب بده و بی‌نهایت نشه و هی منفی‌تر نشه. یه جایی متوقف شه:

```
def fib(n):  
    if n <= 1:  
        return n  
    return fib(n-1) + fib(n-2)
```

حالا دیدین که چقدر بازگشتی می‌تونه کمک کنه که راه‌حل ساده شه؟ با سه خط کد و عیناً نوشتن مفهوم، تونستیم فیبوناچی رو حل کنیم!

پاسخ ۷:

خب قبول دارین که با توجه به تصویری که بهتون نشون دادم، سمت راست‌ترین، ضربدر ۲ به توان صفر میشه؟ یعنی ضربدر ۱. یعنی درواقع خودش. پس سمت راست‌ترین هرچی باشه، خودش جمع میشه با بقیه. پس **base case** رو می‌تونیم همین در نظر بگیریم:

```
def bin_to_dec(b):  
    if len(b) == 1:  
        return int(b)
```

اگر یه عنصر باقی‌مونده بود، خودش رو ریترن کن. ولی خب چون **string** بود، تبدیل به **int** اش می‌کنیم.

خب برای بقیش چیکار کنیم؟

قبول دارن که اگر از سمت چپ بخوایم شروع کنیم (چون **string** شروعش از سمت چپه)، هرچی باشه (چه صفر چه یک)، ضربدر ۲ به توان یکی کمتر از **len** میشه؟ پس درواقع اینطوری:

```
def bin_to_dec(b):  
    if len(b) == 1:  
        return int(b)  
    return 2**(len(b)-1) * int(b[0])
```

خب این تنها که به درد نمی‌خوره. ادامه **string** هم باید باهاش جمع شه. یعنی ادامهش هم همین اتفاق بیوفته. پس برای ادامهش هم همین تابع رو صدا می‌زنیم که هر بار یه عنصر جمع شه و تابع رو برای ادامه استرینگ صدا بزنه:

```
def bin_to_dec(b):  
    if len(b) == 1:  
        return int(b)  
    return (2**(len(b)-1) * int(b[0])) + bin_to_dec(b[1:])
```

```
b = '1011'
```

```

return (2**(4-1) * 1) + bin_to_dec('011') # return 8 + 3 → return 11
b = '011'
return (2**(3-1) * 0) + bin_to_dec('11') # return 0 + 3 → return 3
b = '11'
return (2**(2-1) * 1) + bin_to_dec('1') # return 2 + 1 → return 3
b = '1'
return int('1') → return 1

```

دیدین چقدر بازگشتی می‌تونه کار رو ساده کنه؟

پاسخ ۸:

```

def dec_to_bin(n):
    if n == 1 or n == 0:
        return str(n)
    return dec_to_bin(n//2) + str(n % 2)

```

یعنی هر بار string باقی‌مونده به تهش اضافه میشه و خارج قسمت (حاصل تقسیم صحیح) میره برای مرحله بعد که تقسیم روش صدا بخوره. Base case هم زمانیه که ۱ یا ۰ باشه که خودش قرار می‌گیره.

یا آخرین قسمت چون خارج قسمت صفره (طبق عکس)، می‌تونستیم اینطور هم بنویسیم:

```

def dec_to_bin(n):
    if n // 2 == 0:
        return str(n % 2)
    return dec_to_bin(n//2) + str(n % 2)

```

پاسخ ۹:

فرض کنیم  $n$  تا دیسک داریم. خب قبول داریم که ما باید بزرگترین دیسک که ته‌ترین و فرض کنیم سمت چپ هست رو بذاریم روی برج سمت راست؟ پس قبلش  $n - 1$  تا دیسک رو باید بذاریم روی دیسک وسط که خالی شه و بتونیم بزرگترین دیسک رو بذاریم سمت راست.

خب حالا موضوع اینه که ما باید  $n - 1$  دیسک رو بذاریم وسط. اما چون قانون اینه که دیسک بزرگ‌تر نمی‌تونه روی دیسک کوچک‌تر قرار بگیره، مستقیم نمی‌تونیم انجامش بدیم. بلکه باید از یه برج دیگه کمک بگیریم که بتونیم ببریمشون وسط. از برج سمت راستی کمک می‌گیریم.

حالا  $n - 1$  دیسک با کمک راستی رفتن به وسط.  
بعدش بزرگترین دیسک رو می‌بریم راست.  
حالا باید همون  $n - 1$  تایی که وسط بودن رو با کمک سمت چپی ببریم راست.  
تمام! حالا به صورت کد می‌نویسیم:

```
def Hanoi(n, src, dst, tmp):  
    if n > 0:  
        Hanoi(n - 1, src, tmp, dst)  
        move disk n from src to dst  
        Hanoi(n - 1, tmp, dst, src)
```

این کد از کتاب زیر آورده شده:

“Algorithm” by “Jeff Erickson”<sup>75</sup>

صرفاً همین منطقش برامون مهمه. اینکه هی میره ته و هی یکی یکی move می‌کنه و تا وقتی  $n$  بزرگ‌تر از صفره کار رو ادامه میده و recursive انجام میده.  
کد پایتونی که واقعا حلش کنه، توی ویکی پدیا هست.<sup>76</sup> البته اگر تازه دارین با برنامه‌نویسی آشنا میشین، نیاز نیست حلش کنین (:

پاسخ ۱۰:

ما باید تابع رو هی برای عناصر صدا بزنیم و تا وقتی به عدد نخوردیم، هی باید بریم داخل‌تر. یعنی هی بریم داخل‌تر و هی نگاه کنیم ببینیم کی به عدد صحیح بر می‌خوریم. هر وقت عددی صحیح پیدا کردیم، باید ریترنش کنیم. این ریترن شده‌ها رو توی یه لیست قرار میدیم که بعدش یه لیست صرفاً شامل اعداد صحیح داشته باشیم و طبق تابع sum جمع کنیم.  
باید هی بریم تا برسیم به یه عدد صحیح:

```
def deep_sum(x):  
    if type(x) is int:  
        return x
```

خب این از base case. حالا بریم زمانی که به لیست خوردیم چی؟ باید هی بریم تو تر:

```
[deep_sum(i) for i in x]
```

یعنی صدا بزن تابع رو برای تک‌تک عناصر لیستمون و خب بریزش توی لیست.

- چرا بریزیم توی یه لیست؟

+ چون در آخر یه لیست شامل صرفاً اعداد داشته باشیم. قبول دارین تابع در نهایت که تو رفت، یه عدد پاس میده؟ این عدد میاد توی لیست قرار میگیره و یه لیست شامل صرفاً عدد میسازه. اینطوری می‌تونیم با تابع sum، حاصلش رو به دست بیاریم:

```
def deep_sum(x):
```

75 <https://jeffe.cs.illinois.edu/teaching/algorithms/>, Creative Commons Attribution 4.0 International license.

76 [https://en.wikipedia.org/wiki/Tower\\_of\\_Hanoi#Recursive\\_implementation](https://en.wikipedia.org/wiki/Tower_of_Hanoi#Recursive_implementation)

```

if type(x) is int:
    return x
return sum([deep_sum(i) for i in x])

```

همین رو اگر می‌خواستین به روش عادی حساب کنین، خیلی سخت‌تر میشد!

### پاسخ ۱۱:

هر وقت که نیاز به جابه‌جایی بود، یعنی ممکنه در ادامه هم سورت نشده باشه و باید هی بریم تو تر (با صدازدن دوباره تابع) ولی اگر سورت شده باشه، flag ما false تغییر پیدا نمی‌کنه و یه راست خود تابع رو ریترن می‌کنیم.

توی زبان‌های دیگه برنامه‌نویسی، چیزی به نام لیست نداریم. یه چیز دیگه مشابه این داریم اسمش «آرایه» یا «array» هست. برای همین از الان به بعد اسم آرایه رو بیشتر می‌بریم.

```

def bubble_sort(arr):
    flag = False
    for i in range(len(arr)-1):
        if arr[i] > arr[i + 1]:
            arr[i], arr[i + 1] = arr[i + 1], arr[i]
            flag = True
    # if flag == False -> arr is sorted. so we don't need to go
    further.
    if not flag:
        return arr
    else:
        return bubble_sort(arr)

```

البته نیاز به نوشتن else هم نبود. چون اگر توی if رفت، ریترن میشه و تموم میشه و اصلاً به else نمی‌رسه و اگر هم توی if نرفت، پس قطعاً میاد خط بعدیش که return هست.

### پاسخ ۱۲:

حالت پایه ما چیه؟ زمانی که ماتریسی به طول ۲ داشته باشیم. پس اینو می‌نویسیم:

```

def determinant(l):
    if len(l) == 2:
        return l[0][0] * l[1][1] - l[0][1] * l[1][0]

```

خب اما اگر طولش بیشتر بود چی؟

باید از ردیف اول (ایندکس صفر)، شروع کنیم و عناصر رو به ترتیب ضربدر ماتریس‌های دیگه‌ای کنیم. - چه ماتریسایی؟

+ درواقع اونایی که از ردیف ۱ به بعد شروع میشن و ستون i ام (ستونی که عدد خارجی ما در ماتریس ضرب شده) رو ندارن.

```
def determinant(l):
    if len(l) == 2:
        return l[0][0] * l[1][1] - l[0][1] * l[1][0]
    return l[0][i] *
        determinant([row[:i] + row[i + 1:] for row in l[1:]])
for i in range(len(l))
```

درواقع ما باید اینجا دترمینان رو برای ماتریس‌های کوچکتری که ضربدر عدد شدن صدا بزنیم. پس باید ماتریس بهش پاس بدیم. یه ماتریس از چی تشکیل میشد؟ از لیستی که شامل یه سری لیست‌هاست. پس اینجا هم می‌گیم که چیزی که به تابع پاس میدیم یه لیستی هست که با نارنجی مشخص کردم. ([ ])

حالا وظیفه ما ساخت ماتریس‌های کوچکتره:

پس می‌گیم که درون [ ] که یه لیسته، یه سری لیست دیگه بگذار. (ساخت لیست دوبعدی و ماتریس) این لیست‌ها از concatenate کردن دو لیست به دست میان. row[i:] و row[i + 1:]. که یعنی از هر ردیف لیستیمون (که خود ردیف‌ها از ایندکس ۱ شروع میشن)، عناصری رو بذار که ستونشون با ستون اونی که ضرب کردیم توش یکی نباشه. (یعنی اون رو جا بندازه).

اما حواسمون هم هست که یکی در میون منفی و مثبت. پس این رو هم تأثیر میدیم:

```
def determinant(l):
    if len(l) == 2:
        return l[0][0] * l[1][1] - l[0][1] * l[1][0]
    return (-1) ** i * l[0][i] *
        determinant([row[:i] + row[i + 1:] for row in l[1:]])
for i in range(len(l))
```

همه این چیزایی که حساب شدن هم در آخر باید با هم جمع شن. پس از sum کمک می‌گیریم:

```
def determinant(l):
    if len(l) == 2:
        return l[0][0] * l[1][1] - l[0][1] * l[1][0]
    return sum((-1) ** i * l[0][i] *
        determinant([row[:i] + row[i + 1:] for row in l[1:]])
for i in range(len(l)))
```

تمام! قبول دارم یکم نوشتنش توسط خودستون سخته ولی توی ۳ خط تونستیم ماتریس حساب کنیم و این خیلی خوبه و خیلی ساده‌تر از روشای دیگه هست! این رو اگر می‌خواستیم با حالت عادی پیاده‌سازی کنیم، خیلی سخت‌تر میشد! کد رو کامل می‌کنیم:



```

def determinant(l):
    if len(l) == 2:
        return l[0][0] * l[1][1] - l[0][1] * l[1][0]
    return sum((-1) ** i * l[0][i] *
               determinant([row[:i] + row[i + 1:] for row in l[1:]])
               for i in range(len(l)))

l = []
matrix = []
n = int(input())
for i in range(n):
    l = input().split()
    l = [float(j) for j in l]
    matrix.append(l)

print(format(determinant(matrix), '.2f'))

```

کد رو میشد یکمی بهینه کرد که هر بار نخوام len حساب کنیم (اما تفاوتی توی تایم لیمیت ایجاد نمی‌کنه):

```

def determinant(l, length):
    if length == 2:
        return l[0][0] * l[1][1] - l[0][1] * l[1][0]
    return sum((-1) ** i * l[0][i] *
               determinant([row[:i] + row[i + 1:] for row in l[1:]],
                             length-1) for i in range(length))

l = []
matrix = []
length = int(input())
for i in range(length):
    l = input().split()
    l = [float(j) for j in l]
    matrix.append(l)

print(format(determinant(matrix, length), '.2f'))

```

هر بار که تابع دوباره صدا زده میشه، میگیریم ایندفعه طول تابع،  $length - 1$  هست.

تمرین‌های بیشتر برای بازگشتی:

## • Dictionary

فرض کنیم یه دیتابیس به شکل زیر می‌خواهیم بسازیم:

Username	Password
Alex	Alex256
James	James512
Maria	maria1024

خب بخواهیم با لیست پیاده‌سازی کنیم، باید یه لیست برای username و یه لیست برای password بسازیم که ایندکس‌های متناظر به هم مرتبطن.

اما این یکم سخته. اینجا پایتون یه چیزی داره به نام Dictionary. اینطوری می‌تونیم پیاده‌سازی کنیم:

```
database = {  
    'alex': 'alex256',  
    'James': 'James512',  
    'Maria': 'Maria1024'  
}
```

به سمت چپا می‌گیم کلید (key) و به راستیا می‌گیم مقدار (value). اما برخلاف لیست، نمی‌تونه داخل خودش دو چیز یکسان داشته باشه. یعنی اگر به صورت زیر بنویسیم:

```
database = {  
    24: 'alex256',  
    'James': 'James512',  
    'Maria': 'Maria1024',  
    'James': '1234'  
}
```

چون James دوبار تکرار شده، مقدار دومی رو در نظر می‌گیره. چاپش کنیم بهتر متوجه شین:

```
print(database)
```

اگر بخوایم به یکی از عناصرش دسترسی پیدا کنیم، اینجا اسم کلید رو می‌بریم. مثلاً:

```
database = {  
    24: 'alex256',  
    'James': 'James512',  
    'Maria': 'Maria1024',  
    'James': '1234'  
}
```

```
print(database['Maria'])
```

کلیدها نمی‌تونن یه دیکشنری، لیست یا set باشن. (tuple می‌تونن باشن) ولی value ها هرچیزی می‌تونن باشن.

تابع len هم داریم و تعداد رو میده.

متدهای dictionary هم چیزای جالبین برای کار باهاش:

[https://www.w3schools.com/python/python\\_ref\\_dictionary.asp](https://www.w3schools.com/python/python_ref_dictionary.asp)

مثلاً بخوایم یه چیز آپدیت یا اضافه کنیم:

```
database.update({'James': 1234})
```

درواقع نگاه می‌کنه که اگر James نبود، اضافه‌اش می‌کنه. اگر بود، مقدارشو آپدیت می‌کنه. (این تغییرات روی خود dictionary اعمال میشه و چیزی ریترن نمی‌کنه! ریترن انجام این عبارت، None هست!)

با for میشه کارهای جالبی با کلید و مقدارها کرد. توی لینک زیر بخونیدشون:

[https://www.w3schools.com/python/gloss\\_python\\_loop\\_dictionary\\_items.asp](https://www.w3schools.com/python/gloss_python_loop_dictionary_items.asp)

یکم با چیزای بالا بازی کنین و تمرین کنین که خیلی مهمن.

مقدار یه کلید دیکشنری می‌تونه خودش یه دیکشنری باشه! مثل لیست دو بعدی:

```
school_dict = {  
    'students': {  
        'amir': 20,  
        'Kourosh': 21,  
        'Alex': 21,  
        'Hannah': 22  
    },  
    'teachers': {  
        'Mia': 45,  

```

```
'Angelina': 48,
'Jimmy': 44,
'Hannah': 50
}
```

```
print(school_dict)
```

چون دیکشنری دو بعدیه، پرینت کردنش خیلی نامرتبه. برای همین پایتون یه فانکشن پرینت دیگه‌ای به نام «pprint» داره که میشه چیزای دوبعدی و سه بعدی رو قشنگ‌تر پرینتش کرد:

```
from pprint import pprint
pprint(school_dict)
```

حالا بیایم مثلاً یه value خاصی رو پرینت کنیم:

```
print(school_dict['teachers']['Hannah'])
```

به syntax توجه کنین که اشتباه ننویسینش!

درواقع می‌گیم یه school\_dict داریم که شامل دو عنصره. یکی students یکی teachers. حالا هرکدوم شامل یه دیکشنری هستن!

بین این دو عنصر یه کاما بزرگ نارنجی گذاشتم که جداپذیریش بهتر باشه براتون.

- عه اینجا دو تا Hannah داریم! اشکال نداره؟

+ این Hannah با اون یکی تفاوت داره! یکیش یه دیکشنری هست که value برای students هست

و دومی یه دیکشنری دیگه برای value برای teachers. درواقع دو دیکشنری مجزا هستن!

البته با متدها هم می‌تونستیم همینو چاپ کنیم:

```
print(school_dict.get('teachers').get('Hannah'))
```

- فرقی ندارن چه با عادی و چه با متد برم؟

+ فرقی ندارن ولی اگر عادی بری و چیزی بدی که نباشه، ارور میده و برنامه همونجا تموم میشه. ولی

اگر با متد بری، اگر توش نباشه و پیداش نکنه، None ریترن می‌کنه و ارور نمیده و برنامه تموم نمیشه!

این بهتره! همیشه باید سعی کنین که برنامه‌تون یه دفعه تموم نشه. بلکه بتونه با چاپ یه متن ارور، به

کارش ادامه بده. مثلاً دو مورد زیر رو انجام بدین تا ببینین اولی None چاپ می‌کنه و دومی ارور میده:

```
print(school_dict.get('teachers').get('a'))
print(school_dict['teachers']['a'])
```

حالا می‌تونیم error handling کنیم (هندل کردن ارورها):

```
value = school_dict.get('teachers').get('a')
if value != None:
    print(value)
```

**else:**

```
print('Either the value is "None" or there is no key for given arguments')
```

اگر None نباشه، پس پرینتش می‌کنم. اگر باشه، چون دو حالت وجود داره:

۱- همچین کلیدی وجود نداشته که بخواد value بده.

۲- کلید وجود داشته ولی چون value هر چی می‌تونه باشه (هرچی! حتی None)، ممکنه value مقدارش None بوده باشه.

پس چاپ می‌کنم و بهش میگم یکی از دو حالت بالاس.

دیگه کدم اررو نمی‌خوره و کرش نمی‌کنه و این خیلی بهتره!

## • Dictionary Comprehensions

مثل لیست، میشه for رو درون dictionary هم به کار برد. مثلاً:

```
first_names = ['Bruece', 'Edward', 'Ronald']
last_names = ['Schneier', 'Felten', 'Rivest']
database = {first_names[i]: last_names[i]
             for i in range(len(first_names))}
print(database)
```

یه لیست داریم شامل first\_name ها و یه لیست شامل last\_names. حالا می‌گیم یه دیکشنری بساز که کلیدها first\_names[i] باشن و value ها به ترتیب همون ایندکس‌ها از last\_names باشن. ایندکس‌ها هم توی رنج اندازه نام‌ها باشن.<sup>۷۷</sup>

## • match case (switch case)

فرض کنین که یه سن می‌گیریم. بعد می‌گیم اگر سن ۱۸ بود، پرینت کنه «۱». اگر ۲۰ بود، پرینت کنه «۲» و اگر ۲۲ بود، پرینت کنه «۳». و اگر هیچکدوم نبود، پرینت کنه «Non of them»  
این رو باید اینطوری پیاده‌سازیش کنیم:

```
age = int(input())
if age == 18:
    print(1)
elif age == 20:
    print(2)
elif age == 22:
    print(3)
```

<sup>۷۷</sup> اسمایی که نوشتیم، از بزرگترین افراد در زمینه کامپیوتر و رمزنگاری (cryptography) هستن: مثلاً Ronald Rivest نویسنده یکی از معروف‌ترین کتاب‌های الگوریتم، برنده جایزه Turing که به نوبل کامپیوتر معروفه، استاد دانشگاه MIT و...

**else:**

```
print("Non of them")
```

اما این یکم قشنگ نیست. برای همین پایتون یه قابلیت دیگه داره به نام «match case». میگه رو یه متغیر چک‌ها رو انجام بده. ولی اینطوری:

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

```
match age:
```

```
    case 18:
```

```
        print(1)
```

```
    case 20:
```

```
        print(2)
```

```
    case 22:
```

```
        print(3)
```

```
    case _: # Default -> else
```

```
        print("Non of them")
```

یعنی میگه match ها رو برای age پیدا کن. مثلاً اگر ۱۸ بود، پرینت کن ۱. در آخر یه چیز دیفالت و پیشفرض داریم که اگر بقیه نشدن (یه چیزی شبیه else). ولی می‌نویسیم case و بعدش یه علامت «underscore» می‌ذاریم.

**مثال ۱:**

یه نام بگیرین. اگر برابر «kourosh» بود، چاپ کنه «۱». اگر برابر «amir» بود، چاپ کنه «۲». وگرنه چاپ کنه «۳»

**پاسخ:**

```
name = input()
```

```
match name:
```

```
    case "kourosh":
```

```
        print(1)
```

```
    case "amir":
```

```
        print(2)
```

```
    case _:
```

```
        print(3)
```

حتی می‌تونیم بگیریم اگر یکی از این حالات بود. (or درواقع! ولی خب یه شکل دیگه می‌نویسیمش):

```
name = input()
```

```
match name:
```

```
    case "kourosh":
```

```
        print(1)
```

```
    case "amir" | "jason":
```

```

print(2)
case _:
print(3)

```

علامت «|» شکل «or» ریاضیه.

تازه میشه بیشتر باهاش بازی کرد! یعنی بگیریم در صورتی اون match رو انجام بده که اون مثلاً متغیر ما توی یه لیست باشه:

```

name_list = ["jason", "Adi", "Bruce"]
name = input()
match name:
    case "kourosh":
        print(1)
    case "amir" | "jason" if name in name_list:
        print(2)
    case _:
        print(3)

```

اگر به ورودی amir رو بدیم، میاد توی دومی می‌گه خب case ما دومیه. اما یه شرط هم گذاشتی! پس شرط رو چک می‌کنم. عه! amir توی لیست نیست! پس از این case عبور می‌کنه و میره حالتای دیگه. برای همین ۳ چاپ میشه.

اگر به ورودی «jason» رو بدیم، باز میاد دومی؛ شرط رو هم چک می‌کنه می‌بینه شرط برقراره! پس ۲ رو چاپ می‌کنه.

قدرت پایتون به سادگی و موجود بودن لایبرری‌های زیاده برای کارهای متفاوت. هنر شما اینه که لایبرری‌هایی که متناسب با کار شما هستن رو پیدا کنین و نحوه کار باهاشون رو یاد بگیرین.

ادامه دارد...

نسخه‌های جدید در سایت:

<https://github.com/kamal331/Books/>