

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 - Static Code Analysis

نمونه آموزش

- چرا این آموزش؟

+ من آموزش‌های مختلفی که بوده رو دیدم. ولی مشکلاتی داشتن که به نظرم مانع از پایه درست یادگرفتن میشه. مثلاً:

- آموزش پایتون در ۱۰ ساعت.

جز اینکه شما یه زبون برنامه‌نویسی دیگه قبل پایتون بلد بوده باشین، همیشه واقعاً پایتون رو تو ۱۰ ساعت یاد بگیرین. یادگیری پایتون مثل یادگیری انگلیسیه. بله میشه توی ۱۰ ساعت بیایم قواعد فعلاً و اسما و نمیدونم فلان رو تا حدی گفت. اما آیا شما انگلیسی یاد می‌گیرین؟! نه!

- دوره آموزش پایتون

اکثر دوره‌هایی که من دیدم هم پولی هم رایگان، چند تا مشکل دارن.

۱- تمرینات خوبی ندارن و تمرینات و توضیحاتشون کمه.

۲- فکته صحبت می‌کنن. جای اینکه شالوده رو در ذهن بکارن که فرد خودش بتونه کد بنویسه و یا به اهمیت یه چیز خودش پی ببره، فکت رو در اختیار می‌ذارن به عنوان یه قانون.

۳- آکادمیک نیستند.

مورد سوم خیلی اهمیت داره. به وضوح تفاوت فردی که واقعاً در دانشگاه دنبال علم بوده و مطالعه دانشگاهی + خارج دانشگاهی داشته (نه هر کسی که صرفاً دانشگاه رفته)، به دلیل داشتن دانش از مباحثی مثل معماری کامپیوتر، سیستم عامل، کامپایلر و... خیلی دقیق‌تر و علمی‌تر از فردیه که به صورت تجربی صرفاً دوره گذرونده. این باعث میشه که کدهای تمیزتر، بهتر، با سرعت بالاتر بنویسه. به وضوح دیدم که فردی که یه کورس رو عادی گذرونده و بعد رفته توی همون کورس رو با یه استاد خوب و همچنین مطالعه گسترده گذرونده، چقدر سطح علمیش بالاتر رفته.

برای همین توی این آموزش سعی کردم مباحث علمی و آکادمیک‌تر باشن و باعث شن شما کد تمیزتر، سریع‌تر و علمی‌تر بنویسین.

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 کنن. 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 ذهنیت 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.^۱

آیا شنیدین که میگن هوش مصنوعی قراره آدم‌ها 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

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>

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 به عنصر کمکی ببینیم و از کامپیوترها استفاده کنیم - و نه کامپیوترها از ما -^۲ نگرانی بابت اینکه قراره شغلمون تحت تأثیر قرار بگیره، نداریم.

به قول استیفن هاو کینگ:

"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 به سری از افراد شغلشون رو از دست میدن.

- «**راهنمایی**» های داخل کتاب چی هستن؟

+ **ببینن برنامه‌نویسی مثل ریاضیات.** 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 به 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؟**

+ **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 به خرده سطحشون از سطح ساده بالاتره و آدمایی که می‌خوان بیشتر یاد بگیرن بخوننش.

^۲ منظور این هست که عملاً درگیر و پایبند 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.

- 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 علاوه بر اون، حالتای دیگش رو هم تست کنین.

- نظری دارم برای بهبود این آموزش. می‌تونم بگمش؟

+ بله حتماً! خوشحال میشم! راه ارتباطی به زودی در صفحه اول گیتهاب منتشر می‌شود:

<https://github.com/Mohammad-Kamal-mk>

دونیت

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 کنه و همچنین تمایل داشت بابت وقتی که من برای آموزش دانش خودم به صورت رایگان و آزاد به دیگران صرف کردم، کمک کنه، می‌تونه از طریق لینک زیر، به من **دونیت** (کمک/حمایت مالی) کنه:

<https://zarinp.al/mkamal>

حق نشر

این کتاب جز در قسمت استفاده تجاری، به وسیله لایسنس زیر عرضه شده:

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

استفاده از مطالب این کتاب به شرط ذکر منبع و راه‌های ارتباطی من، بلامانع است. اگر می‌خواهید از این کتاب استفاده تجاری کنید، قسمتی از مبلغ را به من برگردانید. همچنین ذکر کامل منبع و راه‌های ارتباطی من، الزامی است. (راه‌های ارتباطی به زودی در صفحه اول وبسایت گیت‌هاب^۳ نوشته می‌شوند.)

3 <https://github.com/Mohammad-Kamal-mk/>

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 بهتر بگه. می‌تونین به عنوان یک مَنجِم، برنامه‌ای بنویسین که روند چرخش ستاره‌ها رو توی کامپیوتر شبیه‌سازی کنه که بدوین 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 راهگشای‌تر دارن.

خب حالا برگردیم به اینکه برنامه‌نویسی کجای رشته کامپیوتر قرار داره؟ برنامه‌نویسی درواقع ابزاره. شما مثلاً می‌خوای یک سیستم عامل طراحی کنی، میری برنامه‌نویسی سطح پایین انجام میدی. زبونی 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 کنین من یک برنامه‌ای می‌خوام بنویسم که یک 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 یه 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 یه سری قابلیت دارن که تأخیر رو کاهش بدن.

بذارین یه مثال بزنم! فرض کنین من منشی یه 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 برسی کنار میز من و من تازه برگردم دنبال پروندت.

۲- تا از دور دیدمت، برگردم 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 متغیرا 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 مرتب شده سریع تره؟

چون کامپیوتر شروع می کنه از اول، دو سه تای اول می بینه وارد if نمیشه. اما از یه جایی به بعد، می بینه داره وارد if میشه. پس می گه بار بعدی که رسیدم به if، توی زمانی که شرط داره چک میشه، من بیکار نمیشینم! میرم توی if و چیزای داخلش رو حساب می کنم که یکم بیوفتیم جلو. درواقع به دلیل اینکه یه سری محاسبات جلو جلو انجام میشه، حالت مرتب شده سریع تره!

ولی توی مرتب نشده، می بینه یه بار میره تو if، یه بار نمیره و اصلاً نمی فهمه باید چیکار کنه و کدوم دستورا رو جلو جلو اجرا کنه که سرعت زیاد شه!^۴

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

خب حالا یه سوال! فرض کنین شما مدیر گوگل هستین. کدوم رو ترجیح می دین؟
یه برنامه نویس استخدام کنین که دانش بالایی داره و جوری با نحوه نوشتار کد و الگوریتم آشناست که کدی می نویسه که ۱ ثانیه طول میکشه تا جواب بده.
یه برنامه نویس دیگه ای 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 می کنین.

این ارقام خیالی نیستن! ما خودمون توی همین آموزش یه الگوریتم بررسی اینکه یه عدد اوله یا نه می نویسیم. بعدش سعی می کنیم بهینش کنیم و در آخر، بهینه ترین کد، می تونه تا بالای ۳,۰۰۰ برابر سریع تر از کد اولمون بشه! بله! بالای ۳,۰۰۰ برابر سریع تر! از همین الان می خوام بهتون یاد بدم که سعی کنین درست کد بنویسین و از همین الان بتونین خمیرمایه ذهنتون رو درست شکل بدین!

یکم میریم بالاتر، شما برنامه های مختلف رو می نویسین. مثلاً تلگرام و اینا. اینجا هم نیاز به اینکه چه جوری کد بنویسین که خوب باشه دارین ولی دیگه نیاز به مسائل سخت افزاری کمتره توشه. ولی نیازه که انواع الگوریتم ها رو یاد بگیرین و بدونین چه جور برنامه بنویسین.
- یعنی چی؟

^۴ به این میگن «branch predictor». مبحث سختیه و فعلاً نمی تونین بفهمیدش درست! ولی اگر دانش از رجیستر و کمی زبون C و یا اسمبلی و کمی نحوه ران شدن برنامه ها 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/>

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

راه دوم بهش می گن باینری سرچ (جست و جو دودویی) میگه من اولین حدسم رو عدد وسطی انتخاب می کنم. 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 می کنم و به سرعت جواب رو به دست بیارم. این الگوریتم بسیار بسیار سریع تر از الگوریتم عادی هست که سرمو بندازم پایین و پیداش و دونه دونه جلو برم و حدس بزنم.

باز میاین بالا. چیزایی مثل طراحی وبسایت و اینا. مخصوصاً قسمتی که کاربرا روی صفحه می بینن، خیلی خیلی کم نیاز به اون درک سخت افزاری داره. شما مثلاً خوشگلی وبسایت، اینکه این دکمه بزنم چی رخ میده و اینا رو طراحی می کنیم و خب قشنگی هست.

برنامه نویسی یه ابزاره. این ابزار همه جا کاربرد داره. یعنی شما هر رشته ای بخواین برین، باید تا حدی برنامه نویسی بلد باشین. نه اینکه برنامه نویس باشین، بلکه بتونین ابزار بنویسین که کارهاتون رو انجام بده.

بذارین یه مثال بزنم. فرض کنیم بخواین یه گلو صندوق رو باز کنیم. رمزشم ندارین. چیکار می کنیم؟ قاعدتاً دونه دونه ترکیب رمزا رو امتحان می کنیم تا وقتی بالاخره باز شه دیگه. ترکیبیات سال دهمتون!

همین رو برای یافتن پسورد در امنیت استفاده می کنن. ولی امتحان کردن همش دونه دونه سخت و زمان بره. برای همین هم هست که شما مثلاً با همین پایتون، ابزار می نویسین که بره خودکار امتحان کنه. درواقع کارها رو خودکارسازی می کنه. درواقع شما توی امنیت هم به ابزارنویسی احتیاج دارین. شما رشته فیزیک هم بری، در کارهایی مثل مدلسازی حرکت ماشین، جمع اوری و محاسبه محاسبات نیاز به برنامه نویسی پیدا می کنیم. ولی برای ابزارنویسی و خودکارسازی!

• آشنایی کوتاه با سفت‌افزار با دید برنامه‌نویسی

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 می‌نویسین 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 نگهداریشن 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 می‌داریم. یه انباری. این انباری ما توی کامپیوتر، اسمش هارد هست. (البته می‌تونه SSD هم باشه. SSD هم یه نوع انباری دیگس. توجه کنین که SSD هم یه نوع درایو ذخیره‌سازیه و هارد نیست! پس کلمه «هارد SSD» کلمه‌ای اشتباهه)

خب مثلاً وقتی شما دکمه روشن شدن رو می‌زنین 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 زمان‌بره و کنده!

اینجا اومدن یه مکانی نزدیک‌تر و کوچکت‌ر به CPU ساختن به نام RAM. جایی که هر چی ما باهاش کار داریم بیاد اونجا بشینه که دسترسی CPU بهش ساده‌تر باشه.

درواقع وقتی روی یه برنامه کلیک می‌کنین، اون برنامه پا میشه و میاد توی RAM میشینه که CPU بتونه راحت‌تر بهش دسترسی داشته باشه.

اطلاعات برنامه‌های درحال 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 برنامه‌ای 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 می‌کنه. دستوراتش اینه:

۱- یک ثانیه صبر کن

۲- یه دونه صدا رو زیاد کن

۳- کارهای بالا رو تا وقتی صدا ۱۰۰ نشره 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 بینم 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 کارا 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.
بینین 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»^۵

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

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

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 به 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:

$(25)_{10} = (11001)_2$
 $87_{(10)} = (1010111)_2$

راه دوم: (سادگیش موقع تبدیل به باینری هست)

فرض کنیم می‌خوایم 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 دارد 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 مبنای 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 به استرینگ که در مبنای ۳ هست رو تبدیل به `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؟

+ ساخته نمی‌خواد بدونین.^۶ فقط بدونین ذخیره اعشار برای کامپیوتر ساخته 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^۷

بینین یه وبسایتی هست که پر از سواله و شما می‌تونین برین توش و تمرین کنین. تمرین مهم‌ترین قسمت برنامه‌نویسی هست. اگر می‌خوانین 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.

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.

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

7 <https://quera.org>

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 می‌زاری 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 میدن که از مدت زمان تعیین شده توسط سؤال عبور کرده و Time limit خوردین. اگر مطمئنین 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^۸

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.

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.

۸ ترجیحاً اینو بعد فهمیدن نحوه کار با متغیرها بخونین.

خب گفتیم کامپیوتر فقط ۰ و ۱ می‌فهمه درسته؟ پس اگر ما می‌خواهیم 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 سری گفتن که ما متخصصیم و می‌فهمیم چجور 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 چجور 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 برنامه‌نویسی حواسش هست. هر وقت هم با اون خونه کار داشته باشم، بهش می‌گم X و خود زبون برنامه‌نویسی 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 نتایجش 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 کروشه 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 باهاش 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 گلی 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 اینه بگیم 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. زبونی 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؛^۹

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.¹⁰

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

Correct: WHO (World Health Organization)¹¹

wrong: WHO(World Health Organization)

wrong: WHO(World Health Organization)

تایپ ده انگشتی؛

اگر کارتون با کامپیوتره، سعی کنین تایپ ده انگشتی رو یاد بگیرین. اصلاً چیز سختی نیست. صرفاً با روزی ۱۰ دقیقه تمرین شبانه، مطمئن باشین یاد می گیرین کم کم.

^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.

سایت typing^{۱۲} برای یادگیری.

سایت 10fastfingers^{۱۳} برای تست و تمرین بیشتر

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:۱۴ 5
```

یا مثلاً بهش می‌گیم:

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

Wrong:^{۱۵}

```
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
```

12 <https://www.typing.com/>

13 <https://10fastfingers.com/>

۱۴ مقداری که روی صفحه نمایش چاپ میشه. درواقع چیزی که کامپیوتر به ما خروجی (output) میده.

۱۵ این کد نیست! صرفاً یعنی مثال پایینی اشتباهه.

مثال بیشتر:

```
print(2 * 6)
```

output: 12

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 تمیزتر باشه.^{۱۶}
یه خرده تمرین کنین و علامتای مختلف پایتون رو یاد بگیرین:

operator	Name	توضیحات	مثال	حاصل
+	Addition (جمع)	دو چیز رو با هم جمع می‌کنه	2 + 3	5
-	Subtraction (تفریق)	دو چیز رو از هم کم می‌کنه	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 می‌ریزه دور و فقط قسمت صحیح رو نمایش میده	3 // 2	1
%	Modulus (باقی‌مونده) (mod)	تقسیم دومی بر اولی رو انجام میده و باقی‌مونده تقسیم هرچی بود رو نمایش میده. (باقی‌مونده و خارج قسمت زمان دبستان رو یادتونه؟!)	3 % 2	1
**	Exponentiation (توان)	اولی رو به توان دومی می‌رسونه	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؟!

^{۱۶} اصول تمیز نویسی رو می‌تونین از <https://peps.python.org/pep-0008> بخونین که البته فعلاً براتون پیشرفتن. نگران نباشین! توی مسیر یادگیری خودم بهتون میگمش!

الف) اول پرانتز

ب) دوم توان

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 چیزیه 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 کدمو 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 می‌خوان 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؟! منظورت متنه؟ اگر منظورت 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 به خط بالا.

تمرین ۱-ا:

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

راهنمایی:

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

پاسخ:

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

دیدین؟! صرفاً با دستور پرینت به مستطیل کشیدیم. گفتم اول به سری ستاره چاپ کن. بعدش خط بعدیش بیا اول به ستاره، بعد به سه سری فاصله و بعد به ستاره دیگه چاپ کن. و چندبار دیگه همینو انجام بده. آخر هم به سه 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^{۱۷} 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.

^{۱۷} 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.^{۱۸}

تمیز نویسی:

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
```

18 sum of 5 + 6 is 11

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

قبل بازکردن 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 بگذارین؟! 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.^{۱۹}

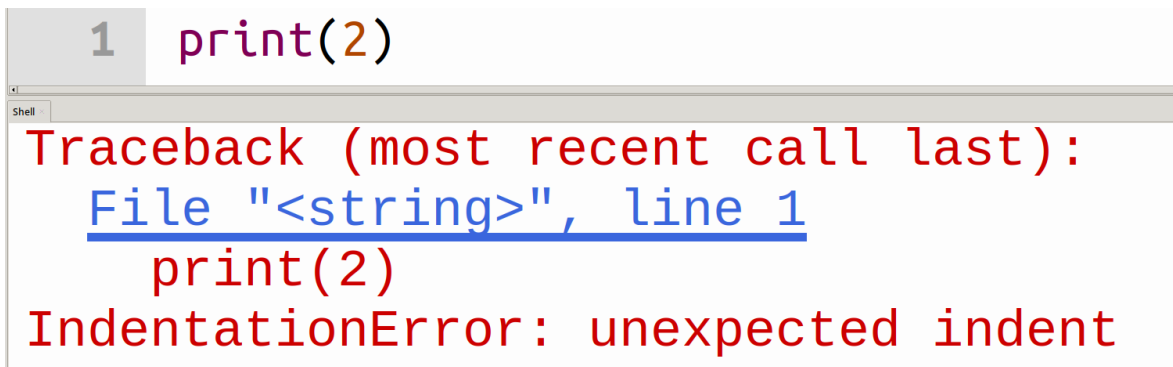
اول هر کد، «\U» قرار می‌گیره و شما کد رو قرار میدین. به جای علامت بعلاوه، اونقدر صفر می‌ذارین که هشت رقم شه. بعدش چاپش می‌کنین!^{۲۰} مثلاً:

```
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 می‌خوریم.

¹⁹ <https://unicode.org/emoji/charts/full-emoji-list.html>

^{۲۰} 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 و از ۱۰۰ به بعد توی کانال یوتیوبش.

2. Variable Naming & Working with variables

ببینیم ما در برنامه‌نویسی گاهی می‌خوایم 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».

این‌ا 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 میگویم 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 ے یاتوں ے:

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»^{۲۱}.

به اعداد اعشاری می‌گن «float»^{۲۲}.

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^{۲۳}. 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!)

21 ..., -3, -2, -1, 0, 1, 2, 3, ...

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

23 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 منها!

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 پرینتن 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.^{۲۴} (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
```

^{۲۴} 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 محتوای اون 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 پرسین که مشکلش چیه. کد زیر رو می‌بین به طرف نشون میدین و مثلاً می‌پرسین مشکلش 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^{۲۵} رو ببینم، حسم 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:

^{۲۵} 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.^{۲۶}
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! ٩٠٩٠٩

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).

^{۲۶} 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 راحتیه 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:

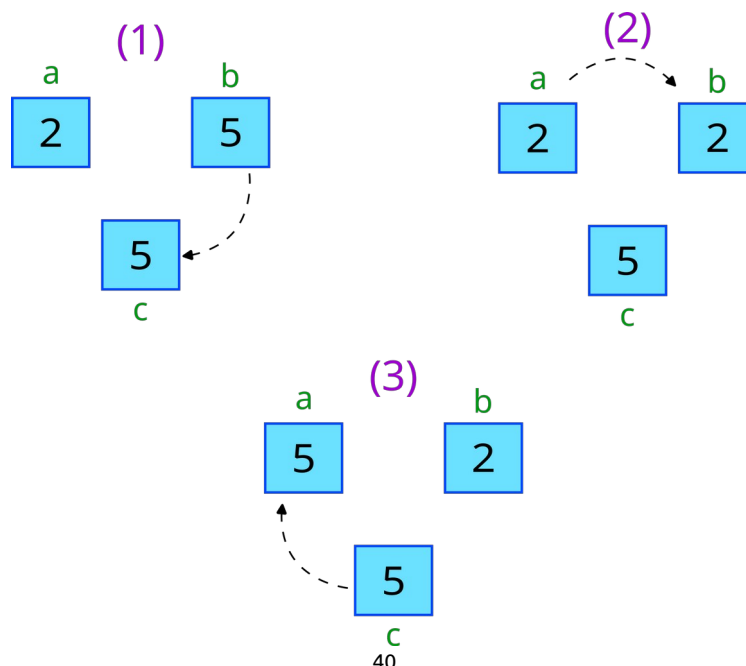
یکم راهنماییتون می کنم. فرض کنین دو جعبه دارین که توی هر جعبه یه توپ هست. می خواین توپ جعبه اول بره جعبه دوم و توپ جعبه دوم بره جعبه اول. همچنین نمیشه دو توپ توی یه جعبه باشن. این زمان چیکار می کنین؟

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
temp = b
b = a
a = temp
```

درواقع من می‌گم متغیر اولی رو توی یه متغیر دیگه به صورت موقت بذار. اسمش با توجه به کاربرتش 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.^{۲۷}
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}')
```

^{۲۷} 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. درواقع اولین چیز داره میگه تو چی رو دادی و دومین چیز، درواقع چیز اصلی هست که به وسیله دستور پرینت چاپ شده. چون شما خط بعدی گفتی که اون متغیر رو چاپ کن. یعنی اگر ما پرینت انجام ندیم، دومی چاپ نمیشه:

```
name = input()
```

درواقع ما فقط یک خروجی داشتیم. اونم خروجی بود که دستور پرینت داده بود. وگرنه اولی که فقط داشت نشون میداد که ما چی چاپ کردیم.

خب بیایم یکم مکانیزم ورودی گرفتن رو بهتر کنیم. پایتون خودش گفته وقتی می‌خوای ورودی بگیری، توی پرانتز می‌تونی متنی که می‌خوای 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. تبدیلیش 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 در 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^{۲۸} کردن:

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 نوشتنم که فکر کنین روشن):

```
age = 18.9
```

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

```
print(age)
```

output²⁹

```
age = 18
```

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

```
print(age)
```

output³⁰

```
age = '18'
```

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

```
age = age + 2
```

```
print(age)
```

output³¹

```
age = '18'
```

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

```
age = age + 2
```

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

output³²

۲۸ کلمه cast یعنی به شکل یه چیز در آوردن. (درواقع ما می‌گیم به شکل int در بیار)

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

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

29 Output: 18

30 Output: 18.0

31 Output: 20

32 Output: 40

```
age = int('18')
age = float(age)
print(age + 2)
```

output³³

تمرین!

حالا می‌خوایم به سن ورودی بگیریم. سن رو ۲ تا زیاد کنیم و بعدش چاپش کنیم. (په کوشولو! اول فوریت بنویس و بعدش پاسفو بنون!)

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 می‌خوانیم؟ بریم به راه ساده‌تر:

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

همیشه پرانتز رو از داخلی‌ترین بخونیم:

یه ورودی بگیر؛ ورودی رو که گرفتی، تبدیلش کن به عدد صحیح و بعدش بریز داخل age.^{۳۴}

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

مثلاً این تبدیلا توی 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.

33 Output: 20.0

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

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

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 می کنم:

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

پاسخ:

```
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!

یه نکته! از من به شما نصیحت که برای حل سؤالات عادی (و نه یه پروژه!)، همیشه تمام ورودی‌ها رو در یک جا بگیرین. یعنی اینطور نباشه که به هم ریخته باشه. یکی یه جای کد باشه، یهو وسط ده خط عادی، یه ورودی دیگه گرفته باشین. این باعث میشه که بعداً گمشون کنین و نتونین برای بررسی کد پیداش کنین.

تمرین:

۱.۱- برنامه‌ای بنویسین که چهار عدد گرفته و میانگین اونها رو حساب کنه.

۱.۲- حالا برنامه رو جوری بنویسین که فقط از دو متغیر استفاده کنه. فقط و فقط دو متغیر!

پاسخ‌ها:

روش اول:

```
num1 = int(input())
num2 = int(input())
num3 = int(input())
num4 = int(input())
print((num1 + num2 + num3 + num4)/4)
```

اول چهار عدد رو می‌گیریم و بعدش جمعشون رو تقسیم بر ۴ می‌کنیم. جمع چهارتا رو هم گذاشتیم داخل پرانتز که با توجه به اولویت ریاضی، حاصل درست حساب بشه.

روش دوم:

```
num1 = int(input())
num2 = int(input())
num3 = int(input())
num4 = int(input())
summ = num1 + num2 + num3 + num4
avg = summ / 4
print(avg)
```

اول جمع رو ریختم داخل یه متغیر. (summ) بعد حاصل میانگین رو ریختم توی یه متغیر دیگه (avg). بعدش متغیر میانگین رو چاپ کردم. اینطوری خواناتره.

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

5. if

خب این قسمت از اون قسمتهای مورد علاقه من هست. چون کم‌کم حس واقعی برنامه‌نویسی رو درک می‌کنین!

قبلش یه خرده با علامتهای 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	$35 \geq 2$	بزرگتر یا مساوی ۲ (دقت ۲ کنین. علامت بزرگتر یا

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

مساوی هست! پس عبارت درستیه که بنویسیم ۲ بزرگ‌تر یا مساوی ۲ است)			
سه کوچکتر یا مساوی ۶	$3 \leq 6$	کوچکتر یا مساوی	\leq
مساوی هست با ۳ ۳	$3 == 3$	مساوی بودن	$==$
نامساوی ۱۰ ۵	$5 \neq 10$	نامساوی بودن	\neq

فرض کنیم شما می‌خواهید پرداخت بانکی انجام بدین. شما مبلغ رو وارد می‌کنید؛ رمز عبور رو می‌زنید و منتظر پرداخت شدن می‌مونید.

در پشت صحنه این اتفاق می‌وفته:

اگر موجودی حساب بیشتر یا مساوی مقدار تراکنش است، پرداخت رو انجام بده.
درواقع ما توی زبان‌های 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 بنویسیم:

Code:

```
# Code 1
if 4 >= 0:
    print('Hi')
```

output:

```
Hi
```

میگه اگر ۴ بزرگ‌تر یا مساوی از صفر بود، کارهای زیر رو برام انجام بده. این کارها رو با دونه نقطه و چهارتا فاصله برات مشخص می‌کنم. اینجا صرفاً یه پرینت انجام داده بودم. اما میشه هرچقدر دستور می‌خواهیم بنویسیم:

```
# Code 1
if 4 >= 0:
    print('Hi')
    print('Hello')
```


بیایم بازم مثال ببینیم:

```
# Code 1
if 3 > 2:
    print('Code 1')

# Code 2
if 2 == 2:
    print('Code 2')

# Code 3
if 3 != 2:
    print('Code 3')

# Code 4
if 3 >= 2:
    print('Code 4')

# Code 5
if 2 <= 2:
    print('Code 5')

# Code 6
if 2 < 1:
    print('Code 6')

# Code 7
if 2 != 2:
    print('Code 7')

# Code 8
if 2 >= 3:
    print('Code 8')
```

خب به نظرتون در کدوم کدها، پرینت اجرا میشه؟ قاعدتاً هر کدوم که حاصل جلوی if درست شد، اجرا میشن.

- کدوما درسته؟ (پاسخ پایین صفحه)^{۳۶}

کد ۱: چون ۳ بزرگتر از ۲ هست، عبارت جلوی if میشه درست، پس اجرا میشه. (همیشه سعی کنین اینطوری بخونین «اگر ۳ بزرگتر از ۲ بود؟» اگر جواب بله بود، عبارت/عبارتهایی که در زیر if هستن و با چهارتا فاصله ازش مشخص شدن، اجرا میشن.)

کد ۲: چون ۲ مساوی با ۲ هست، عبارت جلوی if صحیحه پس اجرا میشه.

کد ۶: چون ۲ کوچکتر از ۱ نیست، عبارت جلوی if میشه غلط. پس اجرا نمیشه.

کد ۷: چون ۲ نامساوی ۲ نیست، عبارت جلوش میشه غلط. پس اجرا نمیشه.

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

Code:

```
if 2 > 3:  
    print('hi')
```

Output:

```
Traceback (most recent call last):  
  File "<string>", line 2  
    print('hi')  
    ^^^^^  
IndentationError: expected an indented block after 'if' statement  
on line 1
```

ارورهای مرتبط با «indent» یعنی فاصله‌گذاری رو درست رعایت نکردین. پس حواستون باشه که هم دو نقطه آخر if و هم چهارتا فاصله‌ای که زیرش می‌خوان چیزها رو اجرا کنین، الزامیه. پایتون یه برنامه‌س که کدهای شما رو می‌خونه و اجراشون می‌کنه. اگر دقیق براش همه چیز رو عیناً اون چیزی که می‌فهمه ننویسی، به مشکل و ارور بر می‌خوره.

من اینجا صرفاً با عدد خالی مثال زدم که ساده‌تر باشه. ولی معمولاً ما شرطامون رو بر حسب متغیرها می‌نویسیم. یعنی اینطوری:

```
# Code 1
num1 = 2
num2 = 10
if num1 > num2:
    print('num1 is greater')

# Code 2
num1 = 12.5
num2 = 12.5
if num1 == num2:
    print('equal')

# Code 3
s1 = 'hello'
s2 = 'hello'
if s1 == s2:
    print('equal')

# Code 4
num = int(input('Enter a number: '))
if num % 2 == 0:
    print('Yes')

# Code 5
num1 = 3
num2 = 10
if num1 == 3:
    num2 = 5
print(num2)

# Code 6
s1 = 'z'
if s1 > 'h':
    print('Code 6')

# Code 7
s1 = 'hello'
s2 = 'hz'
if s1 < s2:
    print('Code 7')
```

کد ۱: اگر عدد اول بزرگ‌تر از دومی بود، چاپ کن یه چیزی رو. این انجام نمیشه. چون ۲ از ۱۰ بزرگ‌تر نیست. (یادتونه گفتم که همیشه اینطوری بخونین «اگر num1 بزرگ‌تر از num2 بود»)

کد ۲: اگر num1 برابر با num2 بود... . بله اجرا میشه. چون برابرن.

کدر۳: بله! ما می‌تونیم بگیم اگر string1 با string2 برابر (عینا یکسان) بود، فلان کار کن. چون برابرن پس اجرا میشه.

کدر۴: یه عدد گرفتیم. گفتیم اگر باقی‌موندش بر ۲ شد صفر، بنویس Yes. به نظرتون این شرط ممکنه به چه کاری بیاد؟ پاسخ پایین صفحه^{۳۷}

کدر۵: می‌گه اگر num1 برابر با ۳ بود... کارهای زیر رو انجام بده. بله برابر هست و برای همین، num2 برابر با ۵ میشه.

بله! صرفاً نیاز نیست توی if صرفاً پرینت انجام بدیم. می‌تونیم هر کاری رو انجام بدیم! هر دستوری که عادی به کار می‌بردیم رو می‌تونیم توی if هم به کار ببریم! مشکلی نداره! همونطور که دیدین، توی کد ۵ من گفتم که اگر این شرط برقرار بود، متغیر num2 از این به بعد برابر ۵ باشه. چون شرط برقرار بود، مقدار متغیر از ۱۰ شد ۵.

کدر۶: فعلاً ما با علامتای بزرگ‌تر و اینا برای استرینگ کار خاصی نداریم. صرفاً آوردیم که بدونین که نحوه کارش چیه.

خلاصه بگم که طبق حروف الفبا، حرف «z» از «h» دیرتر میاد. توی پایتون هم این معنا میشه که بزرگ‌تره. پس درواقع:

$$a < b < c < d < e < f \dots < x < y < z$$

کدر۷: بین دوتا استرینگ اگر بخواد مقایسه انجام شه، ابتدا از حرف اول شروع میشه. چون اینجا هردو حرف اولشون h هست، پس می‌گه خب حرف اول یکسانه. بریم سراغ حرف دوم. حرف دوم کدوم بزرگ‌تره؟ خب «z» بزرگ‌تر از «e» هست. پس درنتیجه s2 بزرگ‌تر از s1 هست. پس کد اجرا میشه.

حالا یه مثال دیگه:

```
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).^{۳۸}

- چه کارهایی؟

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

^{۳۷} برای اینکه بفهمیم آیا یه عدد زوج یا نه، می‌تونیم باقی‌مانده‌شو بر ۲ بگیریم. اگر صفر شد، یعنی زوج. اگر ۱ شد یعنی فرد. این خیلی مهمه. کلی در آینده باهاش کار داریم.

^{۳۸} حواستون باشه که دو نقطه بچسبه به کرکتر قبلیش و فاصله نداشته باشه!

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
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 یه زبون. می‌گه نحوه نوشتاری که پایتون می‌فهمه رو رعایت نکردی. می‌گه من انتظار داشتم (expect) یه 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 یه بک اسلش بذار که بفهمم منظورت خودشه و با کوتیشن‌های طرفین اشتباهش نگیرم. بک اسلش یه کرکتر ویژه هست برای نشون دادن چیزها به پایتون

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

- عه! خب اگر بخوام توی متنم (string ام) بک اسلش به کار ببرم چیکار کنم؟
+ دوتا بک اسلش پشت هم بذارین. اینطوری می‌فهمه منظورتون بک اسلش 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

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

دو مورد که نیاز دارم بلدش باشین:

\n معنای اینکه برو خط بعدی داره.

\t معنای اینکه به اندازه دکمه «Tab» روی کیبورد فاصله بده، داره. (معمولاً یه دونه تب برابر ۴ تا ۸ تا فاصله هست).

مثال:

```
print('a\nb')
```

output:

```
a
b
```

بهش گفتم که اول «a» رو چاپ کن و بعدش یه اینتر بزن (برو خط بعدی) و بعدش «b» رو چاپ کن.

```
print('a\tb')
```

output:

```
a      b
```

بهش گفتم که اول «a» رو چاپ کن و بعدش یه تب بزن و بعدش «b» رو چاپ کن.

```
print('a\n\n\tb')
```

output:

```
a
      b
```

بهش گفتم که اول «a» رو چاپ کن و بعدش دوتا اینتر بزن و بعدش باز یه تب بزن و بعدش «b» رو چاپ کن.

یکی از چیزایی که می‌خواستم توی این آموزش رعایت کنم، یادگیری با کاربرد بود. یعنی نیومدم کل قواعد string رو موقع معرفی‌ش بگم. چون واقعاً شیوه درستی نیست! حداقل اینطوری من یاد نمی‌گیرم. چون یه 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) ۲)

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")
```

خواستون باشه که توی if، برای چک کردن مساوی بودن، دوتا مساوی می‌گذاریم. 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» یا «اگر نه». 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! ولی چیزایی که قراره در صورت رفتن سراغ else، اجرا شن، چهارتا فاصله دارن.

توجه! دقت کنید `else` هیچ شرطی چک نمی‌کند (هیچ شرطی رو جلوش نگذارید!) و تنها زمانی اجرا میشه که هیچ `if` و `elif` ای اجرا نشده.

بیایم یکم دیگه پیچیده‌ترش کنیم.

فرض کنیم 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. خوش‌آمدید.

```
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 به 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 به بعدش به `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`.

توجه ۱: ما می‌تونیم بعد یک `if`، بی‌نهایت `elif` داشته باشیم، ولی در نهایت حداکثر به `else` خواهیم داشت. (می‌تونیم `else` رو هم اصلاً نذاریم!)

توجه ۲: ابتدا سراغ `if` میریم. اگر برقرار بود که از تموم `elif` ها یا `else` نهایی زیرش چشم‌پوشی می‌کنیم و اصلاً نگاهی‌مون نمی‌کنیم! اگر `if` برقرار نبود، میریم اولین `elif` بعدش. اونو چک می‌کنیم. اگر برقرار بود، که از تمام `elif` های دیگه زیرش و `else` چشم‌پوشی می‌کنیم. اگر نه، میریم سراغ `elif` بعدی. خلاصه تا پایان میریم. اگر هیچ‌کدوم از `if` و `elif` ها برقرار نبود، `else` نهایی اجرا میشه..

تمرین!

۱- برنامه‌ای بنویسین که به 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) → B³⁹

[12, 14) → C

[10, 12) → D

[0, 10) → F

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

۵- فرض کنید که سیستمی دارین 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»، اگرنه، 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

۳۹ یعنی درواقع ۱۴ تا ۱۵.۹۹. پرانتز یعنی خود ۱۶ حساب نشه. کروشه یعنی بشه.

output:⁴⁰

Blocked

input:

hi

Hello

1234

output:⁴¹

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:

1

4

9

3

output:

9 1

input:

9

-1

10

10

output:

10 2

input:

3

3

4

1

output:

4 1

۴۰ ۳ بار تلاششو کرد و همش اشتباه! بگین Blocked.
۴۱ بار سوم درست زد و اوکیه.

input:

3
3
3
3

output:

3 4

۷.۱- فرق دو کد زیر چیه؟

```
age = int(input())

# Code 1
if age > 18:
    print('over 18')
elif age == 18:
    print('under 18')

# Code 2
if age > 18:
    print('over 18')
if age == 18:
    print('under 18')
```

۷.۲- 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 میزان هزینه کل بنزین مصرفی به اسلوب.

input:

41

0

output:⁴²

61500

input:

125

40

output:⁴³

225000

پاسفنامه:

پاسخ ا:

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

```
if age > 18:
```

۴۲ 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:
 $41 * 1500 = 61500$

۴۳ مصرف ۱۲۵ لیتر. مقدار اضافه‌مونده از ماه قبل، ۴۰ لیتر. پس ما درواقع $40 + 60 = 100$ لیتر این ماه سهمیه داریم. پس ۱۰۰ لیتر رو با سهمیه ۱۵۰۰ اسلوبی و ۲۵ لیتر دیگر رو با سهمیه ۳۰۰۰ اسلوبی می‌دیم. یعنی:

$100 * 1500 + 25 * 3000 = 225000$

```

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، کوچکتر. پس اگر دوتای قبلی نبود، قطعاً کوچکتر 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 ازش 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 گرفتم ولی الان یه اینتر میزنه و ورودی رو در خط بعد می گیره. شاید گاهی این چیزا تمیزتر به نظر بیاد! راستی! علامت دونقطه به کلمه قبل می چسبه و از کلمه بعدیش فاصله می گیره. (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 می چسبه به دو نقطه و این زبا نیست!

پاسخ ۲:

این سؤال رو با عمد آوردم که شما رو با عملیات های 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 (در غیر 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 شه ولی ببینه 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 نیاز به 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 چک‌کردنی، عدد سومی رو چاپ می‌کنیم.

اگر متوجه نشدین، روی کد فکر کنین 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 می‌خواین مطمئن شین فهمیدین، دو روز بعد سعی کنین همین الگوریتم رو خودتون بنویسین. اگر تونستین بنویسین، یعنی درسته فهمیدین ولی اگر نتونستین، یعنی نفهمیدین!

برنامه‌نویسی مثل ریاضیات! نمیتونین با خوندنش بگین عه فهمیدم. صرفاً فکرتونه که فهمیدین. ولی در اصل نفهمیدینش.

از من به شما نکته!

اگر می‌خواین برنامه‌نویسیتون قوی **نشه**، صرفاً بخونین و رد شین و تلاش برای حل **نکنین**! پاسخا رو خودتون بخونین و تلاشی برای حل موضوع نکنین!
یا اگر خواستین تلاش کنین، دو دقیقه شد و نتونستین، دست از تلاش بکشین و پاسخ‌نامه رو بخونین!
این کار تضمین می‌کنه که شما برنامه‌نویسیتون 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)، می‌تونیم دو یا چند تا شرط رو همزمان توی یک if چک کنیم.

می‌تونیم بگیم اگر فلان چیز و (and) فلان چیز برقرار بود، فلان کار کن.
پایتون گفته من کارتون رو راحت می‌کنم! مثل زبون گفتاری که می‌گی 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	<code>if x > y and x > z</code>	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	<code>if x > y or x > z</code>	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	<code>if not (x > y)</code>	اگر نبود X بزرگ‌تر از Y (یعنی X کوچکتر یا مساوی 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 قشنگ‌تر نشد؟

نکته: **and** صرفاً زمانی درست میشه که تمام عبارتها درست باشن.

نکته: **or** حتی اگر یکدونه هم درست باشه، اجرا میشه. فقط یه دونه از عبارات درست باشن هم اوکیه.
چه زمانی اجرا نمیشه؟ زمانی که همه عبارات اشتباه باشه.

code 1:

```
num1 = 1
num2 = 2
num3 = 3
num4 = 4
if num4 >= num3 and num4 >= num2 and num4 >= num1:
    print('num4 is the largest')
```

اجرا می‌شود یا نه؟ پاسخ در پایین صفحه.^{۴۴}

code 2:

```
num1 = 1
num2 = 2
num3 = 7
num4 = 4
if num4 >= num3 and num4 >= num2 and num4 >= num1:
    print('num4 is the largest')
```

اجرا می‌شود یا نه؟ پاسخ در پایین صفحه.^{۴۵}

code 3:

```
if 1 > 0 or 2 > 5 or 3 > 5:
    print('hi')
```

اجرا می‌شود یا نه؟ پاسخ در پایین صفحه.^{۴۶}

code 4:

```
if (1 > 0 and 1 < 0) or (5 == 5):
    print('hi')
```

اجرا می‌شود یا نه؟ پاسخ در پایین صفحه.^{۴۷}

یا حتی برای حل سوال، 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؟

^{۴۴} بله! زیرا and صرفاً زمانی اجرا می‌شود که همه عبارات درست باشند. خوب همیشه هم درسته. پس اجرا می‌شه.
^{۴۵} خیر! زیرا and صرفاً زمانی اجرا می‌شود که همه عبارات درست باشند. اما همیشه درست نیست که! num4 از num3 کوچکتره. یکیش هم که غلط باشه، اجرا نمیشه.
^{۴۶} بله! زیرا در or، حتی یک عبارت هم درست باشد، اجرا می‌شود.
^{۴۷} بله! پرانتزگذاری خیلی میتونه بهمون کمک کنه. درواقع می‌گه که or بین دو پرانتز هست. حاصل یکی از این پرانتزها هم درست بشه، پرینت صورت می‌گیره. چون دومی درسته، پرینت انجام می‌شه. هر وقت خواستین که یه عبارتی که توش and و or هست رو بسازین، از پرانتز برای درست‌موندن اولویت و خوانایی استفاده کنین

+ اول به صورت منطقی بررسی 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 برنامه نویس، تهیه برنامه ای 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 می تونستین 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.

پایتون قشنگه (:)

پایتون زبونی که گفته بیا من کارهات رو ساده می‌کنم. من کلی دستور از پیش تعریف‌شده دارم که بتونی کارهات رو ساده انجام بدی. مثلاً این سؤالی که ما حل کردیم این بود که ماکسیموم رو پیدا کن. پایتون دقیقاً به دستور داره به همین نام:

code:

```
num1 = 4
num2 = -19
num3 = 20
num4 = 0
print(max(num1, num2, num3, num4))
```

output:

```
20
```

یعنی چاپ کن حاصل max این چهارتا عدد رو. (هر چندتا عدد رو بخوایم می‌تونیم بهش بدیم. اینجا من برای مثال ۴ تا عدد دادم.)
به همین سادگی و خوشمزگی :)
تازه پایتون دستوری برای پیدا کردن مینیموم هم داره:

```
num1 = 4
num2 = -19
num3 = 20
num4 = 0
print(min(num1, num2, num3, num4))
```

output:

```
-19
```

ولی شما برای اینکه یاد بگیرین، از این دستورات آماده استفاده نکنین. فعلاً برای یادگیری، خودتون کدشو بنویسین. ولی در آینده توی کاراتون، از دستورات آماده استفاده کنین. (چون سریع‌تر و قابل‌اعتمادتر از کدایی که شما می‌نویسین هستن!)

پاسخ ۳:

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

روش دوم:

سعی کنین با کمک not پیاده‌سازیش کنین!

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

بعدهش برای not، خواستم بگه نات عبارت داخل پرانتز. 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 بگیم 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 می گیرم. چک می کنم ببینم درسته یا نه. اگر نبود یه دونه به تعداد تلاشا باید اضافه کنم دیگه. بعدش دوباره توی همین if ورودی می گیرم. 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 می گیرم. این ورودی بار آخره! حالا با یه if نهایی، چک می کنم اگر ایندفعه هم درست نبود، خب پس بلاکه. وگرنه میگیرم Welcome.

کد ما توی ۳ تلاشی که فرصت داره، اولین ۱۲۳۴-ای که ببینه! 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 می بینه که درست نیست و دوباره ورودی می گیره. اما توی if دومی، می بینه 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 می کنه Welcome.

```
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 بذاریش 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 مکزیمم 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 ببین 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 ۷.۱:

کد ۱: اگر if اجرا شه، دیگه elif اجرا نمیشه. پس صرفاً زمانی میره سمت elif که if اجرا نشده باشه.
کد ۲: فرقی نداره که if اولی اجرا شه یا نشه. if دوم مستقل از اولی هست. پس سمت دومی هم میره.
- حالا به نظرتون کدوم کد بهتره؟

+ مسلماً کد اولی. چون تنها در صورتی میره سمت شرط دومش (elif) که if اجرا نشده باشه. اما کد دومی در هر صورت میره. خب وقتی سن بزرگ‌تر از ۱۸ باشه، قطعاً برابر با ۱۸ نیست که! پس الکی داریم دوباره چک می‌کنیم که آیا برابر ۱۸ هست یا نه. نیاز نبود و الکی با چک کردن یک شرط، باعث میشیم که برنامه کمی کندتر شه. (چون تک‌تک دستورات زمان می‌برن تا اجرا شن. ما الکی داریم یه شرط رو چک می‌کنیم. همین یه کوچولو سرعت رو پایین میاره.) پس از اضافه‌کاری خودداری کنیم.

پاسخ ۷.۲:

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
```

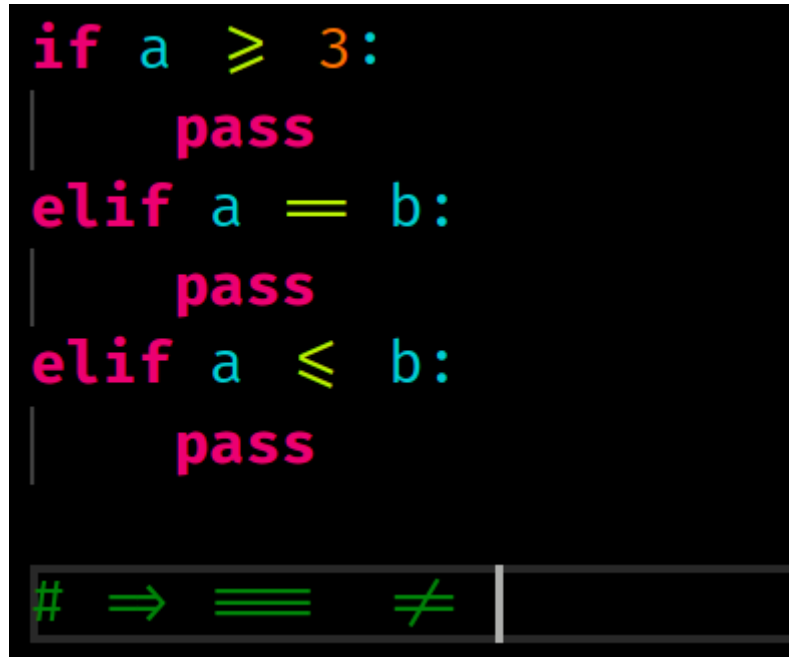
```
total_quota = add_quota + monthly_quota
```

```

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 کنین.

برای رفتن به فایل json (تنظیمات code editor)، روی کنترل کاما بزنین و بعد بالا سمت راست

دومین دکمه کنار سه نقطه، یه گزینه هست که اگه روش بگیرین نوشته: Open Settings (JSON)

• Nested if

ما می‌تونیم توی یه if، یه if دیگه داشته باشیم! یعنی بگیم اگر اومدی توی این if، می‌تونیم یه if دیگه هم داشته باشیم! 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 زوجه؟ اگر زوج بود، پرینت کن even. بعدش بیا بیرون از if دومی. اما حواستون باشه که هنوز توی if اولی هستین 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 نذارین:

```

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 خارجی ترین if می‌گیم if لایه اول. و به موازات اون، هر لایه اضافه شه، میشه مثلاً if لایه دوم. مثلاً اینجا ما دو لایه داریم. خارجی ترین if لایه اول هست و داخلی ترین، 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 یکسان (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 ببینیم یه عدد حداقل ۲ رقمی هست، باید تقسیم صحیحش بر ۱۰ بزرگ‌تر از ۰ شه.
برای اینکه ببینیم یه عدد حداقل ۳ رقمی هست، باید تقسیم صحیحش بر ۱۰۰ بزرگ‌تر از ۰ شه.

پاسخ:

```
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 کنین شما می‌خوااین 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 همچین 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 زدین. دیگه کار نمی‌کنه! اما همونطور که ما می‌دونیم، توی وبسایت‌ها که اینطور نیست! یعنی اگر پسورد رو اشتباه بزنیم، باز از ما پسورد ی‌گیره 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 بگیم 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 میخوایم 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 می‌کنه و بعد می‌گذاره توی number. (یعنی درواقع number ما برابر ۲ میشه. بعدش چاپ می‌کنه Hello.

دوباره میره بالا و دوباره شرط رو چک می‌کنه.

- 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 می‌کنه که آیا ۲ بزرگ‌تر از ۰ هست؟ بله! پس دوباره کارها رو انجام میده. یعنی `number` رو یک می‌کنه و بعد چاپ می‌کنه `Hello`.

حالا چک می‌کنه که آیا ۱ بزرگ‌تر از ۰ هست؟ بله! پس دوباره کارها رو انجام میده. یعنی `number` رو صفر می‌کنه و بعد چاپ می‌کنه `Hello`. یعنی `number` رو یک می‌کنه و بعد چاپ می‌کنه `Hello`. حالا چک می‌کنه 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 می‌گیرن، توی `while` اجرا میشن. ولی ما می‌خوایم بعد تموم‌شدن `while`، کلمه `End` چاپ شه.

درواقع دیدین؟ ما می‌تونیم یه عدد بگیریم و به تعداد اون عدد، یه سری کار رو انجام بدیم. این یکی از معروف‌ترین ساختارهاست. اینکه یه عدد بگیریم و بگیریم تا وقتی صفر نشده، یه سری کار کن. هر بار هم یکی از عدد کم می‌کنیم. می‌تونیم 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 کنید و روند رو دنبال کنید 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

ببینیم:

```
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 رایجه میگن «حلقه بی‌نهایت» یا «infinite loop». اگر

برنامتون خیلی طول کشید و یا صدای فن لپ‌تاپتون بلند شد، احتمالاً توی لوپ بی‌نهایت افتادین!^{۴۸} با `ctrl + c` می‌تونین متوقفش کنین. یا خود IDE ها دکمه توقف یا `stop` توی منوی بالا دارن.

به راه کاربردی برای تشخیص اینکه برنامه کجای لُگِیر 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('-----')
```

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

این تکنیک پرینت خیلی کاربردی. مثلاً می‌خوان 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 ۱:

^{۴۸} شاید برنامتون رو خیلی بد نوشتین و برای اعداد بزرگ خیلی طول میکشه. به این بعداً می‌رسیم. ولی فعلاً همون چیزی که گفتم رو بدونین.

هرکدام از while های زیر چند بار اجرا میشن؟ و در هر بار اجرا شدن، مقدار i چی هست؟

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 ۱:

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 الآن ۲ راه یاد گرفتیم برای اینکه یه حلقه رو `n` بار تکرار کنیم.

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

مثال ۲:

می‌خوایم جمع اعداد یک تا ۱۰۰ رو چاپ کنیم.
راهنمایی: از این شروع کنیم که اعداد ۱ تا ۱۰۰ رو توی `while` به وجود بیاریم.

پاسخ ۲:

خب قبول داریم که کد ساخت اعداد ۱ تا ۱۰۰ اینه:

```
i = 1
while i <= 100:
    i = i + 1
```

خب حالا من مگه قرار نیست اعداد ۱ تا ۱۰۰ رو جمع کنم؟ هر بار یکی از اعدادی که می‌خوام جمع بزنم توی `while` ساخته میشه. پس من می‌تونم جمع رو بشکونم به اینکه در هر مرحله، عدد ساخته‌شده رو با حاصل قبلی جمع می‌کنم. برای این کار نیاز به یه متغیری (ظرفی) دارم که جمع تا اون مرحله رو توش ذخیره کنم. (جمع تجمعی)

```
i = 1
sum_ = 0
while i <= 100:
    sum_ = sum_ + i
    i = i + 1
print(sum_)
```

مثال ۳:

یه پسورد از کاربر می‌گیریم. می‌خواهیم تا وقتی که پسورد برابر با استرینگ «۱۲۳۴» نبود، بهش بگه پسوردتو اشتباه زدی و دوباره ازش پسورد بخواد. این کار رو تا وارد کردن پسورد درست (یعنی «۱۲۳۴») ادامه بده.

پاسخ ۳:

```
password = input()
while password != '1234':
    print('Wrong Password')
    password = input()
print('Welcome')
```

یه پسورد می‌گیرم. می‌گم تا وقتی که برابر استرینگ «۱۲۳۴» نبود، میندازمش داخل یه while که بگم وایسا تا دوباره ازت پسورد بخوام. تا وقتی پسورد اشتباه بزنه، همینجا باقی می‌مونه و نمیتونی وارد سیستم شی. هر وقت پسورد رو درست وارد کرد، while به پایان می‌رسه.

مثال ۴:

کوروش و داریوش دارن با هم یه بازی انجام میدن. کوروش یه عدد توی ذهنش انتخاب کرده (مثلاً ۵۰) و از داریوش می‌خواد که اونو حدس بزنه. داریوش هر بار یه حدس می‌زنه. اگر حدسش درست بود، چاپ میشه «Correct». اگر حدسش بزرگ‌تر از عددی که کوروش در نظر گرفته بود، بود، کوروش بهش میگه «too high». اگر کوچکتر بود، بهش میگه «too low». این مکانیزم رو پیاده‌سازی کنید.

پاسخ ۴:


```

num = 50
guess_num = int(input())
while guess_num != num:
    if guess_num > num:
        print('too high')
    else:
        print('too low')
    guess_num = int(input())
print('correct')

```

اول یه عدد می‌گیره. بعد می‌گه تا وقتی عدد گرفته‌شده، برابر با عدد توی ذهن کوروش نبود، چک کنه ببینه اگر بزرگ‌تر بود، بگه «too high»، وگرنه (کوچک‌تر بود)، بگه «too low». بعد یه عدد دیگه بگیره. که ببینه ایندفعه چی؟ ایندفعه آیا درست میشه؟ همونطور که می‌دونیم بعد گرفتن عدد جدید، چون رسیده به انتهای `while`، میره بالا و شرط رو دوباره چک می‌کنه. اگر شرط برقرار بود (یعنی عدد برابر)، توی `while` می‌مونه. وگرنه (عدد برابر) از `while` خارج میشه و عبارت «correct» چاپ میشه.

مثال ۵:

حدس کولاتز یه مسأله ریاضی هست که می‌گه ما از هر عددی شروع کنیم، می‌تونیم با انجام یه الگوریتم، به عدد ۱ برسیم. به این شکل که اگر عدد فرد بود، ضربدر ۳ + ۱ می‌کنیم. اگر زوج بود، تقسیم بر ۲ می‌کنیم. این کار رو تا رسیدن به عدد ۱ تکرار و ادامه می‌دیم. مثلاً اگر از ۳ شروع کنیم، اینطوری میشه:

3 → 10 → 5 → 16 → 8 → 4 → 2 → 1

برنامه‌ای بنویسین که با دادن یک عدد، مراحل رسیدن به عدد ۱ رو نشون بده. مثل بالا.

input 1:

3

output 1:

10
5
16
8
4
2
1

input 2:

7

output 2:

22
11
34
17
52
26
13
40
20
10
5
16
8
4
2
1

پاسخ ۵:

```
n = int(input())
while n != 1:
    if n % 2 == 0:
        n = n // 2
    else:
        n = 3*n + 1
    print(n)
```

while تا کجا باید ادامه پیدا کنه؟ تا وقتی که عدد ورودی به ۱ نرسیده باشه. اگر نرسیده بود، اگر زوج بود، n جدید ما، n تقسیم بر ۲ میشه. وگرنه (فرد باشه)، n جدید، ۳ برابر n قبلی ضربدر ۱ هست. هر بار هم n هر مرحله رو چاپ می‌کنم.

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 ضررزدن به خودتونه!

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

۱.۱- به شما دو عدد صحیح a و b داده میشه. برنامه‌ای بنویسین که عدد a رو به توان b برسونه. تضمین میشه که b عددی بزرگ‌تر یا مساوی صفر است.

input 1:

2
3

output 1:

8

input 2:

3
3

output 2:

27

input 3:

10
0

output 3:

1

۱.۲- برنامه‌ای بنویسین که ب.م.م (بزرگترین مقسوم‌علیه مشترک) دو عدد صحیح مثبت رو حساب کند.

input 1:

5
10

output 1:

5

input 2:

20
30

output 2:

10

input 3:

10
10

output 3:

10

۲- برنامه‌ای بنویسین که یک عدد به عنوان تعداد نمره‌ها 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 بگیرد و بعد گرفتن هر عدد، جمع اعدادی که تا حالا بهش دادیم رو بهمون بده.

یعنی:

input:

1

2

3

output:⁴⁹

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` گرفتم که پیام جمع رو تا اون لحظه داخلش ذخیره کنم. اینطوری می‌تونم دنبال کنم که تا این لحظه جمع چیه. 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

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

```
18
```

```
9
```

```
-1
```

```
output:
```

```
16.75
```

```
input:
```

```
-1
```

```
output:50
```

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 عدد ماکزیموم و مینیموم رو در کنار هم چاپ کنه. ($-1 \leq n \leq 1000$)
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:

100 -20

input:⁵¹

4

0

0

0

0

output:

0 0

input:⁵²

2

1

1

output:

1 1

input:⁵³

2

100

20

output:

100 20

input:⁵⁴

5

-1

۵۱ ورودی تماماً یک‌شکل
۵۲ صرفاً دوتا ورودی و یک‌شکل.
۵۳ دو ورودی و متفاوت
۵۴ ورودی درهم

```
2
3
-100
200
output:
200 -100
-----
```

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:
1_234
-----
```

```
input:
123
output:
123
-----
```

```
input:
12
output:
12
-----
```

سه کشور A, B, C در منطقه‌ی باختر غیرمیان‌ه واقع شده‌اند. شرایط اقتصادی در این منطقه بحرانیست به طوری که به طور روزانه، قیمت اجناس در کشورها افزایش می‌یابد. افزایش قیمت‌ها در این سه کشور به شرح زیر است:

کشور A روزانه n اشلون

کشور B روزانه m اشلون

کشور C روزانه k اشلون

در زمانی که شما این مسأله را می خوانید، قیمت یک توپ والیبال مطابق با جدول زیر می باشد:

کشور	قیمت
A	800
B	750
C	850

کارشناسان اقتصادی با این مسئله رو به رو هستند که آیا در d روز آینده، روزی وجود دارد که قیمت توپ والیبال در این سه کشور یکسان شود یا خیر. از شما خواسته شده به کمک این کارشناسان بروید و این مسئله را برایشان حل کنید.

پاسفنامه:

پاسخ ا:

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 ا.ا:

```

a = int(input())
b = int(input())
res = 1
while b > 0:
    res = res * a
    b = b - 1
print(res)

```

باز هم ضرب داریم. هر بار a درش ضرب میشه. اگر b بار اجرا شه، درواقع b بار a داره ضرب میشه.

پاسخ ۱.۲:

بزرگترین مقسوم‌علیه مشترک یعنی بزرگترین عددی که بر هر دو بخش‌پذیره. قاعدتاً بزرگترین عددی که بر هر دو بخش‌پذیره، باید کوچکتر و یا مساوی کوچکترین عدد بین اون دو عدد باشه. پس اول به صورت پیشفرض، کوچکترین عدد رو ب.م.م (یا به انگلیسی gcd) می‌گیرم:

```

num1 = int(input())
num2 = int(input())

if num1 > num2:
    gcd = num2
else:
    gcd = num1

```

بعدش یه `while` می‌زنم که هی چک کنه که آیا عدد `gcd` بخش‌پذیر یا نه؟ اگر بود، پرینتش می‌کنم. هر بار هم یکی از `gcd` کم می‌کنم که دونه دونه از بالا بیاد پایین بینه بالاخره بر کدوم بخش‌پذیر میشه.

```

num1 = int(input())
num2 = int(input())
if num1 > num2:
    gcd = num2
else:
    gcd = num1
while gcd >= 1:
    if num1 % gcd == 0 and num2 % gcd == 0:
        print(gcd)
    gcd = gcd - 1

```

شرط `while` رو هم این گذاشتم که تا وقتی که بزرگ‌تر مساوی ۱ هست ادامه بده. (چون کوچکترین مقسوم‌علیه یه عدد، عدد ۱ هست).

اما این کد یه مشکلی داره. این کد داره مقسوم‌علیه‌های این دو عدد رو به صورت نزولی چاپ می‌کنه. اما من صرفاً بزرگترین رو که چاپ کرد، تموم شه.

برای همین من نیاز به این دارم که شرط `while` ام رو جوری طراحی کنم که اولی که پیدا شد، پایان بپذیره. برای این کار میام خلاقیت میزنم و از یه متغیر کمکی استفاده می‌کنم. اسم متغیر رو می‌ذارم `is_end` و مقدار اولیشو صفر می‌ذارم. شرط `while` هم تغییر میدم و میگم که تا وقتی صفر هست، ادامه بده. هر وقت به اولین `gcd` که بر هر دو بخش‌پذیر بود رسیدی، مقدارش رو یک کن که عملاً `while` تموم شه:

```
num1 = int(input())
num2 = int(input())
if num1 > num2:
    gcd = num2
else:
    gcd = num1
is_end = 0
while gcd >= 1 and is_end == 0:
    if num1 % gcd == 0 and num2 % gcd == 0:
        print(gcd)
        is_end = 1
    gcd = gcd - 1
```

پاسخ ۲:

روش اول:

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

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

قبول داریم که قاعدتاً باید یه سری عدد به عنوان نمره بگیریم. جمعشون رو یه جا نگه داریم و در آخر تقسیم بر این تعداد کنیم که میانگین به دست بیاد؟

پس یه متغیر به عنوان نگه‌دارنده جمع نمرات تعریف می‌کنیم:

```
total_score = 0
```

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

خب مرحله بعد چیه؟ باید به تعداد `student_count` ها یه کاری رو انجام بدیم. این با چی ممکن بود؟ با `while`.

- خب چطور `while` رو بسازیم؟ یکم فکر کنیم!

+ می‌تونیم بگیریم مثلاً هر بار که طی میشه، توی `while`، یکی از تعداد دانشجوها کمتر شه. پس شرط `while` رو چطور می‌گذاریم؟

خب تا وقتی که تعداد دانشجو صفر بشه. (چون هر بار یکی یکی از تعداد کم می‌کردیم) 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 می‌نویسین، شرط رو هم بنویسین! بلکه یکم فکر کنین به روند و روند رو طی کنین ببینین شرط چی می‌تونه باشه! مثلاً اینجا من گفتم چطور طی کنم؟ به تعداد دانشجو. پس هر بار می‌تونم یکی از تعداد دانشجو کم کنم و وقتی به صفر رسید یعنی تموم. پس شرط میشه:

```
while student_count > 0:
```

خب توی `while` باید یه نمره از کاربر بگیرم. یعنی:

```
while student_count > 0:
```

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

خب بعدش باید اضافه‌اش 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 دارین 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 می‌تونین 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 می‌سازیم. یعنی مقدار `student_count` رو می‌ریزیم 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 می‌کنه!

ببینین! دقیقاً تفاوت این آموزش با آموزش‌های دیگه اینه که سعی می‌کنم قشنگ فکری که توی ذهن رخ میدره رو پیش ببرم و مثل یه فردی که داره تازه یاد می‌گیره فکر و حل مسأله رو پیش ببرم و همراه با شما کد رو باگ‌بایی (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 می‌خوریم. 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 دارین، 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 نباشه.

```

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» می‌خوردم.

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 اسماتون بامعنا باشه. یعنی score_count اسم بهتری از count هست. چون اطلاعات بیشتری میدی 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 داریم 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 گرفتیم 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 نره‌ها! اگر یادتون بره، score_count صفر می‌مونه و در آخر که بخوایم میانگین بگیریم، تعداد صفره و تقسیم بر صفر می‌خورین!

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

+ باشه باشه! ببینین پایتون اومده گفته که من می‌خوام نوشتار شما رو کمتر کنم. نخواین خیلی زیاد بنویسین. راحت تر شه کارتتون. برای همین اگر می‌خواین 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 ساده‌تره.
مثالای 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 اولویت‌بندی 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 می‌گیم خودش ضربدر ۲ + ۶ و خب اولویت ریاضی اینطور حکم می‌کنه که ضربدر ۶ شه و بعد بعلاوه ۲ شه. پس اگر خواستین جور دیگه باشه، می‌تونین 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 چک‌کردن 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 جوری 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 جوری 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 بذارش توی ماکزیمم، اگر نه، اگر کوچکتر بود، مینیمم جدید ما اینه و پس بذارش توی مینیمم. اگر نه 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 میگی 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 از 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 ۶:

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 چطور 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. همیشه. 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())
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 اینجاست 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 می گیرم 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 می کنه. من نباید هیچ ورودی بگیرم. ولی بدون چک کردن
 شرطی، بیرون while یه ورودی گرفتم! حالا سعی کنین کد رو بهبود ببخشین! 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 ننوشتیم `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 مشکلش 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! ^{۵۵} خواستم یکم اذیتتون کنم **ل(ه)** حالا کد درست کار می‌کنه و
 ما خوشحالیم :)
 - می‌زنمتا o_O

برنامه‌نویسی سراسر تمرین، بررسی و رفع مشکل و دیباگ کردنه. 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 برنامه‌هاتون 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)
 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:

۵۵ - درواقع داره‌ها (: ولی شما فعلاً بلدش نیستین!

```
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
```

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 یه شمارنده استفاده می‌کردیم تا بگیم `while` تا کجا پیش بره؟ مثلاً:

```
i = 0
n = 5
while i < n:
    i += 1
    print('Hi')
```

خب اینجا پایتون گفته من یه ساختار ساده‌تری رو برات در نظر گرفتم. یه چیز دیگه هم داریم که کارت رو ساده‌تر کنه. نخواهیم اول یه متغیر تعریف کنیم، بعد توی `while` اضافه یا کمش کنیم. یه 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.

یه سری چیزای دیگه رو امتحان کنیم. برای فهم بهتر، متغیر *i* هم همونجا چاپ می کنم و بین *for* ها یه سری پرینت خطفاصله می ذارم که توی چیزایی که چاپ میشن، بهتر متمایز شن:

```
# Code 1
for i in range(0, 3, 2):
    print(f'i: {i}')
Print('-----')

# Code 2
for i in range(10, 12, 1):
    print(f'i: {i}')
Print('-----')

# Code 3
for i in range(0, 3, 8):
    print(f'i: {i}')
Print('-----')
```

خروجی:

```
i: 0
i: 2
-----
i: 10
i: 11
-----
i: 0
```

کد ۱: از صفر شروع میشه. میره تا قبل ۳. دوتا دوتا زیاد میشه. بار اول با $i = 0$ وارد for میشه. بار بعد با $i = 2$ وارد for میشه. بعد $i = 4$ میشه و ایندفعه می‌بینه که شرط برقرار نیست. (یعنی دیگه ۴ کوچکتر از ۳ نیست). پس می‌پره بیرون از for. (صرفاً ۲ بار وارد for شد).

کد ۲: از ۱۰ شروع میشه. میره تا قبل ۱۲. یکی یکی زیاد میشه. بار اول با $i = 10$ وارد for میشه. بار بعد با $i = 11$ وارد for میشه. بعد $i = 12$ میشه و ایندفعه می‌بینه که شرط برقرار نیست. (یعنی دیگه ۱۲ کوچکتر از ۱۲ نیست). پس می‌پره بیرون از for.

کد ۳: از صفر شروع میشه. میره تا قبل ۳. هشت تا هشت تا زیاد میشه. بار اول با $i = 0$ وارد for میشه. بعد $i = 8$ میشه و ایندفعه می‌بینه که شرط برقرار نیست. (یعنی دیگه ۸ کوچکتر از ۳ نیست). پس می‌پره بیرون از for.

تمیزنویسی: کما به عدد قبلی می‌چسبه و از عدد بعدی یه 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):  
    print(i)
```

از ۳ شروع میشه و تا قبل ۹ پیش میره. دو تا دوتا پیش میره. یعنی آخرین باری که وارد for میشه، ۷ هست. ۷ ۵ ۳.
تعداد هم به دست میشه آورد:

$$3 = [(7 - 3) / 2] + 1$$

فرمول جمع اعداد هم براتون بنویسم شاید به کارتون اومد:

$$\text{summation} = [(last + first) * count] / 2$$

(آخری بعلاوه اولی) ضربدر تعداد. در آخر هم تقسیم بر ۲

خب شاید ما بخوایم از یه عدد بزرگتر هی بیایم پایین برسیم به یه عدد کوچکتر. مثلاً از ۳ تا قبل صفر پیش بریم و یکی یکی کم کنیم. اینطوری می نویسیمش:

```
for i in range(3, 0, -1):  
    print('hi')
```

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 i چه مقداری داره.
اجراش کنین که بهتر درکش کنین.
یه سری مثال دیگه رو خودتون تست کنین تا بهتر یاد بگیرین.

خب کمی ساده تر!

چون ما خیلی وقتا for هامون شبیه for i in range(0, 3, 1) هست، پایتون گفته بیا من کارو واست ساده می کنم. نیاز نیست بنویسی یکی یکی برو بالا. ننویسش. اگر ننویسیش، من به صورت پیش فرض فکر می کنم 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 تفکیک بشون کنین. خیلی وقتاً 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 شده!

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؟ اینطوری:

```
for i in range(100):
    print(i)
```

متغیر `for` هر چیزی می‌تونه باشه. لزوماً نیاز نیست `i` باشه. صرفاً کلمه `i` خیلی رایجه و رسمه که این اسمو می‌گذارن. همونطور که داخل `while` ما می‌تونستیم هر اسمی برای متغیر استفاده کنیم، اینجا هم میشه. 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 می‌کنین و فکر می‌کنین با قبلی فرق داره ولی فرق نداره و مقدار قبلی عملاً از بین میره و مقدار جدید for جاش قرار می‌گیره:

```
i = 70  
for i in range(4):  
    print(i)
```

```
print(f'i outside of for is: {i}')
```

عملاً اون 70 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 (حتماً ببینین):

<https://youtube.com/shorts/uHbrCFwU2iY>

8. Boolean

خب ما تا اینجا با ساختار if آشنا شدیم. مثلاً دو مورد زیر:


```
if 2 < 3:
    print('hi')

if 2 > 3:
    print('hi')
```

اگر حاصل جلوی if درست بود، اجرا میشد. مثلاً توی مثال بالا، اولین if اجرا میشه. چون ۲ کوچکتر از ۳ هست.

حالا بیایم کدهای زیر رو ببینیم:

```
print(2 == 2)
print(2 > 1)
print(2 < 1)
```

شاید در نگاه اول به نظر بیاد که کد اشتباهه. اما اینطور نیست! اگر چاپش کنیم، به ترتیب موارد زیر چاپ میشه:

```
True
True
False
```

درواقع این عملگرهای منطقی ریاضی، دارن یه چیز رو به ما بر می گردونن. درواقع دارن درستی (True) یا نادرستی (False) رو بر می گردونن. (دقت کنین که کرکتر اول **بزرگ** باید باشه!) چون اولی درسته، به ما میگه درست. دومی هم درسته میگه درست. سومی نادرسته میگه False. حالا 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 نمی دونم!

+ بذارین یه راهنمایی کنم. یادتونه می گفتیم سمت راست تساوی حساب میشه و ریخته میشه توی سمت چپ؟ پس برای اولی سمت راست رو حساب کنین. نگاه می کنه آیا ۵ بزرگ تر از ۳ هست؟ بله هست. پس حاصل میشه True. یعنی:

```
bool_var1 = True
```

برای بقیه هم چاپشون کنین و نتیجهشو ببینین:

```
print(bool_var1)
print(bool_var2)
print(bool_var3)
```

```
print(bool_var4)
```

خب این به چه کاری میاد؟

موقعی که ما اصطلاحاً می‌خوایم یه متغیر داشته باشیم که اگر توی روند برنامه یه تغییری رخ داد، بگیم بله تغییر رخ داد. یعنی فرض کنیم 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 پیش‌فرض یه متغیر به نام `is_even` تعریف می‌کنیم. (توی چیزای `boolean` همیشه سعی کنیم اسامی بامعنا باشن و نشون‌دهنده `boolean` بودنشون باشن. مثلاً اینجا مشخصه `is_even` یعنی «آیا زوج است؟» «بله» یا «خیر».)

بعدش میام چک می‌کنم که اگر بر ۲ بخش‌پذیر بود، یعنی اون پرچم یا `flag` ام که `is_even` بود رو تغییر میدم. می‌گم عه یافتم که زوج. پس فرضم اشتباه بوده. پرچم رو اصلاح می‌کنم به «بله» یا «درست».

بعدش اگر هر جای کد (مثلاً خط ۲۰۰ ام) خواستم بدونم زوج بود یا نه، دیگه نیاز به دوباره باقی‌مونده گرفتن نیست! صرفاً چک می‌کنم که اگر پرچم یا `flag` ام `True` بود، چاپ کنه `Even` (زوج) وگرنه، چاپ کنه `Odd` (فرد).

درواقع من می‌تونم از یه متغیر استفاده کنم که در طول برنامه، یه ویژگی از یه عدد یا چیزی رو ذخیره کنم. مثلاً من الآن ذخیره کردم که عدد فرد یا زوج. دیگه خط ۲۰۰ کد هم اگر خواستم که بدونم زوج یا فرد، نیاز نیست دوباره باقی‌مونده بگیرم. صرفاً از مقدار درون `is_even` استفاده می‌کنم که ببینم فرد یا زوج. این تکنیک خیلی به کار میاد. دیگه هر بار نیاز نیست محاسبات رو انجام بدم. یه بار که انجام دادم، کافیه. بارهای بعد از نتیجه استفاده می‌کنم.

یه نکته! پایتون بازم خواسته کار ما رو ساده‌تر کنه. یعنی گفته که دو تا `if` زیر عیناً یکی هستن:

```
# code 1
if is_even == True:
    print('even')

# code 2
if is_even:
    print('even')
```

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 می‌نویسیم:

```
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.
به طور کلی بهتر از دو مورد زیر به ترتیب برای چک «True» و «False» بودن استفاده شه. خواناترین:

```
if is_even:
    ...
if not is_even:
    ...
```

یکم پیچیده شد. چند بار بخونیدش و تحلیلش کنید و تا وقتی کامل مسلط نشدین، ادامه رو نخونین!

به چه دردی می خوره؟

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
```

ببینین 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

می گردونن. 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 کدهای زیر ارور میدن یا نه؟

```

num = 0

# code 1
if 5 > 2 or 5 / num:
    print('hello')

# code 2
if 5 / num or 5 > 2:
    print('hello')

# code 3
if 2 > 5 and 5 / num:
    print('hello')

# code 4
if 5 / num and 2 > 5:
    print('hello')

# code 5
if 5 > 2 and 5 / num:
    print('hello')

```

اول به مرور:

چه زمانی **or** اجرا می‌شود؟ حتی یکی از عبارت‌ها هم درست باشن اجرا می‌شه. همش غلط باشه اجرا نمیشه.

چه زمانی **and** اجرا می‌شود؟ زمانی که تمام عبارات درست باشن. یکی هم غلط بود اجرا نمیشه!

کد ۱:

برخلاف انتظار که فکر می‌کنیم چون تقسیم بر صفر داره، به ارور بخوره، نمی‌خوره! به این دلیل که پایتون از بالا به پایین و از چپ به راست پیش میره. می‌گه خب اول گفتی ۵ آیا بزرگ‌تر از ۲ هست؟ چون هست، می‌گه خب **or** یعنی یکیش هم درست بود، اوکیه. خب پس اصلاً چه نیازی که من دومی رو چک کنم؟ من برنامه پایتون هستم که یه سری دستور می‌خونم و اجرا می‌کنم. این اجراکردن زمان می‌بره. من هرچقدر بتونم سرعت رو زیاد کنم و از کارهای اضافه جلوگیری کنم، به نفع کاربر میشه. چون سرعت برنامه‌ش زیاد میشه. پس از چپ به راست که دارم می‌خونم، اولین چیزی که دیدم درسته، می‌گم خب عبارت جلوی **if** درسته. پس بقیه رو اصلاً نگاه نمی‌کنم که ببینم اصلاً اروری داره یا نه.

کد ۲:

به ارور بر می‌خوره. چون از چپ به راست که شروع می‌کنه، می‌بینم تقسیم بر صفر داریم و ارور «ZeroDivisionError: division by zero» رو میده.

کد ۳:

خب بازم از چپ به راست شروع می‌کنه. می‌گه عه ۲ که از ۵ بزرگ‌تر نیست! عبارت هم and داره. توی and حتی اگر یکیشم غلط بود، if اجرا نمیشه. پس در اولین غلطی که دید، به خاطر افزایش سرعت، دست نگه میداره و الکی بقیه رو چک نمی‌کنه. پس اصلاً نمی‌فهمه که عبارت‌های بعدی مشکل دارن یا نه. بدون ارور کد رو اجرا می‌کنه.

کد ۴:

به ارور بر می‌خوره. چون از چپ به راست که شروع می‌کنه، می‌بینه تقسیم بر صفر داریم و ارور «ZeroDivisionError: division by zero» رو میده.

کد ۵:

به ارور می‌خوره. دلیلش اینه که اولی رو که چک می‌کنه، می‌بینه که ۵ بزرگ‌تر از ۲ هست. پس باید دومی هم چک کنه (چون and صرفاً زمانی اجرا میشه که بینه آیا همش درست هست یا نه؟ اولی درست بود. باید بینه دومی هم آیا درسته یا نه؟). برای همین می‌بینه عه دومی تقسیم بر صفره و به ارور بر می‌خوره.

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! نکته! همونطور که شما بلد بودین یه چیزو از string به int و از int به float و... تبدیل کنین، اینجا هم ما می‌تونیم چیزا رو به bool تبدیل کنیم! بله! همه چیز در کامپیوتر میتونه تبدیل به boolean بشه! چون کامپیوتر تنها چیزی که می‌فهمه صفر و یک هست. صفر = False و یک = True.

```
print(bool(12))
print(bool(-2))
print(bool(0))
print(bool('hi'))
print(bool(' '))
print(bool(''))
```

همونطور که می‌بینین، تمام اعداد به جز صفر و تمام استرینگ‌ها جز استرینگ خالی، تبدیل به True میشن. حتی استرینگ‌گی که فقط یه دونه فاصله هست هم True هست! (مورد یکی مونده به آخر). درواقع همیشه جلوی while و if و چیزای شرطی، اول مقدار بولین چک میشه و چیزا اجرا میشن. پس درواقع دو while زیر یکسان هستن:

```
while n != 0:
while n == True:
```

اولی می‌گه تا وقتی به صفر نرسیده. دومی هم می‌گه تا وقتی True هست. تنها عددی که False هست، عدد صفر هست. پس هر دو while یه چیزن درواقع!

بریم سراغ یه سؤال خیلی طولانی!

- وای! سخت شد!

+ نه نه! اصلاً نترسین! حل سؤال آسونه. فقط می‌خوایم هی بهینه و بهینه و بهینه‌ترش کنیم! یعنی چندین روش بریم و هی بگیم اینجاش میشه بهتر کرد و هی برنامه رو سریع‌تر 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 بخش‌پذیر باشه.

راهنمایی ۲: یعنی درواقع اگر از ۲ تا قبل عدد پیش بریم، نباید بر هیچ عددی بخش‌پذیر باشه!

راهنمایی ۳: اگر حتی بر یک عدد بخش‌پذیر بود، باید بگین خب اول نیست.

راهنمایی ۴: این با چی بود؟ با flag. یعنی اگر حتی یه عدد هم برش بخش‌پذیر 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 فکرم می‌گه که باید از ۲ تا قبل عدد برم و چک کنم آیا برشون بخش‌پذیر 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 بخش‌پذیر 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 بخش‌پذیر باشه، یعنی اول نیست دیگه! مثلاً عدد ۱۲ رو در نظر بگیرین. هم بر ۲، هم ۳، هم ۴، هم ۶ بخش‌پذیره. ما فقط بدونیم بر ۲ بخش‌پذیره 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 `j` هست می‌پره 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 بخش‌پذیر نباشه، دیگه نیاز نیست بریم تا خود عدد. حداکثر نیازه تا نصفش برم. چون اگر بر ۲ بخش‌پذیر بود، حداکثر بزرگترین مقسوم‌علیهش دیگه تا نصفش بود: یعنی بزرگترین مقسوم‌علیه 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 وگرنه و نمیشد توی for به کارش برد. چون for عدد با عدد صحیح کار می‌کنه نه اعشاری!

روش چهارم:

اما بازم این بهینه نیست. ما تقریباً $n / 2$ بار باید طی کنیم. بهتره بریم تا sqrt. حداکثر مقسوم‌علیه اول یه عدد، جذرش هست دیگه.

- چرا؟

+ توضیح غیر دقیق: بر هر عدد دیگه‌ای بخش‌پذیر باشه، مثلاً بزرگ‌تر از جذر، حتماً یه مقسوم‌علیه اول کوچکتر از جذر داره. مثلاً ۵۵، درسته بر ۱۱ هم بخش‌پذیره، اما یه مقسوم‌علیه 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 مقسوم‌علیهش جذرشه. اگر نبود، حتماً یه مقسوم‌علیه کوچکتر از جذر داره!

+ توضیح دقیق: فرض کنیم عدد اول نیست. پس قاعدتاً باید از ضرب حداقل دو عدد غیر یک به دست بیاد. (چون عدد اول یعنی صرفاً بر خود و یک بخش‌پذیر باشه. پس عددی که اول نباشه، حداقل بر دو عدد غیر یک بخش‌پذیره. مثلاً ۶ که از ضرب ۲ و ۳ به دست میاد.) پس درواقع:

$$n = a * b$$

خب حالا من میگم باید حداقل یکی از a و یا b کوچکتر از \sqrt{n} باشن.

- چرا؟

+ فرض کنیم نباشن. پس یعنی هر دو بزرگ‌تر از \sqrt{n} هستن. پس:

$$a > \sqrt{n}, b > \sqrt{n} \Rightarrow ab > \sqrt{n} \cdot \sqrt{n} \Rightarrow ab > n$$

و عملاً این ممکن نیست. چون فرض اولیمون این بود که $n = ab$. پس باید حداقل یکی از اعداد، کوچکتر یا مساوی \sqrt{n} باشه. اون عدد هم یا اوله یا خودش از عدد اول تشکیل شده. پس قاعدتاً صرفاً نیاز به جذر پیش بریم.

قبلش باید با نحوه جذرگرفتن در پایتون آشنا بشیم:

بینین فرض کنیم 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 بزنی! 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.

خلاصه نوشتن **math** و زدن دکمه **ctrl + space** می‌تونین بقیه دستورات (توابع) رو ببینین و سعی کنین باهاش بازی کنین. 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.^{۵۶} همین مفاهیم ساده، پیش‌زمینه مهم‌ترین چیزای روزمره زندگیمون!

روش پنجم؛

خب بیایم یکم باز بهترش کنیم! به نظرتون چطور میتونیم بهترش کنیم؟ + 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 بخش‌پذیر نباشن، عملاً چک کردن ۴، ۶، ۸ و... بی‌فایده‌س! چون بر این‌ها هم بخش‌پذیر نیستن. یعنی عملاً داریم یکی درمیان اعداد رو الکی چک می‌کنیم! پس بیایم بهینه‌ترش کنیم:

^{۵۶} درواقع رمزنگاری `RSA` بر این استواره. می‌تونین آموزشش رو از قسمت «`RSA`» بخونین!

```

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، میگیریم دوتا دوتا میریم. یعنی ۳، ۵، ۷، ۹ و... یعنی صرفاً اعداد فرد رو چک می‌کنیم.

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 ۱ توی if اولی اشتباهی می‌گیریم که چون زوج اول نیست! ولی خب استثناء هست و حواسمون به استثناءها باشه!
 پس می‌تونیم قسمت if که چک می‌کنه که اگر زوج is_prime رو زوج می‌کنه، بگیریم که اگر زوج و ۲ نیست. یعنی:

```

if divisor % 2 == 0 and divisor != 2:

```

سعی کنید تغییرات سرعت برنامه رو با بنچمارک چک کنید.^{۵۷} چون گاهی فکر می‌کنین که تغییراتی که دادین کد سریع‌تر شده ولی فکرتون اشتباهه. باید تست کنید تا مطمئن بشین.

^{۵۷} یه وبسایت خوب برای نحوه محاسبه زمان:

How to Measure Execution Time of a Program: <https://serhack.me/articles/measure-execution-time-program/>

اگر می‌خواهید پیشرفته‌تر بدوینید، موارد زیر رو بخونید. وگرنه نیاز نیست!

یه راه ساده برای تست مدت زمان اجرای یه برنامه، استفاده از ابزارهای تحت ترمیناله. مثلاً توی لینوکس ابزار «hyperfine» می‌تونه کمک کنه.

سعی کنید ورودی خیلی بزرگ باشه که تأثیر رو بهتر متوجه شین. مثلاً روی ورودی خیلی کوچیک، ممکنه تفاوت سرعتی مشخص نباشه و حتی یکی که کندتره، اشتباهی سریع‌تر نشون بده. پس برای همین، سعی کنید ورودی رو تا حد امکان بزرگ بدین.

```
hyperfine 'python3 file.py'
```

که `file.py` اسم فایلیه هست که می‌خواهید اجرا شه. (حواستون باشه که کدتون نباید ورودی بگیره. باید مقادیر رو توی خود کد ست کنید و ورودی نگیرین! 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 می‌کنن که پردازنده در یه دمای خاص، بدون ران شدن چیز اضافه و با شرایط یکسان تست می‌کنن که شرایط خارجی مثل اجرا شدن برنامه‌ای دیگه روی سرعت اجرای این برنامه تأثیر نداره.

معمولاً سایتا بهتر می‌تونن عمل کنن چون مکانیزمای بهتری دارن نسبت به شما.

درواقع فرض کنید 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 دارید 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.

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

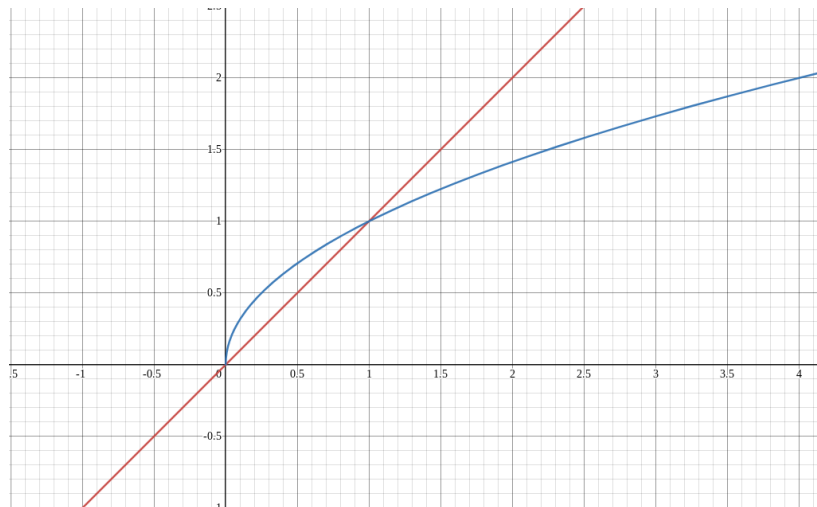
برنامه‌نویسا برای اینکه بتونن کدشونو از لحاظ سرعتی مقایسه کنن، از یه مفهوم ریاضی‌طوری به نام «time complexity» یا «پیچیدگی زمانی» استفاده می‌کنن.

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: کدام بهتره؟ رشد تابع به صورت n باشه بهتره یا \sqrt{n} ؟ (زمان n برابر شه یا زمان \sqrt{n} برابر

شه؟)



تابعی که شکش 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.^{۵۸}

درواقع time complexity مفهوم خیلی مهمیه. درواقع شما اگر الگوریتمی بنویسین که n^3 باشه و من الگوریتمی بنویسم که n باشه، با بزرگ شدن 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:

	Time(n)	Time(n^3)
Input = 1	1	1

^{۵۸} 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.

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! پرقدرت! 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:

$$x_i = \left(x_0^{(2^i \bmod \text{lcm}(p-1, q-1))} \right) \bmod M$$

خب هرکدوم از اینا چی هستن؟

مورد i : 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
...
 x_n

مثال:

input:⁵⁹

11
23
3
6

output:

9
81
236
36
31
202

input:

7

⁵⁹ همونطور که می‌بینین، $p = 11$ و $q = 23$ و $x_0 = 3$ و $n = 6$ است.

11

5

4

output:

25

9

4

16

اشتباهات رایج:

۱- ممکنه خود محاسبه lcm اشتباه رفته باشین.
ک.م.م باید از بزرگترین عدد از بین دو عدد شروع کنیم و یکی یکی بریم بالا. اولین که بر هر دو عدد
ما بخش پذیر بود، همون ک.م.م هست.

$\text{lcm}(10, 15)$

10, 11, 12, ... **30**

۲- ممکنه lcm رو برای p و q حساب کرده باشین. ولی سوال برای p-1 و q-1 می‌خواد. یعنی

incorrect: $\text{lcm}(p, q)$

correct: $\text{lcm}(p-1, q-1)$

پایان نامه:

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 چندتا if و while داشت! پس چیکار کنیم؟ سخت شد نه؟ کذا 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

فرض کنین شما می‌خواین لباس بشورین. خب یه راه اینه که خودتون دستی برین دونه‌دونه لباسا رو بشورین. اما یه راه دیگش اینه که شما یه ماشین بسازین که صرفاً بهش لباسا رو بدین، خودش خودکار بشینه لباسا رو بشوره. قاعدتاً خیلی بهتر میشه دیگه! درواقع شما اصلاً نیاز نیست بدونین که داره چیکار انجام میده. شما صرفاً لباس کثیف میدی بهش، اون لباس پاک بهت تحویل میده. اصلاً نیاز نیست درگیر نحوه شستن بشین!

یادتونه ما از دستور `max`^{۶۰} مثل زیر استفاده کردیم؟

```
print(max(2, 3, -1, 40))
```

درواقع اصلاً نیاز نبود من درگیر این بشم که `max` داره چیکار انجام میده. صرفاً ازش استفاده می‌کردم و برام تنها این مهم بود که یه چیزی میدم و کاری که می‌خوام رو انجام میده. به این مفهوم می‌گن «تابع» یا «function».

درواقع شما یه تیکه کد تو میبری یه جا. اسمشو یه چیزی می‌گذاری. بعداً هر وقت بهش نیاز داشتی، فقط نیازه صداش بزنی. درواقع کدت رو می‌بری یه جا و برای کدت یه تعریف (definition) ارائه میدی. مثلاً میگی تعریف می‌کنم که این یه تابع هست اسمشم می‌ذارم فلان.

یادتونه ماشین لباسشویی رو مثال زدم؟ لباس کثیف می‌گرفت، یه سری عملیات روی لباسا انجام می‌داد و لباس تمیز پس می‌داد. اینجا هم مثل همونه. تابع یه چیزی می‌گیره، یه سری کار می‌کنه و یه چیزی پس میده. درواقع `max`، تابعی آماده بود که افرادی که زبون پایتون رو نوشته بودن، این تابع رو نوشته بودن و یه جا توی فایلای براتون قرار داده بودن.

مثال ۱:

```
def _sum(a, b):  
    summ = a + b  
    return summ  
  
print(_sum(2, 3))
```

میام یه گوشه از کدهام (فرقی نداره کجا! هرکجا باشه اوکیه! فقط همونطور که گفتیم که پایتون از بالا به پایین کد رو می‌خونه، باید اول تابع باشه و بعد از اون ازش استفاده بشه! نمیشه اول استفاده کنین بعد بگذارینش!)، `define` می‌کنم (`def`) که یه تابع دارم می‌سازم به نام `_sum`^{۶۱}. توی پرانتز می‌گم که چه پارامترهایی قراره بگیره. متغیر `a` و `b` رو می‌گیره. بعد جمع می‌زنه می‌ریزه توی یه متغیر دیگه. بعد حاصل رو به ما برمی‌گردونه.

^{۶۰} دستور `max` رو قبلاً مثال بزنی.

^{۶۱} چرا اولش یه `underline` گذاشتم؟ چون `sum` خودش جزء `reserved word` هاست. (قبلاً درباره‌ش صحبت کردیم که یه سری اسما برای پایتون رزرو شده و شما نباید دستش بزنین. (معمولاً رسمه برای تابع، اگر اسم تکراری بود، اولش `underline` می‌ذارن.)

پس من اگر چاپ کنم حاصل جمع ۲ و ۳، میره ۲ و ۳ رو جمع می‌زنه و بر می‌گردونه. یعنی ۵ رو بر می‌گردونه. عملاً توی پرینت ۵ قرار می‌گیره و ۵ چاپ میشه.

توجه: هر وقت `return` انجام شد، دیگه مقدار رو پس داده و اجرای تابع تموم میشه و مقدار رو بر می‌گردونه.

البته می‌تونستیم اون متغیر اضافی رو تعریف نکنیم و صرفاً بنویسیم:

```
def _sum(a, b):  
    return a + b  
  
print(_sum(2, 3))
```

یعنی برام برگردون حاصل جمع `a` و `b` رو.

مثال ۲ (تابعی که توان ۲ یک عدد رو پس بده):

```
def square(a):  
    return a**2  
  
print(square(5))
```

مثال ۳ (تابعی که مکسیموم سه عدد رو پس بده):

```
def _max(num1, num2, num3):  
    if num1 >= num2 and num1 >= num3:  
        return num1  
    elif num2 >= num1 and num2 >= num3:  
        return num2  
    else:  
        return num3  
  
print(_max(3, 4, 5))
```

یه تابع تعریف کردم که مکسیموم سه عدد رو حساب کنه. سه عدد بهش میدیم و چیزی که به ما پس می‌ده، مکسیموم هست. می‌گم:

اگر عدد ۱ بزرگ‌تر یا مساوی ۲ و ۳ بود، عدد ۱ رو برگردون.

اگر نه، چک کن بین اگر عدد ۲ بزرگ‌تر یا مساوی ۱ و ۳ بود، عدد ۲ رو برگردون.

وگرنه، قاعدتاً عدد ۳ بزرگ‌ترینیه. پس اونو برگردون.

و در نهایت بیرون تابع می‌گم که چاپ کن حاصلی که تابع `_max` برمی‌گردونه رو.

دیدین؟ خیلی کد تمیزتر میشه. دیگه هرجای اگر نیاز داشتیم حاصل مکسیموم سه عدد رو به دست بیارم، دیگه نیاز نیست هی if و else و... بنویسم! صرفاً نیازه تابع رو روی اون سه عدد صداش بزnm. اینطوری دیگه کد خیلی تمیز در میاد!

توجه! تابع پس از انجام return تموم میشه و دیگه ادامه خودشو طی نمی‌کنه. مثلاً:

```
def f(a):  
    return 2  
    return 3
```

خب اولین return ای رو که دید، برمی‌گردونه. درواقع همیشه ۲ برمی‌گرده و اصلاً سراغ خط بعدی نمیره. چون return صورت گرفته و تموم شده.

مثال ۳ (تابعی که بفهمه یه عدد زوجه یا نه):

```
def is_even(num):  
    if num % 2 == 0:  
        return True  
    else:  
        return False  
  
print(is_even(5))
```

اگر زوج باشه، همونجا True رو پس میده و عملاً پایان تابع فرا می‌رسه و اصلاً سراغ else نمی‌ره. اما اگر فرد باشه، میره سراغ else و بعدش مقدار False رو ریترن می‌کنه. حالا هر جای کد نیاز داشتیم که ببینیم یه عدد زوجه یا نه، صرفاً تابع رو روی اون عدد صدا می‌زنم. مثلاً فرض کنیم می‌خوام اعداد زوج ۱ تا ۱۰۰ رو پرینت کنم. به نظرتون چیکار کنم؟

```
def is_even(num):
    if num % 2 == 0:
        return True
    else:
        return False

for i in range(1, 101):
    if is_even(i) == True:
        print(i)
```

میگم از ۱ تا قبل ۱۰۱ حرکت کن (چون می‌خواستم ۱۰۰ هم جزء اعداد باشه). بعد می‌گم هر بار که داری روی اعداد حرکت می‌کنی، چک کن ببین آیا حاصل `is_even` روش `True` هست یا نه؟ اگر بود یعنی زوجه و پس پرینتتش کن.

یعنی دیگه نیاز بود کد رو پیچیده کنم و توی `for` باقی‌مونده بگیرم. الان کدم خیلی تمیزتر شد. سوال! توی تابع اسم رو `num` تعریف کردم ولی بیرون تابع دارم بهش `i` میدم. مشکلی نداره؟ + خیر! محیط تابع از محیط عادی جداست. یعنی درسته من دارم `i` میدم، ولی اسمش توی تابع میشه `num`. درواقع متغیر توی تابع با متغیر بیرونش تفاوت داره. یعنی هرکاری من روی متغیر درون تابع انجام بدم، مستقل از بیرونیه هست و بیرونیه هیچگونه تغییری نمی‌کنه!

یعنی:

```
def increment(a):
    a = a + 1
    return a

a = 5
print(increment(a))
print(a)
```

همونطور که می‌بینی، `a` بیرونی، تغییری نکرد! درواقع همونطور که گفتیم، اون متغیر درونی، مستقل از بیرونیه.

حتی می‌تونم جمع اعداد زوج یک تا ۱۰۰ رو چاپ کنم:

```
def is_even(num):
    if num % 2 == 0:
        return True
    else:
        return False

summ = 0
for i in range(1, 101):
    if is_even(i) == True:
        summ += i
print(summ)
```

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. هر دو یه کار انجام میدن.

حتی میشد اینطوری هم نوشتش:

```
def is_even(num):
    return num % 2 == 0
```

یعنی برگردون حاصل اینکه عدد بر ۲ شده ۰ یا نه. (یادتونه گفتیم علامتای مقایسه‌ای، `True` یا `False` بر می‌گردونن؟) اگر باقی‌موندش شده صفر، خب `True` بر می‌گردونه. اگر صفر نشده مثلاً `۱ == ۰` میشه و خب حاصل این `False` هست و `False` بر می‌گردونه.

مثال!

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!

«برنامه‌ای بنویسین که مقسوم‌علیه‌های اول یه عدد رو چاپ کنه.»
اول باید از ۲ تا خود عدد پیش بریم.

– چرا تا خود عدد؟

+ چون یکی از مقسوم‌علیه‌های یک عدد، می‌تونه خود عدد باشه و خود عدد هم ممکنه اول باشه.
و بعدش مقسوم‌علیه‌هاشو پیدا کنیم (یعنی تمام اعدادی که از ۲ به بعد بر اون عدد بخش پذیرن) و پاسشون بدیم به تابع که اگر اون مقسوم‌علیه 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. کدمون خیلی قشنگ میشه!

از همین الان سعی کنین چیزایی که میشه تابعش کرد رو تابع کنین. نخواستن همیشه هی کد is_prime رو ده جای کد بنویسین. صرفاً تابعش می‌کنین که بیاد انجامش بده. این خیلی به تمیز شدن 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^{۶۲} و قابل دیباگ‌تر می‌کنه.
- یه بار تستش کن، همیشه استفاده کن!
- کدت سریع‌تر میشه!^{۶۳}
- اگر یه وقت خواستی تغییری به وجود بیاری توی کد تابع، راحت مشخصه کجا بود که بری تغییر به وجود بیاری و نگران تداخل با خارج نیستی!

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`

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.^{۶۴} برنامه اینه که سه عدد بهتون میدن و شما باید بگین می‌تونه مثلث باشه یا نه؟

۴- 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: 64

output: 32

input: 62

output: 32

^{۶۲} 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)

^{۶۳} 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/>

^{۶۴} البته قسمت تفاضل این نامساوی رو فعلاً نادیده می‌گیریم

تضمین میشه اعداد ورودی شامل ۲ و اعداد بالاتر هستن.

پاسفنامه:

پاسخ ۱:

```
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")
```

در واقع ابتدای کد دو تابع رو تعریف کردم. بعدش ازشون پایین استفاده کردم. خیلی تمیزتر نشد؟ در واقع وقتی شما یه پروژه می‌زنین، همه چیزاتون تابع‌طوری (یا اصطلاحاً functional) میشه و به شدت قابل فهم میشه.

پاسخ ۲:

```
def is_login_valid(username, password):
    if username == "admin" and password == "1234":
        return True
    return False

username = input("Enter username: ")
password = input("Enter Passwrod: ")

while is_login_valid(username, password) == False:
    print("Invalid username or password")
    username = input("Enter username: ")
    password = input("Enter Passwrod: ")

print("Welcome")
```

خب یه username و یه 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 یه 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 که 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!

تذکره! همچنین پسوردهایی خیلی ساد و هیچ وقت نباید از شون 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 سمبل ها.
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 می تونه 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 می کنم که می تونه تشکیل بده. یعنی یه متغیر در نظر می گیرم 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 می کنم.
در نهایت مقدار متغیر رو return می کنم.

```
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؟

پاسخ ۴:

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 می نویسیم:

```
def pow2_below(num):  
    result = 1  
    while result < num:  
        result *= 2 # result = result * 2  
  
    return result // 2  
  
inp_num = int(input("Enter a number: "))  
print(pow2_below(inp_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 می خواهیم نیست! ما می خواستیم `result < num` بشه. ولی شرط بیرون اومدن اینه که `result` بزرگ تره. پس درواقع یه ضربدر ۲ اضافه هست. پس تقسیم بر ۲ می کنیم که اون از بین بره. (اینو اگر اشتباه رفتین، ایرادی نداره. با تست و بررسی نهایی تابع می فهمین مشکلو رفع می کنین.)

- 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 می ده.

- 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 می ده.

خب به نظرتون اگر راه حل رو اینطور می نوشتم، مشکل کد چی بود؟

```
def pow2_below(num):
    result = 2
    while result < num:
        result *= result # result = result * result

    return result // 2

inp_num = int(input("Enter a number: "))
print(pow2_below(inp_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 اینجاست 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!

نکات:

صفرم) تابع لزوماً نیاز نیست `boolean` برگردونه! می‌تونه هر چیزی برگردونه. چه عدد اعشاری چه متن. هرچی! الف) تابع ممکنه چیزی نگیره! مثلاً:

```
def fun1():
    a = 2
    return a + 2

print(fun1())
```

وقتی پرینتش کنیم، ۴ بر می گردونه. هیچی نگرفت ولی یه چیزی پس داد. (یادمون هست که پرینت همیشه مقدار ریترن شده تابع رو برمی گردونه).

ب) تابع ممکنه چیزی پس نده!

```
def fun2():  
    a = 2  
  
print(fun2())
```

خب قرار بود تابع وقتی پرینت انجام می دیم، مقدار ریترن شده برگرده. حالا که چیزی ریترن نشده، عملاً «None» به معنای «هیچی» چاپ میشه. درواقع وقتی تابعی چیز خاصی بر نمی گردونه، یه چیز خاص به نام هیچی برمی گردونه و خب پرینت هم میاد همینو چاپ می کنه.
مثال دیگر:

```
def fun3():  
    print('hi')  
  
print(fun3())
```

بله! چیز عجیبی نیست! تابع یه سری کد هست. می تونه داخل خودش پرینت داشته باشه. پس وقتی تابع صدا زده شد، میره پرینت «hi» رو انجام میده. بعدش هم «None» رو برمی گردونه و پرینت خارج تابع، مقدار ریترن شده یعنی «None» یا همون «هیچی» رو چاپ می کنه.
- من نمی خوام اون «None» هم چاپ شه الکی. راهی هست فقط تابع رو استفاده کنم و این هیچی چاپ نشه؟
+ بله میشه! صرفاً تابع رو صدا بزنین. توی پرینت نذارینش که پرینت بیاد مقدار ریترن رو چاپ کنه:

```
def fun3():  
    print('hi')  
  
fun3()
```

ج) حتی `return` یه تابع می تونه چندتا چیز برگردونه ولی موقعی که صداش می زنیم، 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!

ا 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
```

از اینجا به بعد (قبل مبحث بعدی)، کمی پیشرفته‌تره. پیشنهاد می‌کنم فعلاً نخوندیش چون گیج میشی. بزارین بعداً بخونیش.

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 اون هم مقدار پیشفرض بدم.

- آیا همیشه ساخت تابع باعث سریع تر شدن کد میشه؟
+ خیر! مثلاً:
به نظرتون کدوم کد سریع تره:

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 سریع تره. چون که خود صدا زدن تابع هزینه بر داره. پس برای هر چیزی تابع ننویسین! یا اگر for ای که خیلی بار اجرا میشه (مثل مورد بالا) رو روی یه تابع پیاده سازی می خوانین کنین، برای سریع تر شدنش، for رو ببرین توی تابع که تعداد function call هامون کمتر شه.

10) String

تا اینجا خیلی با اعداد بازی می کردیم. اما مهمتر از اعداد، متن و رشته ها (string) هستن. خب یه متغیر string رو چطور تعریف می کردیم؟ 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 ترتیبه:

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. مطمئناً اگر به چیزی خارج از اون بخواین دسترسی داشته باشین، ارور می‌ده:

```
s = 'abcdefghijk'
print(s[100])
```

ارور می‌ده:

```
IndexError: string index out of range
```

منطقیه هست! خب اصلاً ایندکس ۱۰۰ نداره که!

حالا فرض کنین که می‌خوایم کرکتر اول (ایندکس صفر) تا قبل از کرکتر چهارم (ایندکس ۳) رو پرینت کنیم. شبیه for می‌گیم که:

```
string2 = "abcdefgh"
print(string2[0:3:1])
```

یعنی از صفر برو تا قبل ایندکس ۳. یکی‌یکی برو بالا.

می‌تونیم دوتا دوتا هم بریم بالا. یعنی بگیم مثلاً ۲ تا ۲ تا ایندکس برو جلو:

```
string2 = "abcdefgh"
print(string2[2:5:2])
```

قبل اجرا کردنش، یکم روش فکر کنین چه کرکترایی چاپ میشه؟

+ خب می‌گیم از ایندکس ۰ شروع کن برو تا قبل ۵. دو تا دوتا برو جلو. اون کرکترایی که توی رنج جا می‌گیرن رو چاپ کن. که میشه:

```
ce
```

به این کار میگن slicing.

اینجا هم مثل for بود که می‌تونستیم پارامتر سوم رو ندیم. اگر ندیم، به صورت پیش‌فرض، ۱ در نظر می‌گیره:

```
string2 = "abcdefgh"
print(string2[0:3])
```

که خروجی زیر رو میده:

```
abc
```

مثل for اگر اولی رو نگذاریم، یعنی از ۰ شروع کن:

```
string2 = "abcdefgh"
print(string2[:3])
```

از ایندکس ۰ تا قبل ۳.

تازه می‌تونیم آخری هم ندیم. اگر ندیم یعنی برو تا تهش.

```
string2 = "abcdefgh"
print(string2[2:])
```

از ایندکس ۲ شروع کن برو تا تهش!

حتی می‌تونیم مثل for از بالا بیایم به پایین. یعنی.

```
string2 = "abcdefghi"
print(string2[7:4:-1])
```

از ایندکس ۷ شروع کن. بیا تا قبل ۴. چه جوری بیا؟ یکی یکی کم شو.

یه چیز جالب! پایتون (برخلاف خیلی از زبان‌های 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 می‌تونین کار کنین. اما یکم سخت‌تره. بیشتر زمانی به کار میاد که می‌خوانین بگین مثلاً حرف آخر یا یکی مونده به آخر رو چاپ کنین. راحت می‌گین:

```
string2 = "abcdefghi"
print(string2[-1])
```

فرض کنین بخوام یه رشته به طول n رو برعکس کنم:

```
s[n-1::-1]
```

می‌گم از ایندکس آخر (که یکی از طول کمتره) شروع کن و برو تا پایان. (پایان اینجا معنای پایان بازه می‌ده. یعنی از آخر بیای تا اول اول). یکی‌یکی کم شو.

یکم با این ایندکس‌ها 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 برگردین و... خلاصه یکم باهش بازی کنین.

جمع‌بندی:

۰- رنج ایندکس‌ها از «۰» تا « $n-1$ » هست که n نشون‌دهنده طول استرینگ هست.
۱- اگر بخوایم به کرکتر آخر دسترسی داشته باشیم، با ایندکس « -1 » یا « $n-1$ » که n نشون‌دهنده طول هست، می‌تونیم دسترسی داشته باشیم.

۲- به نظرتون هرکدوم از موارد زیر چیا چاپ می‌کنن؟

```
s = '0123456'
n = len(s)
print('s[0]=', s[0])
print('s[5]=', s[5])
print('s[-1]=', s[-1]) # s[n-1]
print('s[0:4] =', s[0:4])
print('s[0:5:2] =', s[0:5:2])
print('s[n-1::-1]=', s[n-1::-1])
```

پاسخ:

```
s[0]= 0
s[5]= 5
s[-1]= 6
s[0:4] = 0123
s[0:5:2] = 024
s[n-1::-1]= 6543210
```

توضیح مورد آخر: از ایندکس آخر (= طول منهای یک)، شروع کن یکی یکی بیا پایین، و برو تا آخر بازه. (وقتی چیزی نداریم برای دومی، یعنی برو تا تهش که اینجا یعنی از آخر بیا تا اول.)

11) for + string

برای کار روی استرینگ، خیلی وقتا ما نیاز داریم که طول استرینگ رو بدونیم. پایتون خودش یه تابع داره به نام len. که طول استرینگ رو میده:

```
s = 'abcde'
length = len(s)
print(length)
```

که چاپ می کنه ۵. یعنی از ۵ کرکتر تشکیل شده.

رو مورد بسیار مهم که انتظار داریم اونقدر خوب بفهمینش که انگار حفظش کردین!

(الف)

فرض کنین می خوام از ابتدای استرینگ تا آخر استرینگ برم. به نظرتون for رو چطور بنویسم؟ قبول دارین که من باید از ایندکس ۰ تا $n - 1$ پیش برم که روی تمام ایندکس ها حرکت کنم و بتونم به کرکترها دسترسی داشته باشم؟ خب همین کار رو انجام بدیم:

```
s = '01234'
for i in range(len(s)):
    print(s[i])
```

یادتون هم هست که وقتی یه عدد به for میدیم، از ۰ تا یکی کمتر از اون عدد میره؟ اینجا هم پس از ۰ تا $n - 1$ ای که خواستیم پیش میره.

(ب)

فرض کنین می خوام از آخر استرینگ تا ابتدای استرینگ پیش برم. به نظرتون for رو چطور بنویسم؟

قبول داریم که من باید از ایندکس آخر یعنی $n - 1$ تا ایندکس اول یعنی 0 پیش برم که روی تمام ایندکس‌ها از آخر به اول حرکت کنم و بتوانم به کرکترها دسترسی داشته باشم؟ خب همین کار رو انجام بدیم:

```
s = '01234'
for i in range(len(s) - 1, -1, -1):
    print(s[i])
```

میگم که از ایندکس آخر یعنی $n - 1$ شروع کن و تا قبل -1 (خود صفر) پیش برو. یکی یکی بیا پایین. اینطوری دارم از آخر به اول میام. این دو خیلی مهمن. این دو رو حفظ باشین عملاً که در تمام سوالا به اینا نیاز داریم.

خب حالا سعی کنین یه تابع بنویسین که تعداد «a» های درون 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 و از اول تا آخر استرینگ حرکت می‌کنم. دونه دونه ایندکسا 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 کنین توابع رو به صورتی بنویسین 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(a_count(char, string))
```

درواقع اومدم به یه حالت کلی‌تر نوشتیم که بشه بعداً هم استفاده‌ش کرد و صرفاً برای یه کار خاص نباشه. برای هر کرکتری بشه استفاده‌ش کرد.

هرچی تابعتون کلی‌تر باشه که بشه جاهای بیشتری استفاده کرد، بهتره! فرض کنین شما بعداً می‌خوانین 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 پاک‌کردن یه کرکتر خاص رو برای ویرگول صدا می‌زنین ← `remove(',', s)`
- سوم تابع پاک‌کردن یه کرکتر خاص رو برای نقطه صدا می‌زنین ← `remove('.', s)`

دیدین چقدر مرحله به مرحله فکرکردن بهتره؟ درواقع اگر صرفاً کدتون که خارج از تابع بود رو برید توی تابع بنویسین که هنر نکردین! هنر تقسیم کارها برای تمیزترشدن کد و بررسی راحت‌تر 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 راحت‌تر می‌تونم مشکلات کدمو پیدا کنم. کدم تمیزتر و خواناتر.

اما خب یه راه دیگه هم میشه `for` رو برای `string` ها پیاده‌سازی کرد. اینطوری:


```
def char_count(char, s):
    count = 0
    for c in s:
        if c == char:
            count += 1
    return count
```

اینطوری ترجمه میشه:

برای تک تک حروف درون string که اسمشو c گذاشتم، ببین اگر برابر char بود، یکی به تعداد اضافه کن.

این اسم میتونه هرچی باشه. می‌تونین بذارین:

```
for hello in s
```

ولی خب همیشه گفتیم اسما با معنی باشه. من c به معنای کرکترهای درون string گذاشتم. یعنی دونه دونه کرکترها رو طی می‌کنه. یعنی اولین بار char، کرکتر ایندکس ۰ هست. بعدش کرکتر ایندکس ۱. بعدش کرکتر ایندکس ۲ و الی آخر. اینطوری با ایندکس کار نمی‌کنیم و داریم روی خود حروف حرکت می‌کنیم که خواناتر و سریع‌تر هم هست! حتی اینم داریم:

```
s1 = 'hi'
s2 = 'oavhivs'
if s1 in s2:
    print('yes')
else:
    print('no')
```

میگه اگر s1 درون s2 بود، پرینت کن «yes».

درواقع ساختارهای for ... in و if ... in خیلی کاربردین.

لطفاً ساختار for ... in و if ... in رو هم کامل کامل بلد باشین که خیلی نیازش داریم.

پس چی شد؟ دوتا از قبل بلد بودین، دوتا هم اینجا میشه ۴.

- به for از ابتدا به انتها می‌استرینگ با کمک ایندکس.

- به for از انتها به ابتدا می‌استرینگ با کمک ایندکس.

- به for ... in.

- به if ... in.

مثال:

تابعی بنویسین که بفهمه آیا یه string، آینه‌ای (palindrome) هست یا نه؟
توضیح: به string ای می‌گیم که آینه‌ای که چه از چپ به راست و چه از راست به چپ بخونیمش، یک چیز هست. مثال:

```
aba  
aaabaaa  
hih  
abcba  
aaaa
```

پاسخ:

نیازه از اول تا نصب برم و دونه‌دونه با تهی‌ها مقایسه کنم. اگر یه دونه هم با اونوریش یکی نبود، پس palindrome نیست!

پنج نوع مختلف می‌نویسم. همشو بخونین. ۵ امی رو فعلاً بلد نیستین. بعد پایان متدهای string برگردین بخونین:

```
def is_palindrome1(s):  
    n = len(s)  
    for i in range(n // 2):  
        if s[i] != s[n - 1 - i]:  
            return False  
    return True  
  
def is_palindrome2(s):  
    is_pal = True  
    for i in range(len(s) // 2):  
        if s[i] != s[len(s) - 1 - i]:  
            is_pal = False  
            break  
    return is_pal  
  
def is_palindrome3(s):  
    n = len(s)  
    return s[n-1::-1] == s  
  
def is_palindrome4(s):  
    return s == s[::-1]  
  
def is_palindrome5(s):  
    return s == ''.join(reversed(s))
```

توضیح راه اول: از ۱ تا نصف پیش میرم و هر بار با طرف راستیش مقایسه می‌کنم. توجه دارین که که آخرین ایندکس همیشه یکی از طول کمتره؟ و همچنین ایندکس‌های متناظر زیر با هم مقایسه میشن:

```

i = 0 and i = n - 1
i = 1 and i = n - 2 → i = (n - 1) - 1 → i = (n - 1) - i
i = 2 and i = n - 3 → i = (n - 1) - 2 → i = (n - 1) - i
i = 3 and i = n - 4 → i = (n - 1) - 3 → i = (n - 1) - i

```

پس درواقع هر بار داره با $n-1-i$ مقایسه میشه. پس برای همین شرط رو گذاشتم که اگر اینوری با اونوری برابر نشد، همونجا ریترن کنه False رو. (چون یکیشم اوکی نباشه، palindrome نیست.) وگرنه در پایان ریترن کنه True رو. پس درواقع True صرفاً زمانی ریترن میشه که کل for رو رفته باشه و عملاً به هیچ نابرابری نخورده باشه. (همه برابر شده بودن). اینطوری نشون داده میشه که کل for رو رفتی، ریترن False ای صورت نگرفت، پس درواقع یعنی palindrome هست و نیازه True رو ریترن کنم.

توضیح راسمه: یادتون هست که این علامتای «==»، «!=» و...، درستی (True) یا نادرستی (False) رو بر می گردونن. پس درواقع مقدار boolean اینکه آیا یه استرینگ با معکوشش برابره یا نه رو چاپ می کنه.

12) Comment

معمولاً برنامه نویسای جاهایی که نیاز به توضیح داره، یه سری توضیحات می ذارن. بهش می گن کامنت. کامنت صرفاً برای خوانایی بیشتر کد هست و اجرا نمیشه. 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 می خواین کامنت رو توی همون خط بگذارین، با دو تا فاصله از کد قرار بدین. کامنت رو می تونین توی یه خط تنها هم بگذارین. مثلاً:

```

# Count the number of a char 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: char, string  
    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>

کامنت گذاری 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! مثلاً یه بار یکی اومده بود برای تک به تک خط‌های کدش کامنت گذاشته بود. حتی یه متغیر ساده هم که تعریف کرده بود، کامنت گذاشته بود. اینطوری مثلاً:

```
string = 'hello' # I have defined a variable here
```

خب خسته نباشی! یه متغیر تعریف کردی! این دیگه که کامنت نمی‌خواست! کامنت رو جایی می‌ذارن که ممکنه کد مبهم باشه. کامنت رو می‌ذارن برای خوانایی بیشتر. نه اینکه هر کاری کردی یه کامنت بداری!

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 if ... in) 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 ۱:

```
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 بگیریم 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 خلاقیته! هوشمندتر باشین! یا حتی من بهتون slicing رو گفتم. پس بدون تابع سعی کنین پیاده‌سازی 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.

پاسخ ۲:

```
def uppercase_count(s):
    count = 0
    upper_chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    for char in s:
        if char in upper_chars:
            count += 1
    return count

inp_string = input("Enter a string: ")
print(uppercase_count(inp_string))
```

یه متغیر ساختم که تمام حروف بزرگ رو توش گذاشتم. بعد گفتم برای تک تک کرکتهای string ام، اگر کرکتر توی uppercase_chars بود، یه دونه به تعداد اضافه کن. راه بهتر:

```
def count_common(s1, s2):
    count = 0
    for c in s1:
        if c in s2:
            count += 1
    return count

upper_chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
inp_string = input("Enter a string: ")
print(count_common(inp_string, upper_chars))
```

پاسخ ۳:

```
inp = input("Enter a string: ")
sub = input("Enter a substring: ")

if sub in inp:
    print("Yes")
else:
    print("No")
```

به همین سادگی! میگیریم اگر دومی توی اولی بود، چاپ کن Yes. اگر نه، چاپ کن No.

پاسخ ۴:

خب فرض کنیم زیرمجموعه یا sub یا همون استرینگ که می‌خوایم ببینیم توی اولی هست یا نه، سایش ۳ هست. باید روی اولی حرکت کنیم و هی سه تا سه تا جدا کنیم و ببینیم با sub برابره یا نه؟

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، سایز sub رو می‌گذاریم و با for یکی یکی میریم جلو:

```
def if_in_string(inp, sub):  
    sub_size = len(sub)  
    for i in range():  
        if inp[i:i+sub_size] == sub:  
            return True  
  
    return False
```

خب for رو نمی‌دونم تا کجا پیش ببرم. توی پایتون اگر به ایندکسی بخواین دسترسی پیدا کنید که خارج از مکریموم یا مینیموم ایندکس استرینگ باشه، بهتون ارورد میده و میگه توی رنج نیست! 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 دارم میگم از i تا i + sub_size پیش برو.

خب بیایم مقدار بدیم که بتونیم درکش کنیم:

```
inp = "0123456"  
sub = "123"
```

خب سایز sub ما ۳ هست. حداکثر ایندکس input ما هم ۶ هست. پس i + sub_size حداکثر باید ۷ بشه. چون میگیم از i تا قبل i + sub_size پیش برو. یعنی درواقع بخوایم تا آخرین ایندکس input رو پوشش بدیم، باید بگیم مثلاً inp[4:7] == sub. قبول دارین این آخرینش دیگه؟

یعنی آخری، i چهار بوده:

```
inp[i:i+3] == sub
```

یا درواقع می‌تونیم بگیم که:

```
i + 3 = inp_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
```

```
inp_size - sub_size + 1
```

یا درواقع:

```
for i in range(len(inp) - sub_size + 1):
```

پس کد رو تکمیل کنیم:

```
def if_in_string(inp, sub):
    sub_size = len(sub)
    for i in range(len(inp) - sub_size + 1):
        if inp[i:i+sub_size] == sub:
            return True

    return False

inp = input("Enter a string: ")
sub = input("Enter a substring: ")
print(if_in_string(inp, sub))
```

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 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 کوچکت نیست؟!^{۶۵}

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.

حالا فرض کنین من نام و نام‌خانوادگی رو جداگانه گرفتم و می‌خوام توی یه متغیر با یه فاصله از هم قرارش بدم. مثلاً:

```
first_name = 'Bruce'
last_name = 'Schneier'
```

رو داریم. و می‌خوام یه full_name داشته باشیم که اینطوری باشه:

```
full_name = 'Bruce Schneier'
```

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)
```

^{۶۵} در بعضی از زبان‌ها گیر نمیدن ولی در زبانای زیادی ارور میخورین.

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 بین دو `string`، می‌تونین علامت جمع هم بگذارین. فقط توجه کنین که علامت جمع بین دو چیز یکسان کار می‌کنه. یعنی شما نمی‌تونین عدد ۲ رو با یه `string` جمع کنین! همیشه یه عدد رو با `string` جمع زدن! ولی `string` رو با `string` می‌تونین! مثال:

```
print('hi' + 2)
print('hi' + '2')
```

پرینت اولی ارور میده چون نمی‌تونین یه عدد رو با یه استرینگ جمع بزنین. اما پرینت دومی درسته. چون دو تا استرینگ رو درام با هم جمع می‌زنم.

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!

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).^{۶۶}

توجه! از تابع `len` نمی‌تونید استفاده کنید! خودتون باید تابعی به نام «`length`» تعریف کنین که یه استرینگ می‌گیره و یه عدد که به معنای طول اون هست رو بر می‌گردونه.

input:

01010101101110

۶۶ به این کار می‌گن «Binary Sign-Extension».

0000000000000000000000001010101101110

101010

111111111111111111111111101010

dabce

input:

abcde
2

output:

cdabe

یه بار d اومده اول و «dabce» ساخته شده. یه بار دیگه هم rotate صورت گرفته و «cdabe» ساخته میشه.

به طور کلی روتیت‌های مختلف به ترتیب اینطورین:

abcde
dabce
cdabe
bcdae
abcde

۶.۱- یه string و یه عدد بگیرین. به تعداد عدد، string رو rotate کنین. (بچرخونین. یعنی کرکتر اول رو ببرین آخر).

input:

abcd
1

output:

bcda

input:

abcd
2

output:

cdab

input:

abcd
4

output:

abcd

input:

abcd

10

output:

cdab

۷- یک عدد گرفته و فاکتوریلش 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. چندان 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:⁶⁹

93326215443944152681699238856266700490715968264381621468592963895

21759999322991560894146397615651828625369792082722375825118521091

68640000000000000000000000000000

output: 24

input: 76

factorial:

18854947016660502549879322608611465582303945353793293356724879829

6184404349553792311772997222400000000000000000000

output: 18

۹-

```
import math
print(math.factorial(100))
```

۶۹ البته خودتونم می‌تونستین با استفاده از

حاصل فاکتوریل یه عدد رو به دست بیارین که ببینین آیا تابع فاکتوریتون رو درست نوشتین یا نه؟

پاسخنامه:

پاسخ:

اول یه تابع length رو می نویسم:

```
def length(s):  
    cnt = 0  
    for char in s:  
        cnt += 1  
    return cnt
```

چون طول رو نمی دونم از ... in for استفاده می کنم که هر بار که روی تک تک کرکترها حرکت می کنم، یکی به تعداد اضافه کنم.

بعدش بیایم یه عدد باینری بگیریم. این عدد باینری رو به صورت همون string می گیرم. چرا؟ چون می دونم این استرینگ هست که می تونم 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 اندازه ای که با عددمون بشه ۳۲ رقم. خب پس ۳۲ منهای سایز عددمون تا ۱ اضافه شه.

```
def length(s):  
    cnt = 0  
    for char in s:  
        cnt += 1  
    return cnt  
  
def sign_extend(n, bin_str):  
    if bin_str[0] == "1":  
        bin_str = "1"*(32-n) + bin_str  
  
bin_str = input()  
n = length(bin_str)
```

خب برای حالتی که صفر هم تهش بود، اضافه می کنیم:

```
def length(s):
    cnt = 0
    for char in s:
        cnt += 1
    return cnt

def sign_extend(n, bin_str):
    if bin_str[0] == "1":
        bin_str = "1"*(32-n) + bin_str
    elif bin_str[0] == "0":
        bin_str = "0"*(32-n) + bin_str
    return bin_str

bin_str = input()
n = len(bin_str)
print(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 یه 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.^{۷۰}

```
def length(s):
    cnt = 0
    for char in s:
        cnt += 1
    return cnt

def sign_extend(n, bin_str):
    if n != 0 and bin_str[0] == "1":
        bin_str = "1"*(32-n) + bin_str
    elif n != 0 and bin_str[0] == "0":
        bin_str = "0"*(32-n) + bin_str
    return bin_str

bin_str = input()
n = length(bin_str)
print(sign_extend(n, bin_str))
```

ببینیم باز آیا مشکلی نداره؟

^{۷۰} 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!

اگر ورودی یه اینتر خالی باشه، یه خط خالی چاپ میشه. که نباید میشد! (کلاً هر وقت پرینت داشته باشیم، یه `new line` هم انگار داریم). پس یه `if` هم برای خارجش می‌خوام:

```
def length(s):
    cnt = 0
    for char in s:
        cnt += 1
    return cnt

def sign_extend(n, bin_str):
    if n != 0 and bin_str[0] == "1":
        bin_str = "1"*(32-n) + bin_str
    elif n != 0 and bin_str[0] == "0":
        bin_str = "0"*(32-n) + bin_str
    return bin_str

bin_str = input()
n = length(bin_str)
if n != 0:
    print(sign_extend(n, bin_str))
```

پاسخ ۲:

در نگاه اول شاید بگیم که:

```
s = '1111111'
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 نمیشه یه قسمت رو عوض کرد، بگیم استرینگ جدید رو از کانکتینیت کردن (جمع و چسبوندن کرکتر و رشته به هم) می‌سازم. خب برای اینکه کارم ساده‌شه، همیشه سعی می‌کنم با ایندکس (و نه جایگاه کرکتر که یکی از ایندکس‌های پیش‌تره)، پیش برم. استرینگ جدید اینطور ساخته میشه: تا قبل ایندکس `i` مقدار اولیه رو داره. بعدش کرکتر جدید قرار می‌گیره. بعدش از ایندکس `i + 1` به بعد، مقدار قبلی رو داره.


```
inp_str = input('enter a string: ')
char = input('enter a character: ')
n = int(input('enter the place of the char you want to change: '))
i = n - 1
new_str = inp_str[:i] + char + inp_str[i+1:]
print(new_str)
```

همونطور که دیدین، من مستقیم با کرکتر n ام کار نکردم. بلکه با ایندکس (که یکی از جایگاه کمتره)، کار کردم که ساده‌تر باشه برام.

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 ۳:

قاعدتاً باید روی تک‌تک کرکترها حرکت کنم و تشخیص بدم اون کرکتر عدد فرده یا نه؟

روش اول:

```
def odd_digit_count(num_str):
    count = 0
    for char in num_str:
        if char in '13579':
            count += 1
    return count
```

روی تک‌تک کرکترها حرکت می‌کنم و هر وقت که کرکتری توی رشته «۱۳۵۷۹» بود، فرده و یکی اضافه می‌کنم.

توجه! چون درون متغیر `num_str`، عدد به صورت استرینگ ذخیره شده، اسمش هم جووری نوشتم که مشخص باشه استرینگ هست.

روش دوم:

```
def odd_digit_count(num_str):
    count = 0
    for char in num_str:
        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 فرده یا نه.

پاسخ ۴:

روش اول:

```
def reverse(s):
    return s[::-1]

inp_str = input()
print(reverse(inp_str))
```

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.

حالا تلاش کنین فقط سه تا کرکتر اول یه استرینگ رو برعکس کنین:

```
s = '12345'
first3 = s[:3]
print(first3[::-1])
```

روش دوم:

شاید ما دلمون بخواد که توی تابع پرینت رو انجام بدیم.

بله! میشه توی تابع پرینت کرد! تابع یه سری کد هست که بله توش می‌تونیم پرینت انجام بدیم.

```
def reverse(s):
    for i in range(len(s) - 1, -1, -1):
        print(s[i])

inp_str = input()
print(reverse(inp_str))
```

اما خروجیشو به ازای ورودی «abcd» ببینیم:

```
d
c
b
a
None
```

مشکلش چیه؟

خب اولین مشکلش آینه که None چاپ شده. به نظرتون دلیلش چیه؟ چرا None چاپ شده؟
یادتونه موقع توضیح تابع بهتون گفتم که تابع موجودیه که یه چیز می‌گیره و یه چیز پس میده.
همچنین وقتی می‌گیم «`print(reverse(inp_str))`» درواقع داریم می‌گیم که حاصل تابع `reverse` رو چاپ کن. یا درواقع داریم می‌گیم که پرینت کن حاصلی که تابع `reverse` برای ما `return` می‌کنه رو.
خب نگاهی به تابع بندازیم. آیا این تابع چیزی رو ریترن می‌کنه؟ خیر!
آیا می‌شد تابع چیزی رو ریترن نکنه؟ بله! (توضیحش دادیم قبلاً)
برای همین وقتی چیزی ریترن نمیشه، درواقع داره «هیچی» یا همون «None» ریترن میشه. برای همین «None» قصه ما چاپ شده.
درواقع پس فهمیدیم که وقتی ریترنی صورت نمی‌گیره، عملاً «None» یا همون «هیچی» بر می‌گرده و پرینت هم همینو داره چاپ می‌کنه.
اگر نمی‌خواهیم این «None» چاپ شه، باید صدازدن تابع رو از پرینت بیرون بیاریم. یعنی عادی صدازش بزنینم. توی پرینت نباشه:

```
def reverse(s):
    for i in range(len(s) - 1, -1, -1):
        print(s[i])

inp_str = input()
reverse(inp_str)
```

حالا مشکل حل شد. برگردیم به مشکل دوم.

مشکلش آینه که هر بار که `print` انجام میشه، به واسطه خاصیت `print`، حروف توی خط‌های مجزا چاپ میشن. شاید من نخواهم که در خط مجزا چاپ شن. دوست دارم در کنار هم چاپ شن و فاصله‌ای بینشون نیوفته. چیکار کنم؟
پایتون راه حل داده! گفته که 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 بزنام؟ اصلاً فاصله ندم؟ چیکار کنم. اینطوری بهش می‌گیم

```
print('a', 'b', end='$')
print('c')
```

output:

```
a b$c
```

میگه خط پرینت اولی رو که اجرا کردم، انتها یا «end»ش چی پرینت کنم؟ بهش میگم که علامت «\$» رو پرینت کن. (دیگه به صورت پیشفرض که می‌رفت خط بعدی و enter می‌زد رو انجام نمیده). یا مثلاً می‌تونیم بگیم:

```
print('a', end='00')
print('c')
```

output:

```
a00c
```

بهش می‌گم که بعد چاپ کردن «a» صرفاً دوتا صفر بذار. همین! بعدش c رو در ادامه اون دو تا صفر چاپ می‌کنه.
مثال‌های دیگه:

```
print('hi', end='123')
print(2, end='00')
print('bye')
```

output:

```
hi123200bye
```

```
print('hi', end='\t')
print('bye')
```

output:

```
hi      bye
```

بهش می‌گم که بعد چاپ کردن «a» صرفاً دوتا صفر بذار. همین! بعدش c رو در ادامه اون دو تا صفر چاپ می‌کنه.

```
print('a', end='00')
print('c')
```

output:

```
a00c
```

یادتونه `\n` و `\t` چی بودن؟ اینا یه کرکتر خاص بودن که معنای خاص داشتن. `\n` یعنی برو خط بعد و `\t` یعنی به اندازه یه `tab` کیبورد، فاصله بده.

```
print('hi', end='\n\t\t')
print('bye')
```

output:

```
hi
      bye
```

بهش می گم که وقتی «hi» رو چاپ کردی، اول یه اینتر بزنی. بعدش دوتا تب بزنی. توی پرینت بعدی هم میاد `bye` رو چاپ می کنه. (چون قبلش دوتا تب بود، با فاصله چاپ میشه.)

```
hi123200bye
```

```
print('hi', end='')
print('bye')
```

output:

```
hibye
```

بهش می گم که بعد چاپ کردن «hi» یه استرینگ خالی چاپ کن! (یعنی انگار هیچی چاپ نکن و هیچ کاری نکن!) برای همین «bye» در ادامه همون قبلی چاپ میشه.

حالا بریم سراغ سؤال اصلیمون. به نظرتون چیکار کنم که بعد هر پرینت نره خط بعدی و همونجا بمونه. چون که می خواستم که کرکترای بعدی، به قبلیا بچسبن. خب راحت می نویسم که بعد پرینت هیچی انجام نده:

```
def reverse(s):
    for i in range(len(s) - 1, -1, -1):
        print(s[i], end='')

inp_str = input()
reverse(inp_str)
```

میگیم هر بار پرینت تموم شد. هیچی چاپ نکن. که بعدش میای دوباره پرینت کنی، پرینت جدید، ادامه پرینت قبلی میاد.

حالا اجرا کنین میبینین که کرکترها پشت هم چاپ میشن و همچنین هیچ اینتری نداره:

```
dcba
```

این هم یه نکته دیگه از `print` که در زمان نیاز یادش گرفتیم. نه اینکه توی پرینت تمام چیزاشو یهو بهتون حفظی وار بگم که فراموش کنین. هرچیزی باید زمان نیازش یاد گرفته شه (:

پرینت یه پارامتر دیگه هم می تونه بگیره به نام «`sep`» که میگه بین چیزایی که توی یه پرینت دارن انجام میشن، چیکار کنم؟ 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:

```
print('hi', 'bye', sep='0')
```

output:

```
hi0bye
```

بهش می گم که بین چاپ هایی که داری توی یه پرینت انجام میدی، صفر بذار.

```
print('hi', 'hello', 'bye', sep='00')
print('next line')
```

output:

```
hi00hello00bye
next line
```

بهش می گم که بین چاپ هایی که داری توی یه پرینت انجام میدی، دوتا صفر بذار. همچنین مقدار `end` هم ندادم بهش و به صورت پیش فرض یه اینتر زده خودش.

```
print('hi', 'hello', 'bye', sep='', end='\t')
print('next line')
```

output:

```
hihellobye      next line
```

بهش می‌گم که بین چاپ‌هایی که داری توی یه پرینت انجام میدی، هیچی چاپ نکن. به هم بچسبونشون. همچنین مقدار `end` هم گفتم یه تب. یعنی بعد انجام این خط پرینت، یه تب بزن. پس پرینت بعدی با یه تب از این پرینت چاپ میشه.

روش سوم:

من نمی‌خوام توی تابع پرینت کنم. من می‌خوام مقدار ریورس شده رو برگردونم. قاعدتاً اگر بخوام یه استرینگ که ریورس استرینگ قبلی رو داره ریترن کنم (مشابه روش ۱)، نیاز دارم استرینگ ریورس رو بسازم. اما نمی‌خوام از روش یک برم. می‌خوام خودم با `for` بسازمش. قاعدتاً می‌تونم یه `for` بزنم و از آخر به اول بیام و دونه‌دونه کرکتر رو به ترتیب ریورس و برعکسی که می‌خوام داشته باشم. خب این کرکتر رو هم می‌تونم یه جا ذخیرش کنم که تو هوا نباشن! برای همین یه متغیر می‌سازم که اول یه رشته خالی هست. توی `for` دونه‌دونه کرکتر رو به تهش اضافه می‌کنم که وقتی `for` کامل شد، استرینگ ریورس من ساخته شده باشه:

```
def reverse(s):
    rev_s = ''
    for i in range(len(s) - 1, -1, -1):
        rev_s += s[i]

inp_str = input()
print(reverse(inp_str))
```

درواقع کاری که کردم آینه که یه استرینگ خالی در نظر گرفتم، بعدش توی `for` دونه‌دونه کرکترایی که از ته میومدم رو بهش اضافه کردم و پُرش کردم. این تکنیک خیلی رایجه که شما یه استرینگ خالی در نظر می‌گیری و با چیزایی که می‌خوای پرش می‌کنی. منم با کرکترایی که از آخر اومدم به اول پرش کردم.

پاسخ ۵:

```
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 ۶:

خب ما باید عنصر یکی مونده به آخری (ایندکس منفی ۲) رو بیاریم اول و بقیه استرینگ، خودش باشه. یعنی s جدید: ایندکس منفی ۲ اولی + از صفر تا قبل منفی ۲ + ایندکس آخر

```
def custom_rot(s, n):
    for i in range(n):
        s = s[-2] + s[:-2] + s[-1]
    return s

s = input()
n = int(input())
print(custom_rot(s, n))
```

پاسخ ۶.۱:

```
def rotate_string(s, n):
    for i in range(n):
        s = s[1:] + s[0]
    return s

string = input()
rotate_count = int(input())
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.

پاسخ ۷:

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 می‌روم. 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 کلی 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 اینه که یه تابع بنویسم و هم فاکتوریل حساب کنه و هم تعداد صفرا!!

بهتره دو تابعش کنم. یکی فاکتوریل حساب کنه و یکی تعداد صفر. یا بهتر. تابع دوم، تعداد یه کرکتر دلخواه در سمت راست یه استرینگ رو به دست بیاره که تابع ما حالتی کلی‌تر داشته باشه.

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 کلی‌تر.

از ته شروع می‌کنم و هی میام چپ‌تر. هر بار به کرکتر مورد نظر خوردم، یکی به تعداد اضافه می‌کنم و اولین کرکتر غیر کرکتر خاصم رو که دیدم از حلقه‌ام می‌پرم بیرون.

```
def right_chars(s, char):  
    count = 0  
    for i in range(len(s)-1, -1, -1):  
        if s[i] == char:  
            count += 1  
        else:  
            break  
    return count
```

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_chars(s, char):
    count = 0
    for i in range(len(s)-1, -1, -1):
        if s[i] == char:
            count += 1
        else:
            break
    return count

n = int(input())
print(right_chars(str(factorial(n)), '0'))
```

گفتیم هر وقت که پرانتز تو در تو دیدین، برین از داخلی‌ترین پرانتز شروع کنین. می‌گه که اول فاکتوریل بگیر. بعد تبدیل به استرینگ کن و بعد تعداد right_chars های «0» رو به دست بیار.

البته می‌شد با ایندکس منفی هم پیش رفت. یعنی:

```
def factorial(num):
    result = 1
    for i in range(1, num + 1):
        result *= i
    return result

def right_chars(s, char):
    count = 0
    i = -1
    while s[i] == char:
        count += 1
        i -= 1
    return count

n = int(input())
print(right_chars(str(factorial(n)), '0'))
```

روش دوم:

اما بدون کمک string انجامش بدیم. یعنی با عدد. خب چه جوری من صفرا رو بشمرم؟ قبول داریم که صفرا 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 شیفت دادن به سمت راست. یعنی با تقسیم صحیح بر عدد ۱۰. یعنی رقم سمت راست رو می‌ریزم بیرون.

```
def factorial(num):  
    result = 1  
    for i in range(1, num + 1):  
        result *= i  
    return result  
  
def right_chars(num):  
    count = 0  
    while num % 10 == 0:  
        count += 1  
        num //= 10  
    return count  
  
n = int(input())  
print(right_chars(str(factorial(n))), '0')
```

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.
در واقع 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 بینیم اون عدد ما بر ۵ بخش پذیر هست یا نه؟ تا زمانی که ۵ توی خودش داشت،
هی یکی به تعداد صفر 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 ۹:

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 می‌بینیم 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

می‌سازم و با `for` پرینتش می‌کنم. 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 می کنه. حالا با این، سؤالمون رو تمیزتر حل می کنیم:

```
def print_square(n):  
    print('*' * n)  
  
    middle = ' ' * (n-2)  
    middle = '*' + middle + '*'  
    for i in range(n-2):  
        print(middle)  
  
    print('*' * n)  
  
print_square(5)
```

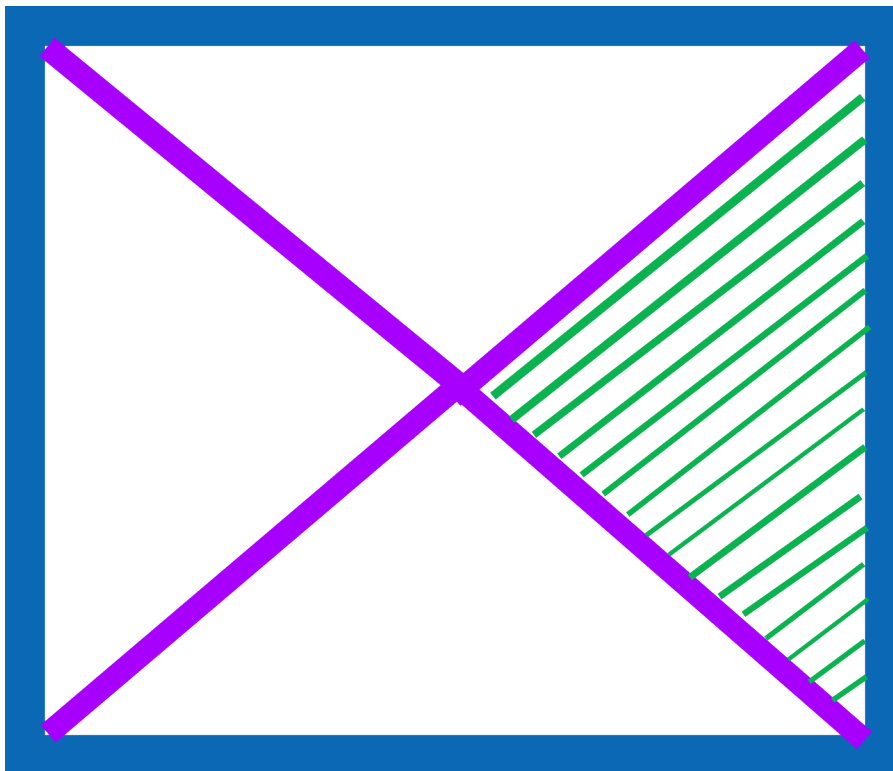
تمیزتر نشد؟

تمرین بیشتر:

<https://quera.org/problemset/283/>

<https://quera.org/problemset/296/>

اگر توی سؤال دومی نمی دونین که یه چهارم سمت راست یعنی چی، این شکل رو ببینین:



درواقع می‌گه قطرا رو بکش. قطرا رو که با هشتگ کشیدی، سمت راست رو هم پر کن.

- **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 هستن، بعد حروف کوچیک. حتی عدد کرکتری هم، عدد مخصوص به خودشو داره. حتی علامت سؤال (?) و ...

پایتون برای تبدیل این چیزا به هم خودش توابعی داره. توابع chr و ord. 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 کنین من یه string دارم و می‌خوام با استفاده از یه تابع، حروف بزرگشو تبدیل به حروف کوچیک کنم. سعی کنین خودتون اول بنویسینش و بعد نگاه پاسخ کنین. راهنمایی ۱: فاصله بین حروف بزرگ و کوچیک 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(s):
    for i in range(len(s)):
        if s[i] >= 'A' and s[i] <= 'Z':
            s[i] = chr(ord(s[i]) + 32)
    return s
```

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 کرکتر که ببینم درست کار می‌کنه؟ (چون فاصله یه کرکتر حروفی نیست و همونطور که یادموئه 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
```

نوشته که شما به یه آیتم از `string`، بیای `assignment` انجام بدی. (`assignment` یعنی مساوی یه چیزی قرار بدی). 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.

- چطور این کار رو انجام بدیم؟

+ خب بذارین فکر کنیم. ما چه ساختارهایی داریم؟ if, else, elif, for, while و من نمی‌تونم یه

for بزنم و دونه‌دونه اونایی که uppercase هستن رو عوض کنم. پس چیکار کنم؟ آیا می‌تونم پیام یه استرینگ جدید بسازم و اون کرکترایی که عادی هستن، خودشون و اونایی که بزرگن رو پیام تبدیل به کوچک کن بریزم توش؟ یعنی پیام دونه دونه روی کرکترهای این string عادیمون حرکت کنم. و دونه دونه به استرینگ جدید اضافه کنم:

```
def tolower(s):
    lower_s = ''
    for char in s:
        if char >= 'A' and s[i] <= 'Z':
            lower_s += chr(ord(s[i]) + 32)
        else:
            lower_s += s[i]
    return lower_s
```

راه دوم:

بدون استفاده از ایندکس و با استفاده از for ... in:

```
def tolower(s):
    lower_s = ''
    for char in s:
        if char >= 'A' and char <= 'Z':
            lower_s += chr(ord(char) + 32)
        else:
            lower_s += char
    return lower_s
```

حتی می‌تونستیم قسمت if رو اینطورم بنویسیم:

```
def tolower(s):
    lower_s = ''
    for char in s:
        if 'A' <= char <= 'Z':
            lower_s += chr(ord(char) + 32)
        else:
            lower_s += char
    return lower_s
```

Vigenère cipher (16th 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 هشتیمین 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 کوچیکن. 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؟ قاعده‌تاً 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 می‌خوانین؟

من به 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 به متن می‌گیره و به 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 به متغیر به نام encrypted_text می‌سازم:

```
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
    return encrypted_text
```

می‌گم اگر اون کرکتر من (یعنی همون plain_text[i]) به صورت کرکتری بود، به کار انجام بده. اگر نبود، همون توی کرکتر رمزی ماست. بلاک else می‌گه که خود کرکتر، همون کرکتر رمزیه و کاری نکن. و در نهایت encrypted_char (کلمات رمز شده) مرحله به مرحله به آخر encrypted_text اضافه میشن. و در آخر هم متن رمزی رو ریترن می‌کنم.

خب دیدین؟ لزوماً قرار نیست همون اول if رو کامل کنم. فعلاً else رو نوشتم. قسمت بلاک if رو بعداً کامل می‌کنم.

تازه به تابعی رو نوشتم که هنوز تعریفش نکردم. یعنی دیدم به نیاز به به 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

    return encrypted_text
```

خب میایم قسمت بلاک if رو بنویسیم:
فعلاً برای سادگی کار، میام میگم که ببینم چقدر باید بره جلو.

```
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

    return encrypted_text
```

میگم که بیا ord برای کرکتر plain_text ما حساب کن. منهای ord کرکتر a کن که انگار کرکتر از شماره گذاری شده. (برای سادگی که بفهمم چقدر جلوتر از a هست.) بعدش بعلاوه ایندکس متناظرش در key کن. اونم منهای a کن که اونم از صفر انگار نامگذاری شه. اینطوری می تونم بفهمم که چقدر باید بره جلو.

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 می نویسمش:

```
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

    return encrypted_text
```

خب حالا چی نیازه؟

اینکه چک کنیم آیا از ۲۶ زده بیرون یا نه؟ (شماره گذاریمون 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:

```
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
            else:
                encrypted_char = plain_text[i]

            encrypted_text += encrypted_char

    return encrypted_text
```

حالا دلیل اینکه منهای a کردم رو فهمیدین؟ منهای a کردم که در آخر بتونم بهتر باقی‌مونده بگیرم. تا اینجا ما به مقدار عددی از ۰ تا ۲۵ به دست آوردیم. خب چطور کرکترش رو بسازیم؟ کافیه این مقدار عددی 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
```

خب خیلی هم عالی! حالا که اینو ساختیم، بیایم حالتی رو بسازیم که کلید کوچکتر از متنه و کلید باید تکرار بشه.

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 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 که کد تکمیل شد، بیایم `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'
```

```
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))
```

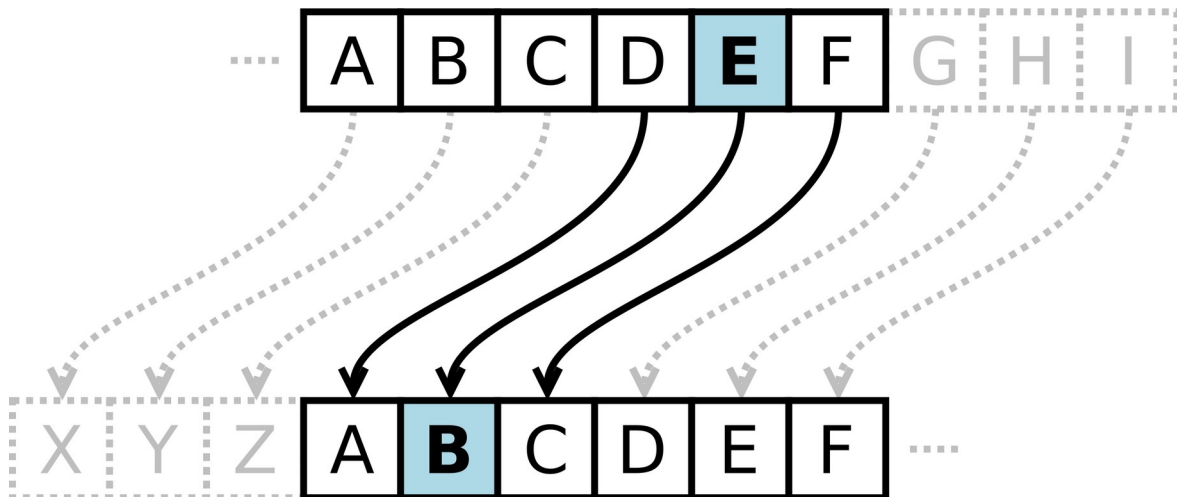

حالا به من بگین که برنامه‌نویس خوب حواسش به چی هست؟

+ 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 میدن 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!

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 یه 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!

۹/۹/۹۵

خط اول: یه متن که می‌تونه شامل کرکتر کوچک، بزرگ و یا کرکترهای غیر حروف الفبا مثل فاصله و علامت سؤال و... باشه

خط دوم: تعداد شیفت که می‌تونه هر عدد صحیحی باشه!
توجه! کرکترهای غیرحروف الفبایی نیاز به رمزشدن ندارن!

تست کیس^{۷۱}:

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:

Fc eb exa xkvqefkd zlkcfabkqfxi ql pxv, eb tolqb fq fk zfmeho, qexq fp, yv pl zexkdfkd qeb loabo lc qeb ibqqbop lc qeb ximexybq, qexq klq x tloa zlrta yb jxab lrq.

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 از به PascalCase کنین.

۷۱ با کمک وبسایت زیر تولید کردم:

<https://cryptii.com/pipes/caesar-cipher>

فقط اگر می‌خوان ۳ تا شیفت بدین چپ، به این وبسایت منفی ۳ بدین.

اگر هیچکدام 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 متفاوت!

پاسخنامه:

پاسخ ۱:

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 کرکتر دونه‌دونه حرکت می‌کنم. اگر کرکترم جزء حروف الفبا بود، باید رمزش کنم. در غیر این صورت (خط ۹)، میام به ciphertext، کرکتمو اضافه می‌کنم.

حالا قسمت اگر حروف الفبا بود رو می‌کنم:

اگر is_lower (حروف کوچک) 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! البته بهتر بود یه pass می‌داشتیم که اروری هم نده فعلا.

```
def caesar_encrypt(plaintext, shift):
    ciphertext = ""
    for char in plaintext:
        if is_alpha(char):
            if is_lower(char):
                pass
            else: # upper case
                pass
        else:
            ciphertext += char
    return ciphertext
```

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. برای حروف بزرگ هم منهای «A»:

```
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

            else: # upper case
                cipher_char = ord(char) - ord('A') - shift

        else:
            ciphertext += char
    return ciphertext
```

حالا آیا درسته؟ توی عکس اگر نگاه کنیم، یه وقتایی ممکنه از سمت چپ بزنه بیرون. برای همین باید برای هر دو چک کنم که بعد منفی کردن، زد بیرون از رنج، اصلاحش کنه. این یعنی چی؟ یعنی عدد که باید از ۰ تا ۲۵ می‌بود، شده منفی. پس باید اصلاح شه:

```
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

            else: # upper case
                cipher_char = ord(char) - ord('A') - shift
                while cipher_char < 0:
                    cipher_char += 26

        else:
            ciphertext += char
    return ciphertext
```

الآن یه عدد در رنج ۰ تا ۲۵ دارم که نیاز به تبدیل به کرکتر ASCII کنم که بتونم چاپشون کنم. برای کوچیک‌ها بعلاوت «a» و برای بزرگ‌ها بعلاوه «A» و در نهایت chr گرفتن ازش:

```
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: # upper case
                cipher_char = ord(char) - ord('A') - shift
                while cipher_char < 0:
                    cipher_char += 26
                cipher_char = chr(cipher_char + ord('A'))

        else:
            ciphertext += char
    return ciphertext
```

خب کد ما تکمیل شد. حالا توابعی که استفاده کرده بودم رو می‌نویسم و کد رو 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: # upper case
                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))

```

مطمئنأ یادمون نرفته که در if بیرونی، cipher_char ساخته‌شده رو بریزم توی ciphertext.
(هایلایت کردم بهتر متوجه شین)

پاسخ ۲:

برای تبدیل pascal به snake باید هرجا حروف بزرگ دیدیم (جز حرف اول ← ایندکس صفر)، حرف بزرگ رو کوچیک 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
            pascal_text += toupper(snake[i])
        else:
            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 بگیریم 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 برین. 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 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 نمیشه که کرکت 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 کوچیک 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 می‌گذارم که اولین تکرار رو حذف نکنم.

اینطوری که وقتی پیدا کرد، چک می‌کنه که is_first مقدارش False باشه. اگر باشه، یعنی اولین تکرار نیست؛ پس تکرار رو حذف می‌کنه. اما اگر باشه، یعنی بار اوله و نباید تکرار رو حذف کنه. ولی باید بپره بره is_first رو False کنه که برای دفعه‌های 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 می‌کردیم که آخرین مقایسه برای مثلاً یه عنصر ۴ عضوی اینه:

johnjohn

یعنی آخرین i ما، len(text) - len(duplicate) هست. (اگر نگاه کنین ۸ کرکتره. یعنی ۷ ایندکس. ما آخرین رو می‌خوایم از ایندکس ۴ تا ۷ مقایسه کنیم. پس درواقع می‌تونیم بگیم 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؟ برنامه‌م کندتر 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 + 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 یه چیزی به عنوان اینکه نامعتبره چاپ کنه و یه 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 بودا!)
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.

• String methods

پایتون یه سری توابع درونی داره که میشه ازشون استفاده کرد و کار رو خیلی سریع‌تر کرد. یه خرده با توابع عادی فرق دارن.^{۷۲} درواقع مخصوص string ها هستن. یعنی درواقع برای string ها هستن. برای همین با یه نقطه به string ها پیوند می‌خورن! این‌ها رو method (مِتُد) صدا می‌زنیم. مثلاً:

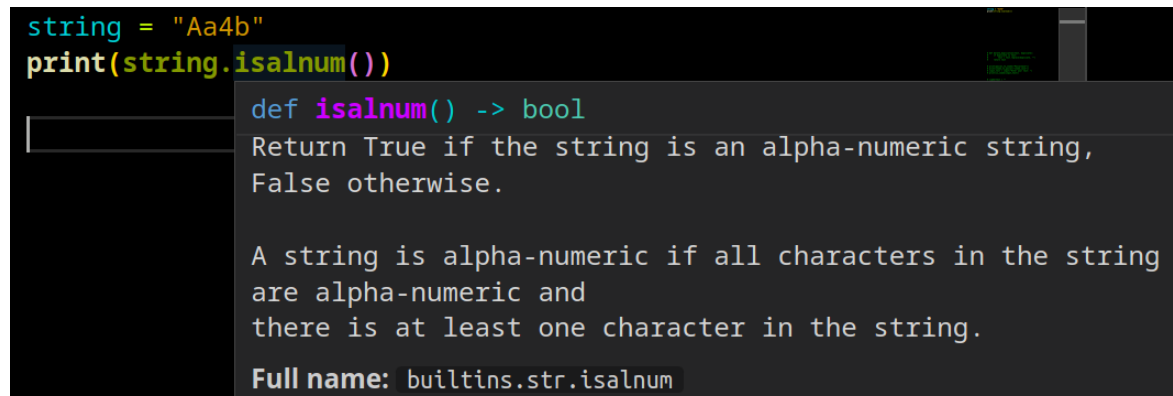
```
string = "ABcdE"
print(string.isalpha()) # True
```

^{۷۲} برای اونایی که با برنامه‌نویسی شیء‌گرا هستن، اینا متدهای built-in کلاس string هستن!

متد `isalpha`، به ما می‌گه که آیا همه کرکترای این `string`، جزء حروف الفبا هستن یا نه؟ مقدار `boolean` بر می‌گدونه. 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 می‌تونین 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`

مثلاً اینجا نوشته که `isalnum`، یه `bool` برمی‌گردونه. 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` بر می‌گردونه. اگر نبود، `False` بر می‌گردونه.

متدهای مهم دیگه:

```
string = "aBC"
print(string.islower())
```

اگر صرفاً حروف کوچک الفبا باشه، `True`. اگر نه `False`.

```
string = "FS"
print(string.isupper())
```

اگر صرفاً حروف بزرگ الفبا باشه، `True`. اگر نه `False`.

```
string = "743928"
print(string.isdigit())
```

اگر صرفاً `digit` (رقم) باشه، `True` بر می‌گردونه.

```
string = "223"
print(string.isnumeric())
```

اگر صرفاً عدد باشه.

فرق بین isdigit و isnumeric چیست؟

راحت می‌تونین سرچ کنین و فرقتشو پیدا کنین. کاری نداره! بنویسین:

What is the difference between `isdigit()` and `isnumeric()`?⁷³

```
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 استرینگ که می‌خواد رو می‌گیره و با دومی عوض می‌کنه. اینجا `name` رو با «!!!» عوض کرد.

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 می‌گرده 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.

متدهای استرینگ رو می‌تونین از سایت «w3schools» یاد بگیرین.⁷⁴

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)

⁷³ 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>

⁷⁴ https://www.w3schools.com/python/python_ref_string.asp

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 می‌بریم 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 ادامش 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، 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 گفتم که ادامشه! پس اگر می‌خوانین نجسبه، یه فاصله بگذارین:

```
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، ادامشه.

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

روی هر کدوم 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.

• Nested while

درواقع یادتونه یه تابع نوشتیم که بیاد عدد اول حساب کنه و بعد به وسیله اون اومدیم مقسوم‌علیه‌های اول یه عدد رو حساب کردیم؟ درواقع هر بار که حلقه اجرا میشد، داشتیم یه تابع رو حساب می‌کردیم که خود اون تابع داخل خودش حلقه داشت. یعنی درواقع حلقه تودرتو بود. همونطور که if می‌تونست داخل خودش یه if دیگه داشته باشه، while هم می‌تونه داخل خودش while داشته باشه!

تمرین:

۱- دنباله‌ای از اعداد که با فاصله جدا شده‌اند، به شما داده می‌شود. این دنباله می‌تواند شامل اعداد مثبت، صفر و منفی باشد. (اما همگی اعداد صحیح اند) همچنین اعداد می‌توانند چند رقمی هم باشند. مثلاً:

```
1 -12 0 231 921 -229 0 212
```

از شما خواسته شده که در این دنباله، بزرگ‌ترین زیردنباله‌ای (از لحاظ تعداد اعداد) که می‌تونین پیدا کنین که همه اعداد مثبت باشند، رو بیابید.

input:

```
1 -12 0 231 921 -229 0 212
```

output:

```
2
```

بزرگ‌ترین زیردنباله‌ای که همه اعدادش مثبت هستن، زیر دنباله ۲۳۱ و ۹۲۱ هست که دو عدد هستن. پس ۲ باید چاپ شه.

input:

```
1 0 21 -1 12 91 82
```

output:

```
3
```

بزرگ‌ترین زیردنباله، زیردنباله ۱۲، ۹۱، ۸۲ هست که ۳ عدد هستن. پس ۳ باید چاپ شه.

Finding the Longest Word in a Sentence

```
sentence = "Python programming is interesting"
longest_word = ""
i = 0

while i < len(sentence):
    current_word = ""
    while i < len(sentence) and not sentence[i].isspace():
        current_word += sentence[i]
        i += 1
    if len(current_word) > len(longest_word):
        longest_word = current_word
    i += 1

print("Original Sentence:", sentence)
print("Longest Word:", longest_word)
```

۲- «Kaprekar's routine»:

برنامه ای بنویسید که یک عدد طبیعی n را از ورودی بگیرد. تضمین می شود n حداکثر ۹۹۹۹ است و همچنین حداقل دو رقم متمایز دارد. سپس الگوریتم زیر را روی n آن قدر اجرا کند تا n با ۶۱۷۴ برابر شود. سپس تعداد دفعاتی که الگوریتم اجرا شده تا به ۶۱۷۴ برسد را در خروجی چاپ کند:

- ارقام n را به صورت صعودی و نزولی مرتب کند تا دو عدد چهار رقمی حاصل شود (در صورت نیاز آن قدر سمت چپ عدد صفر می گذاریم تا چهار رقمی بشود)
- عدد کوچکتر را از عدد بزرگتر کم کند و حاصل را به عنوان n جدید ذخیره کند.
- مجدداً الگوریتم را به ازای n جدید اجرا کند.

مثال:

$n = 1495$

```
9541 - 1459 = 8082
8820 - 0288 = 8532
8532 - 2358 = 6174
```

مثال:

$n = 9888$

9888 - 8889 = 999
9990 - 0999 = 8991
9981 - 1899 = 8082
8820 - 0288 = 8532
8532 - 2358 = 6174

ورودی

شامل یک خط و نشان دهنده عدد مورد نظر.

خروجی

شامل یک خط و نشان دهنده تعداد اجرای الگوریتم.

input:

1495

output:

3

input:

9888

output:

پاسخ:
پاسخ ۱:
روش اول:

```
s = input()
n = len(s)
max_cnt = 0
cnt = 0

i = 0
while i < n:
    if s[i] > '0' and s[i] <= '9':
        while (i < n) and (s[i] >= '0' and s[i] <= '9'): # skip
other digits
            i += 1
            cnt += 1
        elif s[i] == '0': # zero means we have reached the end of a
positive sequence
            if cnt > max_cnt:
                max_cnt = cnt
            cnt = 0
        elif s[i] == ' ': # skip spaces
            pass
        else: # s[i] < '0' or s[i] > '9'
            i += 1 # skip the negative sign
            while (i < n) and (s[i] >= '0' and s[i] <= '9'):
                i += 1
            if cnt > max_cnt:
                max_cnt = cnt
            cnt = 0
        i += 1

if cnt > max_cnt:
    max_cnt = cnt
print(max_cnt)
```

پاسخ ۲:

```

n = int(input())
count = 0
while n != 6174:
    s = str(n)

    # ezafe kardan sefr be s ta 4 raghami shavad
    # (baraye voroodi haye kamtar az 4 ragham ham be 6174 miresim)
    s += '0' * (4 - len(s))

    # moratab kardan az bozorg be koochak:
    s_bozorg = ''

    while s != '':
        """
        find the maximum digit in s in each iteration and add
        it to s_bozorg and remove it from s and repeat this
        until s is empty which means s_bozorg is sorted from
        big to small"""
        maximum = '0'
        maximum_index = 0

        # maximum baadi ra be dast miavarim
        for i in range(len(s)):
            if s[i] > maximum:
                maximum = s[i]
                maximum_index = i

        s_bozorg += maximum

        # hazf kardan maximum az s:
        s = s[:maximum_index] + s[maximum_index+1:]

    # ezafe kardan sefr be s_bozorg ta 4 raghami shavad
    # نکته: agar x <= 0: 'reshte' * x == ''
    s_bozorg += '0' * (4 - len(s_bozorg))

    s_koochak = s_bozorg[::-1]

    n = int(s_bozorg) - int(s_koochak)

    count += 1

print(count)

```

۱- سؤال در لینک زیر:

<https://quera.org/problemset/298/>

۲- سؤال در لینک زیر:

<https://quera.org/problemset/1359/>

پاسفنامه:

پاسخ:

```
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}*'

        # 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 عددی بیان رو به ترتیب نوشتیم. `power` اول صفر هست و هر بار که بخش پذیره، یکی به توان اضافه میشه.

`is_a_factor` هم نشون میده که آیا اون `i` ما، فاکتوری از عدد هست یا نه؟ اگر نیست، هیچ وقت 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!

پاسخ ۲:

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 دلیلش 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 راحت که همه عناصر بودن. مثلاً ایندکس آخر رو در نظر بگیریم. میوفته توی while درونی و تا وقتی پیدا نشه، از توش بیرون نمیدا! این برای تمام ایندکسا 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

خب تا اینجا ما یه سری ساختار داشتیم که می‌تونستیم توشون چیزمیز نگهداری کنیم. مثلاً ساختار integer که میشد داخلش عدد صحیح قرار داد. ساختار float که میشد عدد اعشاری نگهداشت. ساختار string که میشد رشته نگهداشت. اگر متغیری از این ساختارها استفاده می‌کرد، صرفاً می‌تونست یه چیز رو داخل خودش نگهداره. نمی‌تونست ده‌تا چیز رو نگهداره. فقط یه مقدار رو می‌تونست در یه زمان نگهداره. نمی‌تونستیم بگیم:

```
age = 20 30
```

حالا شاید من بخوام ساختاری داشته باشم که بگم می‌خوام در اون چندتا عدد نگه‌دارم. مثلاً یه لیستی از سن‌های کاربران. اینو چیکار کنم؟ پایتون گفته غمت نباشه! من یه ساختاری برات تعبیه کردم اسمش «list». بیا ببین چجوریه:

```
mylist = [1, 2, 5, -1, -7, 200]
print(mylist)
```

میگه یه نوع خاص از داده هست که شامل چندتا چیزه. توی برکت می‌تونی عنصرهایی که می‌خوای رو بنویسی.

یه اشتباه رایج:

ما نباید یادمون بره که بین عناصر کاما بذاریم. اگر نذاریم، مشکلاتی پیش میاد. مثلاً:

```
name_list = ['Tom',
             'Lily',
             'Rose',
             'Sarah',
             'Jack']

print(len(name_list))
```

خروجی:

4

به نظر میاد ۵ تا استرینگ توشه. اما اگر طولش رو پرینت کنیم، میگه ۴. نگاه کد اگر بندازیم، می‌بینیم که یه کاما جلوی «Rose» یادمون رفته و عملاً پایتون فکر کرده که «Sarah»، ادامه «Rose» هست و چون استرینگ هستن هردو، هر دو می‌چسبونه به هم (concatenation انجام میشه!).^{۷۵}

فرض کنین می‌خواین لیست نمرات دانشجوها 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)
```

^{۷۵} این رو یه بار قبلاً می‌دونستم ولی از حافظم پاک شده بود. دوباره توی چنل «<https://www.youtube.com/@Indently>» دیدم، گفتم بیارم براتون. این چنل جالبه و چیزای جالب پایتون رو آورده. ویدیوش: <https://www.youtube.com/shorts/Zn4Hh5fY2II>

- حالا شاید بگین چطور می‌تونم به عناصرش دسترسی داشته باشم؟
+ پایتون می‌گه که برای سادگی کار، دسترسی به اعضا رو شبیه استرینگ در نظر بگی. یعنی هر کدوم از این عناصر، یه ایندکس دارن که می‌تونین با دادن شماره ایندکس بهش دسترسی داشته باشین. بدیهیه که ایندکس‌ها از ۰ تا ۱ - len هستن. (بله تابع len برای لیست هم کار می‌کنه!)

```
mylist = [1.28, 'John', -256]
print(mylist[0])
print(mylist[1])
print(mylist[2])
```

و خروجی:

```
1.28
John
-256
```

اینجا دقیقاً مثل ایندکسینگ استرینگ. همه چیزایی که قبلاً بلد بودیم اینجا هم به کار میاد. (تمام اون slicing و for i in range و for c in s و چیزای دیگه). مثلاً:

```
mylist = [1.28, 'John', -256]
print(mylist[0])
print(mylist[1])
print(mylist[2])
print(mylist[-1])
print(mylist[0:2])
```

و خروجی:

```
1.28
John
-256
-256
[1.28, 'John']
```

اینجا برخلاف استرینگ، می‌تونیم یه عنصر رو مستقیم عوض کنیم. یعنی:

```
l = ['Tom', 'Lily', 'Rose', 'Sarah', 'Jack']
l[1] = 'Lilei'
print(l)
```

و خروجی:

```
['Tom', 'Lilei', 'Rose', 'Sarah', 'Jack']
```

عنصر ایندکس ۱ عوض شده.
حتی این کارم میشه کرد:

```
l = ['Tom', 'Lily', 'Rose', 'Sarah', 'Jack']  
l[1] = 'Lilei'  
l[1:3] = ['Lilei', 'Lucy']  
print(l)
```

و خروجی:

```
['Tom', 'Lilei', 'Lucy', 'Sarah', 'Jack']
```

عنصر ایندکس ۱ تا قبل ۳ (یعنی ایندکس ۱ و ۲)، عوض شده.

حتی ما می‌تونیم یه چیزی از ورودی بگیریم و بذاریم توی لیست! مثلاً:

```
inp1 = input()  
inp2 = input()  
l = [inp1, inp2]  
print(l)
```

حالا این لیست به چه کارهایی میاد؟ فرض کنین من استاد یه دانشگاهم. می‌خوام اطلاعات دانشجویها رو ذخیره کنم. اینطوری ذخیره می‌کنم:

```
name_list = ['Tom', 'Lily', 'Rose', 'Sarah', 'Jack']  
last_name_list = ['Smith', 'Jackson', 'Johnson', 'Lee', 'Taylor']  
age_list = [24, 28, 32, 30, 29]  
std_num_list = [201801, 201802, 201803, 201804, 201805]
```

یعنی درواقع اسما رو توی یه لیست، فامیلا یه لیست دیگه. سن یه لیست. شماره دانشجویی یه لیست دیگه. و خب موقع نگه‌داری، حواسم بود که ایندکس صفر همه لیستا، اطلاعات یه نفره. ایندکس یک، اطلاعات نفر بعدی. ایندکس ۲ نفر بعدیش و....

حالا اگر بخوام اطلاعات همه رو چاپ کنم، اینطور عمل می‌کنم:
مثلاً:

```

name_list = ['Tom', 'Lily', 'Rose', 'Sarah', 'Jack']
last_name_list = ['Smith', 'Jackson', 'Johnson', 'Lee', 'Taylor']
age_list = [24, 28, 32, 30, 29]
std_num_list = [201801, 201802, 201803, 201804, 201805]

for i in range(len(name_list)):
    print(name_list[i], last_name_list[i], age_list[i],
          std_num_list[i])

```

و خروجی:

```

Tom Smith 24 201801
Lily Jackson 28 201802
Rose Johnson 32 201803
Sarah Lee 30 201804
Jack Taylor 29 201805

```

یعنی عملاً گفتم که یه `for` بزن به اندازه طول. بعدش در هر بار `for` ایندکس `i` ام رو برای لیست‌های مختلف چاپ کن.

- چرا صرفاً `len` برو برای `name_list` گرفتی؟

+ چون طول همه لیستا ثابتن. پس نیازی نبود طول بقیه رو چک کنم. اگر طول متغیر بود، بله نیاز بود چک کنم. چون ممکن بود طول یکی کمتر باشه و عملاً به ارور `IndexError: list index out of range` بر می‌خوردم.

گاهی که تعداد اعضا زیاده، برای خوشگلی و خوانایی، اینطوری هم می‌نویسین:

```

my_list = ['Bruce Schneier',
           'Edward Felten',
           'Ronald Rivest']

```

چیز خیلی عجیبی نیست! هر کاما که بذارین، می‌تونین یه اینتر بزنین. اینطوری خواناتر. فقط یادتون نره که کاما رو بگذارین. چون اگر نذارین، `concatenate` رخ میده و دوتا می‌چسبن به هم!

لیست زیر رو در نظر بگیرین:

```
l = ['james', 'jack', 'hannah', 'amir']
```

خب این لیست شامل ۴ عنصره. ایندکسش از ۰ تا ۳ هست. ما اگر مثلاً بگیم ایندکس صفر رو پرینت کن، برامون «james» رو پرینت می‌کنه.

حالا قبول دارین «james» خودش یه استرینگه؟ پس میشه روی ایندکس‌های همین `james` هم حرکت کرد و چیزی که می‌خوایم رو به دست بیاریم! مثلاً:

```
l = ['james', 'jack', 'hannah', 'amir']
print(l[0][1])
```

بهش می‌گیم برو توی ایندکس صفرم لیست، عنصر رو پیدا کن. بعدش ایندکس ۱ از اون عنصر رو بده. عملاً می‌ده:

```
a
```

ایندکس یک از james همون کرکتر «a» هست!
یا مثلاً:

```
l = ['james', 'jack', 'hannah', 'amir']
print(l[1][1:3])
```

خروجی:

```
ac
```

برو توی ایندکس ۱ لیست، ایندکس ۱ تا قبل ۳ اون عنصر رو چاپ کن.

خب شاید دلتون بخواد که به یه لیست، یه چیزی اضافه کنین. شاید اولین چیزی که به نظرتون بیاد آینه که خب مثل string، از تکنیک concatenation استفاده کنم:^{۷۶}

```
my_list = [1, 'Hello', 3.4, 'World', True, 'Python']
my_list += 3
print(my_list)
```

اما اگر اجراش کنیم، ارور می‌ده:

```
TypeError: 'int' object is not iterable
```

یکم به دلیلش فکر کنین!

+ درسته ما می‌تونستیم یه استرینگ رو با یه استرینگ دیگه جمع بزنیم. درسته ما می‌تونستیم یه عدد رو با یه عدد دیگه جمع بزنیم. اما یکم دقت کنین! اینجا شما دارین یه عدد رو با یه لیست جمع می‌کنین! این امکان‌پذیر نیست!

بله اگر یه لیست دیگه داشتیم، می‌تونستیم با یه لیست دیگه جمع بزنیم. ولی جمع بین دو چیز مختلف، امکان‌پذیر نیست! اینجا هم باید ۳ رو لیستی کنیم و اضافه کنیم به لیست اصلی:

```
my_list = [1, 'Hello', 3.4, 'World', True, 'Python']
my_list += [3]
print(my_list)
```

^{۷۶} شاید دقت کردین که این مثال، True هم درون لیسته. بله میتونه باشه! چرا نتونه! True هم یه چیزه دیگه!

اجراش کنیم:

```
[1, 'Hello', 3.4, 'World', True, 'Python', 3]
```

حالا درست شد. چون علامت جمع بین دو چیز یکسان کار می‌کند. هر دو لیستن، پس کار می‌کند. یا اینطوری:

```
my_list = [1, 'Hello', 3.4, 'World', True, 'Python']
value = 3
my_list += [value]
print(my_list)
```

ایندفعه یه متغیر رو اضافه کردم به لیست. خروجیش:

```
[1, 'Hello', 3.4, 'World', True, 'Python', 3]
```

شاید بعضیا ذهنشون بره سمت `type casting`. یعنی بگن که همونطور که من قبل استرینگ رو به اینتیجر و بالعکس و... تبدیل می‌کردم، الآن هم نگاه کردم دیدم پایتون تبدیل به لیست هم داره. پس اول بیایم ۳ رو لیست کنیم و بعد جمع بزنیم:

```
my_list = [1, 'Hello', 3.4, 'World', True, 'Python']
my_list += list(3)
print(my_list)
```

اما اجراش کنیم می‌بینیم به ارور می‌خوریم:

```
TypeError: 'int' object is not iterable
```

دلیلشم آینه که بله درسته پایتون تبدیل به لیست داره، اما این تبدیل به لیستش، چیزی که می‌گیره، یه چیز «iterable» هست.

- عه! «iterable» یعنی چی؟

+ خیلی ساده بگم یعنی اینکه هرچیزی که شما می‌تونین با مکانیزم `for` یعنی:

```
for x in my_variable:
```

پایاده‌سازیش کنین. یعنی هر چیزی که بتونین روش حرکت کنین. چیا میشد؟ استرینگ، لیست. یعنی این `casting`ی که `list` داره، صرفاً می‌تونه استرینگ، لیست، `file`، `tuple`، `dictionary` تو پرانتز بگیره.^{۷۷} مثلاً یه استرینگ بدیم:

^{۷۷} چند مورد آخری رو فعلاً بلد نیستین. نگرانش نباشین. فعلاً فقط `string` و `list` رو مدنظر قرار بدین.

```
s = 'abcde'
print(list(s))
```

اجراش کنیم:

```
['a', 'b', 'c', 'd', 'e']
```

یعنی میاد دونه دونه کرکتر رو جدا می کنه و به عنصر جدای لیست بهشون اختصاص میده.

حالا شاید خسته شدین از این همه دردسر! خب بابا به چیزی بده بهم که من اگر خواستم به عنصر به لیست اضافه کنم بتونم!

پایتون میگه همونطور که من برای string، به سری متد داشتم، برای لیستم دارم. (متدها چون مخصوص خود اون تایپ هستن، با به نقطه بعد اون متغیر نمایش داده میشن). مثلاً متد Append. با موس روش گرفتم 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 به عنصر رو به انتهای 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 هیچی یا همون «None» رو ریترن می کنه. 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
```

اجراش کنیم:

```
[1.28, 'John', 'a']  
None
```

اول میاد کرکتر «a» رو اضافه می‌کنه به لیست. بعد لیست رو چاپ می‌کنه. همونطور که انتظار می‌رفت، «a» به ته لیست اضافه شده.

حالا دستور بعدی یه پرینت هست. پرینت همیشه چی رو چاپ می‌کرد؟ مقداری که بر می‌گرده. اینجا هم درسته توی پس‌زمینه اون چیزی که می‌خواستیم، اضافه شده اما پس از اضافه‌شدن مقدار «None» یعنی هیچی برگشته. پس همون «None» چاپ میشه.

بیایم خودمون همین متد رو به وسیله یه تابع خودنوشته پیاده‌سازی کنیم:

```
def append_to_list(mylist, value):  
    mylist.append(value)  
    return mylist  
  
mylist = [1.28, 'John']  
value = 'a'  
mylist = append_to_list(mylist, value)  
print(mylist)
```

این ساده‌ترین حالتش که با کمک همون متد `append` بود. اومدم مقدار ریترن‌شده تابع رو گذاشتم توی متغیر اصلی. خروجیش:

```
[1.28, 'John', 'a']
```

حالت دوم:

```
def append_to_list(mylist, value):  
    mylist += [value]  
    return mylist  
  
mylist = [1.28, 'John']  
value = 'a'  
mylist = append_to_list(mylist, value)  
print(mylist)
```

اول `value` رو تبدیل به `list` کردم و بعدش اضافه‌اش کردم.

- **Split()**

فرض کنیم ما می‌خوایم یه استرینگ به شکل زیر داریم:

```
s = 'Sarah Micheal Smith Robert'
```

شاید ما بخوایم اسما رو بریزیم توی لیست. یه چیزی شبیه این:

```
l = ['Sarah', 'Micheal', 'Smith', 'Robert']
```

- راه چیه؟

+ پایتون گفته که بیا یه راه بهت بدم. یه متد داریم به نام «split». این متد برای **string** هاست! و با نقطه بعد استرینگ میشه صداش زد. این متد میاد یه **string** میگیره. بر حسب چیزی که تو میگی، جدا می‌کنه و می‌ریزه توی یه لیست:

```
s = 'Sarah Micheal Smith Robert'
l = s.split(' ')
print(l)
```

همونطور که می‌بینی، بهش گفتم که جدا کن بر حسب فاصله:

```
['Sarah', 'Micheal', 'Smith', 'Robert']
```

Code 1:

```
s = '1,2,3,4,5'
l = s.split(',')
print(l)
```

همونطور که می‌بینی، بهش گفتم که جدا کن بر حسب کاما:

```
['1', '2', '3', '4', '5']
```

Code 2:

```
s = '1, 2, 3, 4, 5'
l = s.split(', ')
print(l)
```

همونطور که می‌بینی، بهش گفتم که جدا کن بر حسب کاما:

```
['1', ' 2', ' 3', ' 4', ' 5']
```


همونطور که می بینیم بر حسب کاما جدا کرده و فاصله ها هم موندن. چون گفتیم بر حسب کاما جدا کن. هر جا کاما دید، یه عنصر جدید می سازه.

Code 3:

```
s = 'Sarah Micheal Smith Robert'
l = s.split(' ')
print(l)
```

یادتونه گفتیم که پایتون همیشه می خوام کار ما رو ساده کنه؟ پایتون میگه خیلی وقتا آدمای می خوان بر حسب فاصله یه سری چیز رو جدا کنن. پس اگر بهش توی پرانتز `split` چیزی ندی، به صورت پیشفرض فرض می کنم منظورتون که به وسیله فاصله جدا کنم. خسته بودی حوصلت نمیشه هر دفعه بنویسی بر حسب فاصله:

```
['Sarah', 'Micheal', 'Smith', 'Robert']
```

Code 4:

```
s = 'hihellohi'
l = s.split('hello')
print(l)
```

خروجی:

```
['hi', 'hi']
```

بر حسب کلمه «hello» جدا کرده.

نکته: می تونیم از چیزایی مثل «\n» و این ها استفاده کنیم. یعنی مثلاً وقتی یه استرینگ چندخطی داریم و می خواین چیزهای مختلفشو جدا کنیم.

از این متد می تونیم برای گرفتن چندتا عدد که به وسیله یه فاصله توی یه خط جدا شدن، بهره بگیریم. ولی حواستون باشه که عددا به صورت `string` توی یه لیست قرار می گیرن!

متدهای لیست رو می تونیم از سایت زیر بخونیم. همش مثال داره:

https://www.w3schools.com/python/python_lists_methods.asp

- `join()`

فرض کنیم شما یه لیست از `string` ها (و نه اعداد صحیح و...) داریم. می خواین تمام استرینگ ها رو با جداسازی که می خواین، بریزین توی یه دونه استرینگ. مثلاً لیست زیر رو داریم:

```
l = ['hello', 'world', 'hi']
```

می‌خواهیم به همچنین string ای داشته باشیم:

```
s = 'hello$world$hi'
```

یعنی تمامشون رو به هم وصل کنیم با به فاصله مثلاً. این رو با متد «join» انجام میدیم. این متد به لیست شامل صرفاً استرینگ می‌گیره و با چیزی که می‌خواهیم، اعضا رو به هم وصل می‌کنه. مثلاً:

```
l = ['hello', 'world', 'hi']  
s = '$'.join(l)  
print(s)
```

بهش گفتم که این لیستو بگیر، با علامت «\$»، اعضا رو به هم وصلشون کن:

```
hello$world$hi
```

یا حتی می‌تونیم بگیریم که:

```
l = ['hello', 'world', 'hi']  
s = '-'.join(l)  
print(s)
```

خروجی:

```
hello--world--hi
```

یا حتی می‌تونیم بگیریم با فاصله وصل کن:

```
l = ['hello', 'world', 'hi']  
s = ' '.join(l)  
print(s)
```

خروجی:

```
hello world hi
```

یا حتی می‌تونیم بگیریم با استرینگ خالی (هیچی! فاصله یا هیچی ندار)، وصل کن:

```
l = ['hello', 'world', 'hi']  
s = ''.join(l)  
print(s)
```

خروجی:

```
helloworldhi
```

فقط حواستون باشه که به متد «join»، یه لیست استرینگی باید بدین! عددی ندین!

- **copy()**

شاید از دیدن متد `copy` توی لیست، تعجب کنین. بگین خب من اگر بخوام یه کاپی از مثلاً `l1` داشته باشم، به راحتی می‌تونم مثل متغیر عادی، اینطوری بنویسم:

```
l1 = [1, 2, 3, 4]
l1_copy = l1
```

دیگه چه نیازی که یکساعت برم اینطوری بنویسم:

```
l1_copy = l1.copy()
```

اما لابد یه دلیلی وجود داشته که اومدن متدی به نام `copy` ساختن. بیکار نبودن که! بیاین دلیلشو نشونتون بدم:

فرض کنین ما می‌خوایم یه استرینگی به شکل زیر داریم:

```
l1 = [1, 2, 3, 4]
l1_copy = l1
l1_copy[0] = 'a'
print(f'l1_copy: {l1_copy}')
print(f'l1: {l1}')
```

اومدم یه لیست تعریف کردم. یه کاپی گرفتم ازش. توی کاپی ایندکس صفرم رو تغییر دادم. بعد هم کاپی و هم عادی رو چاپ کردم:

```
l1_copy: ['a', 2, 3, 4]
l1: ['a', 2, 3, 4]
```

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^{۷۸} 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:

۷۸ مربوط به مفاهیم `reference` و ایناس.

```
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 = [1, 2, 3, 4]
l1_copy = l1[:]
```

بازم انگار کاپی کردیم. یعنی گفتیم دونه دونه ایندکسا رو بریز توی `l1_copy`.

• Remove() → remove an element

حذف کردن یه عنصر از یه لیست با متد `remove` انجام میشه. یعنی یه مقدار می‌گیره و عنصر رو پاک می‌کنه:

```
l1 = ['a', 'b', 'c', 'd', 'e', 'f']
l1.remove('b')
print(l1)
```

مثلاً گفتیم که `b` رو پاک کن. خروجیش:

```
['a', 'c', 'd', 'e', 'f']
```

نکته! اگر مقداری رو بدیم که وجود نداشته باشه، ارور میده:

```
l1 = ['a', 'b', 'c', 'd', 'e', 'f']
l1.remove('z')
print(l1)
```

خروجیش:

```
ValueError: list.remove(x): x not in list
```

همونطور که می‌بینین گفته یه `x` ای رو گفتی حذف کنم ولی داخل لیست نیست! پس می‌تونین قبلش اول ببینین اصلاً هست توش یا نه. اینو با یه `if` ساده میشه هندل کرد:

```
l1 = ['a', 'b', 'c', 'd', 'e', 'f']
if 'z' in l1:
    l1.remove('z')
print(l1)
```

خروجیش:

```
['a', 'b', 'c', 'd', 'e', 'f']
```

و اروری نداد.

فقط حواستون باشه که اگر چندتا از اون عنصر وجود داشته باشه، فقط اولی پاک میشه. پس می‌تونین با یه `while`، اینو هندل کنین که تمام عنصرایی که مقدارشون اونه پاک شن:

```
l1 = ['a', 'b', 'c', 'a', 'e', 'f', 'a']
while 'a' in l1:
    l1.remove('a')
print(l1)
```

خروجیش:

```
['b', 'c', 'e', 'f']
```

- `pop()`

شاید ما بخوایم یه عنصر در یک ایندکس خاص رو حذف کنیم. یعنی شاید مقدارشو ندونیم، ولی می‌دونیم در کدوم ایندکسه. این با متد `pop` انجام میشه:

```
l1 = ['a', 'b', 'c', 'a', 'e', 'f', 'a']
l1.pop(2)
print(l1)
```

بهش گفتیم عنصر موجود در ایندکس ۲ رو پاک کن. خروجیش:

```
['a', 'b', 'a', 'e', 'f', 'a']
```

مطمئنأ اینجا هم باید حواستون باشه که یه وقت ایندکس اشتباهی ندین. این رو می‌تونین باز با یه `if` هندل کنین:

```
l1 = ['a', 'b', 'c', 'a', 'e', 'f', 'a']
i = 100
if i < len(l1) and i >= -len(l1):
    l1.pop(i)
print(l1)
```

بهش می‌گیم اگر ایندکس کوچکتر از `len` بود و همچنین رنج ایندکس‌های منفی که از `-۱` شروع میشه و تا منفی طول ادامه پیدا می‌کنه، بود، حالا حذفش کن:

```
['a', 'b', 'c', 'a', 'e', 'f', 'a']
```

تذکره مهم!

خواستون باشه که وقتی روی 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, 3]
for i in range(len(l)):
    if l[i] == 3:
        l.remove(l[i])
print(l)
```

این باعث ارور میشه. چون شما داری روی یه چیز حرکت می‌کنی و یهو وسط کار عوضش می‌کنی:

```
IndexError: pop index out of range
```

این مواقع روی یه لیست حرکت کنین و روی یکی دیگه که کاپیش هست تغییرات اعمال کنین:

```
l = [1, 2, 3, 3]
l_copy = l.copy()
for i in range(len(l)):
    if l[i] == 3:
        l_copy.remove(l[i])
print(l_copy)
```

این باعث ارور میشه. چون شما داری روی یه چیز حرکت می‌کنی و یهو وسط کار عوضش می‌کنی:

```
[1, 2]
```

مثال:

به شما یه سری نمره به صورت اعشاری داده میشه. تا وقتی که عدد ۱- وارد نشده، هی نمره می‌گیرین و به لیست اضافه می‌کنین. 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: "))

summ = 0
for i in range(len(score_list)):
    summ += score_list[i]

if len(score_list) != 0:
    print(summ / len(score_list))

```

هی می گیرم و هی به لیست اضافه می کنم. تا کجا؟ تا وقتی که برابر منفی ۱ نشده! (اگر اینو متوجه نمیشین، برگردین مثلاً و تمرینای while رو بخونین. اینو دقیقاً اونجا توضیح دادم).

خارج while با for میام جمع رو حساب می کنم و در آخر هم میانگین می گیرم. البته که حواسم هست که مثل اون سؤال while که قبلاً حل کردیم، ممکنه در ابتدا به من -۱ داده باشه و عملاً باعث شده باشه که هیچی درون لیست نباشه و len اش برابر صفر شه و ارور تقسیم بر صفر بخورم. پس قبلش گفتم اگر طول صفر نبود نیاز به یه چیز چاپ کنی. وگرنه که چیزی نیاز نیست!

اما بیاین یکم کد رو بهینه تر کنیم. کجا داریم اضافه کاری می کنیم؟ قسمت محاسبه len. سه بار 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: "))

summ = 0
length = len(score_list)
for i in range(length):
    summ += score_list[i]

if length != 0:
    print(summ / length)

```

برنامه نویس 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 نمی زنیم. 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 می‌کند. 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 گذاشتن جلوی `while` و همیشه تکرار میشه و برنامه‌تون اونجا گیر می‌کنه.

برای اینکه برنامه‌تون 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: "))

summ = 0
length = len(score_list)
for i in range(length):
    summ += score_list[i]

print(summ / length)
```

یعنی `while` بینهایت. هر بار ولی چک می‌کنم که اگر ورودی برابر -۱ بود، `break` کنه و از حلقه بیرون 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: "))

summ = sum(score_list)
length = len(score_list)
print(summ / length)

```

یادتونه قبلاً می‌گفتیم که اسم متغیر رو `sum` تعریف نکنین؟ اون موقع می‌گفتیم. اون موقع می‌گفتیم چون رنگش شبیه پرینت و اینا میشه و نشون میده که مال پایتونه و ما نباید دستش بزنین. الآن دقیق‌تر می‌تونیم بفهمیمش. دلیلش آینه که `sum` یه تابع درونی پایتونه و ما نباید اسمش مشابه اون انتخاب کنیم.

مثال:

یه سری عدد به شکل

1 -1 2 4 -3

بهتون میدن و شما باید بریزیدش توی لیست و هر دفعه جمع دوتا کناری‌ها 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` جداسازی 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؟^{۷۹}

حالا میایم به 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 به عنصر داره، ایندکس صفرم رو پرینت می‌کنیم که جوابه. خب تستش کنیم ببینیم چی خروجی میده:

input:

1 -1 2 4 -3

output:

1-1-12-1224-1224244-3

^{۷۹} چون خط بعدیش ما داریم عنصر $i + 1$ حساب می‌کنیم و خب نمی‌خوایم $i + 1$ به len برسه. پس i نباید به $len - 1$ برسه.

- عه چرا اينطور شد؟! انگار به هم چسبیدن. انگار concatenate شدن كه!
+ آره دقيقاً! يعنى انگار عددا 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 ذهنم سمت concatenate كردن در string رفت.

تمرین:

مى توانيد از متدهاى join (تبدیل لیست به استرینگ) و متد split (تبدیل استرینگ به لیست) و متد index (یافتن محل و ایندکس قرارگیری) استفاده كنيد.
۱- كد مورس، روشى برای انتقال اطلاعات مى باشد.

International Morse Code

1. The length of a dot is one unit.
2. A dash is three units.
3. The space between parts of the same letter is one unit.
4. The space between letters is three units.
5. The space between words is seven units.

A	• —	U	• • • —
B	— • • •	V	• • • — •
C	— • — •	W	— • — •
D	— • •	X	— • • — •
E	•	Y	— • — • — •
F	• • — •	Z	— — • •
G	— — •		
H	• • • •		
I	• •		
J	• — — —	1	• — — — —
K	— • —	2	• • — — —
L	• — • •	3	• • • — —
M	— —	4	• • • • —
N	— •	5	• • • • •
O	— — —	6	— • • • •
P	• — — •	7	— — • • •
Q	— — • —	8	— — — • •
R	• — •	9	— — — — •
S	• • •	0	— — — — —
T	—		

اینطوریه که میان به جای هر حرف، یه سری نقطه یا خط میذارن که پیام کد بشه. یعنی به جای اینکه ما حرف a رو بفرستیم، میایم «.-» رو می‌فرستیم. مطمئناً برای اینکه نقطه خط‌های مختلف با هم قاطی نشن، بین هر کلمه یه سری فاصله و درون هر حرف هم یه سری فاصله داده میشه که نقطه و خط‌ها با هم قاطی نشن.

مثال:

..... . -... -... --- .-- --- .-. -... -..

این کد مورس که نشون‌دهنده «helloworld» هست، درون هر حرف فاصله‌ای نداره ولی بین حرفا، ۳ تا فاصله داره.

مثال:

• • • • • - • • • - • • - - - • - - - - - • - • • - • •
- • •

این کد مورس که نشون‌دهنده «helloworld» هست، درون هر حرف ۱ فاصله و بین حرفا، ۲ تا فاصله داره.

به شما یه پیام که حاوی کد مورس داده میشه. و ازتون میخوایم که به پیام عادی (حروف انگلیسی) تبدیلش کنین.

ورودی:

سه خط.

خط اول تعداد فاصله بین کلمات (حتما بزرگتر در درون هر حرف)

خط دوم تعداد فاصله درون هر حرف (می تونه صفر هم باشه!)

پیام مورسی

خروجی:

متن کدگشایی شده

input:

```
3
0
.... . .-.. .-. --- .- - - .- .-.. -..
```

output:

```
helloworld
```

input:

```
2
1
. . . . . - . . . - . . - - - . - - - - . - . - . .
- . .
```

output:

```
helloworld
```

input:

```
3
2
. - - - . .
```

output:

az

توجه! استفاده از ۲۶ تا if و elif و یا تعریف یک متغیر برای هر حرف انگلیسی (یعنی ۲۶ تا متغیر تعریف کردن)، مجاز نیست! باید از ساختاری بهتری کمک بگیرید.

پاسخ نامه:

پاسخ ۱:

```

def morse_to_msg(between_words, between_letters, morse_msg,
                 morse_code_list, alphabet):
    l = morse_msg.split(between_words * ' ')
    msg = ''
    for word in l:
        if between_letters == 0: # if there is no space between
letters
            letter = word
        else:
            letters_list = word.split(between_letters * ' ')
            letter = ''.join(letters_list)

            # implement msg += alphabet[morse_code_list.index(letter)]
without .index()
            for i in range(len(morse_code_list)):
                if morse_code_list[i] == letter:
                    msg += alphabet[i]
                    break

    return msg

morse_code_list = ['.-', '...-', '-.-.', '-..', '.', '...-', '--.',
'....',
                    '...', '----', '-.-', '....', '--', '-.', '---',
'---',
                    '----', '...', '...', '-', '...-', '...-', '---',
'---',
                    '-.-', '----']
alphabet = 'abcdefghijklmnopqrstuvwxyz'
between_words = int(input())
between_letters = int(input())
morse_msg = input()
print(morse_to_msg(between_words, between_letters, morse_msg,
                   morse_code_list, alphabet))

```

از یه لیست و یه استرینگ استفاده کردم. یه لیست شامل کد مورس و استرینگ با ایندکس متناظر، شامل حروف الفبا. که بگم وقتی یه حرف از کد مورس رو جدا کردی، match اون (ایندکس متناظر با اون) توی استرینگ، همون حروف هست.

همینجا که هستیم بدونیم که ضرب یه لیست در یه عدد، باعث میشه که اعضاهاش همون تعداد بار تکرار شن. (مثلاً استرینگ بودا! اینجا هم همونطوره!):

```

s = 'a'
print(s * 3)
l = [1, 2, 3]
print(l * 3)

```

یا مثلاً می‌تونیم از این تکنیک استفاده کنیم که یه لیست شامل ۱۰ تا صفر داشته باشیم:

```
l = [0]
l = l * 10
print(l)
```

خروجی:

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

این تکنیک ضرب بیشتر وقتایی به کار میاد که بخوایم توی یه مسأله، یه لیست رو n بار گسترش و تکرار (extend) بدیم توی خودش.

• 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 یه سری `i` درون `mylist` باشن که `i` ها در رنج ۱ تا قبل ۱۱ هستن.
مثال:


```

# Code 1
l = [i % 2 for i in range(10)]
print(l)

# Code 2
l = [i * 2 for i in range(10)]
print(l)

# Code 3
l = [0 for i in range(10)]
print(l)

# Code 4
l = ['hi' for i in range(10)]
print(l)

# Code 5
l1 = ['1', '2', '3', '4']
l2 = [element for element in l1]
print(l2)

```

توضیح:

کد ۱: می‌گه یه سری i رو باقی‌مونده بگیر بر ۲، بریز توی لیست. i هات از رنج ۰ تا قبل ۱۰ (خود عدد ۹) باشن. (مطمئنأً چون حاصل باقی‌مونده بر ۲، یا صفر هست یا یک، یه لیست شامل صفر و یک خواهیم داشت.)

کد ۲: می‌گه یه سری $i * 2$ بذار داخل لیست. که این i ها در رنج ۱۰ هستن.

کد ۳: یه سری صفر بذار برای i هایی که در رنج ۱۰ هستن. بله! من از خود i هیچ استفاده‌ای نکردم. نیاز نداشتم. تنها استفادم، برای شمردن تعداد صفر گذاشتن بود. حالا بگین چندتا صفر چاپ میشه؟^{۸۰}

کد ۴: مثل ۳. ایندفعه گفتم استرینگ «hi» بذار.

کد ۵: می‌گه یه سری `element` بذار توی لیستم، برای `element` هایی که در لیست اولی قرار دارن. عملاً انگار دارم کاپی می‌کنم. خروجی‌ها به ترتیب:

```

[0, 1, 0, 1, 0, 1, 0, 1, 0, 1]
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
['1', '2', '3', '4']

```

یادتونه که موقع `split` کردن، `string` ریخته میشد توی لیست؟ خب فرض کنین بخوایم تک‌تک تبدیلیش کنیم به `Integer`. به نظرتون باید چیکار کنیم؟

^{۸۰} به اندازه بارهای اجرای `for` که ۱۰ تاست. (`for` از صفر تا خود ۹ می‌رفت که میشه ۱۰ بار.)

باید یه for بزنینم:

```
num_list = ['1', '2', '3', '4']
for i in range(len(num_list)):
    num_list[i] = int(num_list[i])
print(num_list)
```

حتماً یادتون هست که میشه یه عنصر لیست رو عوض کرد. مثلاً بگیم ایندکس فلانش بشه فلان. (که حتماً یادتون هست که در استرینگ این ممکن نبود!)

اما خب میشه این رو ساده‌تر 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 جمع دو عدد کناری رو می‌تونستیم 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])
```

اول اسپلیت کردم ریختم تو لیست. ولی از اونجایی که استرینگی ریخته‌شدن تو لیست، int شون کردم.

دیگه عادى جمع مى‌زنم.

خلاصه از این به بعد این مراحل رو حتماً بلد باشین:

```
# '1 2 3 4'
s = input()

# ['1', '2', '3', '4']
num_str_list = s.split()

# [1, 2, 3, 4]
num_int_list = [int(num) for num in num_str_list]

# ['1', '2', '3', '4']
num_str_list = [str(num) for num in num_int_list]

# '1 2 3 4'
s = ' '.join(num_str_list)
```

به ترتیب:

- ۱- یه استرینگ ورودی گرفتیم.
- ۲- تبدیل به لیستش کردم.
- ۳- اعضای لیست چون string بودن، تبدیل به integer کردم.
- ۴- یه لیست استرینگی ساختم.
- ۵- دوباره به هم وصلشون کردم که بتونم راحت با یه استرینگ چاپشون کنم.

حتى مى‌تونیم این کار هم کنیم که همونجا که اسرینگ گرفتیم، split اش هم کنیم. همونجا:

```
# ['1', '2', '3', '4']
num_str_list = input().split()

# [1, 2, 3, 4]
num_int_list = [int(num) for num in num_str_list]

# ['1', '2', '3', '4']
num_str_list = [str(num) for num in num_int_list]

# '1 2 3 4'
s = ' '.join(num_str_list)
```

اینطوری ترجمه میشه که input که داری ورودی می‌گیری، همونجا که گرفتی، نذار توی متغیر. بده به split، بعدش خروجی split رو بذار توی متغیر.

تمرین!

۱- ما متد upper رو برای استرینگ داریم. متدیه که یه استرینگ می‌گیره و همه کرکترهاشو تبدیل به حروف بزرگ می‌کنه و استرینگ جدید رو برمی‌گردونه (return می‌کنه).
در یک خط به شما یه سری کرکتری که با فاصله جدا شدن داده میشه و کار شما آینه که با استفاده از list comprehension، یه لیست شامل کرکترهای بزرگ بسازین و پرینتش کنین. مثلاً:

input:

```
a b c d
```

output:

```
['A', 'B', 'C', 'D']
```

یه لیست شامل حروف بزرگ رو چاپ کرد.

پاسخنامه:

پاسخ ۱:

```
s = input()
l = s.split()
l = [char.upper() for char in l]
print(l)
```

توضیح: قرار شد وقتی که یه سری چیز با فاصله میدن و می‌خوایم توی یه لیست ذخیره کنیم، اول با یه استرینگ بگیریم و بعد از متد split استفاده کنیم که بریزیم توی یه لیست. در قدم بعد می‌خوام حروف رو بذرنم. می‌گم که کرکترها رو upper کن و بذار توی لیست. خود کرکتر از کجا بیان؟ خود کرکتر، اونایی هستن که توی l1 وجود دارن.

نه تنها list comprehension ساده‌تره و کوتاه‌تره، بلکه سریع‌ترم 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 می‌کنه. بعدش میره بالا، یه دونه به j 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 $j = 6$ زشد، از for داخلی می‌پره بیرون و ادامه رو طی می‌کنه. می‌رسه به خطی که چندتا خط‌چین پرینت می‌کنه. بعدش میره بالا و i میشه ۱. دوباره میاد توی for دومی. j از ۰ تا ۲ پیش میره. این روند ادامه پیدا می‌کنه. با دیباگر IDE یا Text editor تون تستش کنین قشنگ‌تر 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 کنیم تا کامل درکش کنیم.

• 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 ببینیم 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 بگیم، می‌گیم 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.

تمرین:

۱- ماتریس زیر رو در خروجی چاپ کنیم:

```
1 2 3  
4 5 6  
7 8 9
```

۲- فرض کنیم لیست زیر رو داریم:

```
l = [1, 2, 0, 4 -1]
```

سعی کنین خروجی زیر رو تولید کنین:

```
1
2 1
0 2 1
4 0 2 1
-1 4 0 2 1
```

۳- فرض کنین لیست زیر رو دارین:

```
l = [1, 2, 0, 4 -1]
```

سعی کنین خروجی زیر رو تولید کنین:

```
1 2 0 4 -1
2 0 4 -1
0 4 -1
4 -1
-1
```

پاسخنامه:

پاسخ ۱:

```
for i in range(1, 10):
    print(i, end=' ')
    if i % 3 == 0:
        print()
```

- عه چرا nested for نرفتی؟ بهش می خورد nested for باشهها!
+ هر گردی گردو نیست! قبل حل مسأله فکر کنین که چه راهی مناسب تره!

پاسخ ۲:

قبول دارین انگار روی عناصر حرکت کردم و در هر مرحله که به یه عنصر دسترسی دارم، اومدم تا عقب و تا ایندکس صفر عنصرها رو پرینت کردم؟ پس یه for می خوام برای حرکت روی عناصر و یه حلقه دیگه هم می خوام که از اون عنصر پیام تا عقب:

```
l = [1, 2, 0, 4, -1]
for i in range(len(l)):
    j = i
    while j >= 0:
        print(l[j], end=' ')
        j -= 1
    print()
```

پرینت خالی، خودش یه اینتر میزنه. برای اینکه سطرها جدا شن، اینتر خالی زد.

پاسخ ۳:

این برعکس قبلیه. قبول دارین انگار روی عناصر حرکت کردم و در هر مرحله که به یه عنصر دسترسی دارم، رفتم تا انتها و تا $len - 1$ رو پرینت کردم؟ پس یه `for` می‌خوام برای حرکت روی عناصر و یه حلقه دیگه هم می‌خوام که از اون عنصر برم تا انتها:

```
l = [1, 2, 0, 4, -1]
for i in range(len(l)):
    j = i
    while j < len(l):
        print(l[j], end=' ')
        j += 1
    print()
```

هر دفعه با `while` میرم تا انتهای اون.

این دو تمرین خیلی مهم بودن. اینکه چطور من بتونم روی عناصر حرکت کنم و برم تا انتها و چطور بتونم حرکت کنم و بیام تا ابتدا، بسیار بسیار مهمن.

خب حالا برای کاربرد `nested for`، بیاین مثال زیر رو حل کنیم:

• Bubble Sort ($\theta(n^2)$)^{۸۱}

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 $i = 0$, $i = 1$ → swap(4, 5)

4 5 3 2 1 → Compare $i = 1$, $i = 2$ → swap(5, 3)

4 3 5 2 1 → Compare $i = 2$, $i = 3$ → swap(5, 2)

۸۱ تایم کامپلکسیتی n^2 هست.

4 3 2 5 1 → Compare $i = 3$, $i = 4$ → swap(5, 1)

Result: 4 3 2 1 5

خب بار اول انجام شد. به نظرتون چه اتفاقی افتاد؟
+ عدد ماکزیموم رفت آخر.
آره دقیقاً درسته!

Second time:

4 3 2 1 5 → Compare $i = 0$, $i = 1$ → swap(4, 3)

3 4 2 1 5 → Compare $i = 1$, $i = 2$ → swap(4, 2)

3 2 4 1 5 → Compare $i = 2$, $i = 3$ → swap(4, 1)

Result: 3 2 1 4 5

آیا نیازه ۴ و ۵ رو چک کنیم؟ نه! چون گفتیم عدد ماکزیموم رفته آخر. پس درواقع هر بار یه دونه چک کردنمون کمتر میشه. (هر بار یه دونه کمتر میریم تا ته) درواقع الکی نیاز به چک کردن نیست! چون مطمئنیم که نیاز به جابه جایی نیست.

Third time:

3 2 1 4 5 → Compare $i = 0$, $i = 1$ → swap(3, 2)

2 3 1 4 5 → Compare $i = 1$, $i = 2$ → swap(3, 1)

Result: 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 می تونیم 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 $i = 0$, $i = 1$ → swap(2, 1)

Result: 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 نیاز به یه for داریم که دونه دونه بریم جلو و نیاز به یه for دیگه داریم که از اون ایندکس به بعد رو مقایسه و swap کنیم. بیایم for رو بنویسیم تا بهتر درکش کنیم:

```
def sort_list(l): # Bubble sort
    length = len(l)
    for i in range(length):
        for j in range():
            if l[j] > l[j + 1]:
                # swap
    return l
```

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 + ۱.

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(l): # Bubble sort
    length = len(l)
    for i in range(length):
        for j in range(length - i - 1):
            if l[j] > l[j + 1]: # Swap
                temp = l[j]
                l[j] = l[j + 1]
                l[j + 1] = temp
    return l
```

```
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 باشن.
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 هستن.
جالب بود نه؟! پس کد پاسخ سؤال رو تمیزتر بنویسیم:

```
def sort_list(l): # Bubble sort
    length = len(l)
    for i in range(length):
        for j in range(length - i - 1):
            if l[j] > l[j + 1]: # Swap
                l[j], l[j + 1] = l[j + 1], l[j]
    return l

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 ($\Omega(n), O(n^2)$)⁸²

یکی از راه‌های دیگه مرتب‌سازی آینه که فرض می‌کنیم این لیست ما تا عنصر اول مرتب‌شده. بعدش ما انگار قراره عنصرای جدید بهش اضافه کنیم. بعد خب قاعدتاً باید عنصر جدید که به راست اضافه شده، با سمت چپ‌ها مقایسه کنیم. یعنی هی بیایم عقب و سمت چپ و تا جایی که نیاز جابه‌جاش کنیم. بعدش هی میریم جلو و عنصر بعدی. برای عنصر بعدی هم این کار رو انجام میدیم.
با ویدیو بهتر درکش می‌شه کرد.⁸³

input:

5 4 3 2 1

⁸² 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.

⁸³ Recommended: <https://www.youtube.com/watch?v=JU767SDMDvA>

فرض یه لیست شامل یه عنصر داریم. اون عنصر ۵ هست. چون لیست شامل یک عنصره، به صورت پیش فرض می گیم مرتب شده.

حالا انگار ۴ بهش اضافه شده. پس باید ۴ رو با ۵ مقایسه کنیم و اگر نیاز بود جاشو عوض کنیم:

First time:

5 4 3 2 1 → swap(5, 4)

4 5 3 2 1

حالا انگار یه لیست به طول ۲ داریم که مرتب هست. همون فرض insertion sort. یعنی تا یه جایی مرتبه، حالا عنصر جدید (راستیا رو) می خوایم اضافه کنیم. خب ۳ رو می خوایم اضافه کنیم. قاعدتاً ۳ باید تا وقتی که کوچکتر از چیپا هست بیاد سمت راست:

Second time:

4 5 3 2 1 → swap(5, 3)

4 3 5 2 1 → swap(4, 3)

3 4 5 2 1

پس ابتدا با ۵ مقایسه شده. بعد swap میشه. بعد با ۴. بازم نیاز هست swap شه. حالا ۳ اومده سمت چپ.

اینطوری حالا انگار یه لیست شامل ۳ عنصر داریم که مرتبه. حالا قراره ۲ بهش اضافه شه:

Third time:

3 4 5 2 1 → swap(5, 2)

3 4 2 5 1 → swap(4, 2)

3 2 4 5 1 → swap(3, 2)

2 3 4 5 1

باز ۲ تا وقتی که کوچکتر از چیپش هست، میاد سمت راست. ایندفعه انگار یه لیست تولید شد که شامل ۴ عنصر هست و مرتبه. حالا قراره ۱ رو بهش اضافه کنیم:

Fourth time:

2 3 4 5 1 → swap(5, 1)

2 3 4 1 5 → swap(4, 1)

2 3 1 4 5 → swap(3, 1)

2 1 3 4 5 → swap(2, 1)

1 2 3 4 5

عملاً یه لیست شامل ۵ عنصر داریم که مرتبه. هیچ عنصر دیگه‌ای هم نمونه. پس لیست مرتب شد!

Fifth time:

1 2 3 4 5

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 قاعدتاً نیازه به `for` بزنیم و دونه‌دونه بریم جلو. هر بار که رفتیم جلو و به عنصر جدیدی رسیدیم، باید تا ابتدا بیایم عقب و تا جایی که نیازه `swap` کنیم. پس این نیاز به یه حلقه درون یه حلقه دیگه‌ای داره:

```
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
```

شروع برگشتن به عقب باید از کجا باشه؟ هر بار از اون عنصری که هستیم. پس برای همین `z` رو برابر `i` قرار دادیم و با `z` برگشتیم به عقب.

اما یه چیزی ممکنه درست نباشه! `z` هی داره کم میشه و یه جایی صفر میشه عملاً میشه ایندکس منفی! الکی داره میچرخه! به خاطر `z - 1` توی `while`، عملاً وقتی `z = 1` بشه، ایندکس صفر توی `[z - 1]` بررسی میشه و اوکیه!

پس یه شرط دیگه اضافه می‌کنیم که تا وقتی که `z` بزرگ‌تر از صفره نیازه توی `while` بمونه:

```
def insertion_sort(l):
    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. ولی اگر دوم می‌گذاشتیم، ابتدا سمت چپ رو نگاه می‌کرد و می‌گفت که `[j]` با این ایندکس ندارم که!

حالا پایتون ایندکس منفی داریم و ارور نمی‌خوریم ولی خیلی از زبونا نداریم. اونجا به مشکل می‌خوریم.

• Selection Sort⁸⁴

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 کوچکترین رو میاریم اول.

84 More info: <https://www.geeksforgeeks.org/selection-sort/>

مثال (در هر مرحله مینیمومی که از مرحله قبل پیدا شده، با رنگ آبی هایلایت شده). min_i هم ایندکس عنصر مینیموم هست:

Input:

12 15 7 9 2

First time ($i = 0$):

12 15 7 9 2 → Compare 12 and 15 → 12 is less → $\text{min_i} = 0$

12 15 7 9 2 → Compare 12 and 7 → 7 is less → $\text{min_i} = 2$

12 15 7 9 2 → Compare 7 and 9 → 7 is less → $\text{min_i} = 2$

12 15 7 9 2 → Compare 7 and 2 → 2 is less → $\text{min_i} = 4$

12 15 7 9 2 → Reach to the end. $\text{Swap}(l[0], l[\text{min_i}])$

2 15 7 9 12

Second time ($i = 1$):

2 15 7 9 12 → Compare 15 with 7 → 7 is less → $\text{min_i} = 2$

2 15 7 9 12 → Compare 7 with 9 → 7 is less → $\text{min_i} = 2$

2 15 7 9 12 → Compare 7 with 12 → 7 is less → $\text{min_i} = 2$

2 15 7 9 12 → Reach to the end. $\text{Swap}(l[1], l[\text{min_i}])$

2 7 15 9 12

Third time ($i = 2$):

2 7 15 9 12 → Compare 15 with 9 → 9 is less → $\text{min_i} = 3$

2 7 15 9 12 → Compare 15 with 9 → 9 is less → $\text{min_i} = 3$

2 7 15 9 12 → Reach to the end. $\text{Swap}(l[2], l[\text{min_i}])$

2 7 9 15 12

Fourth time ($i = 3$):

2 7 9 15 12 → Compare 15 with 12 → 12 is less → $\text{min_i} = 4$

2 7 9 15 12 → Reach to the end. $\text{Swap}(l[3], l[\text{min_i}])$

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. (اینجا دیگه نیاز به دونقطه برای if نیست!)

تمرین! همش با list comprehension انجام بده!

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]]

```

۵- 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 میشه توش استفاده کرد! آره نگفته بودم ولی می خواستم کم کم روی پای خودتون بایستین و با امتحان و خطا syntax (نحوه نوشتار) رو پیدا کنین. مثلاً می تونستین `else` رو هم همون ته بذارین. می دیدین که به ارور می خورین. میارین اول می بینین عه درست شد! میگه i رو بذار توی لیست، اگر 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 = [i  
    if i % 2 == 0  
    else 0  
    for i in range(1, 10)]
```

-۴

همیشه سعی کنین مسأله های سخت رو بشکونین به مسئله های کوچکتر. کاری که من اینجا می کنم اینه که از دید سطح بالا شروع می کنم و عناصر اصلی رو می سازم و بعدش جزئیات رو کامل می کنم. یهو نمی پرم توی جزئیات!

خب قبول داریم 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 به for می‌زنیم و حاصل به 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 به 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 به تابع ببینین. میره اول نگاه می‌کنه که آیا حداقل به 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 چه‌جور 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 «r» توی لیست درونی دسترسی داشته باشیم:

```
mylist = [1.28, 'John', -256, ['car', 'bus', 152], False]
```

```
print(mylist[3][0][2])
```

توجه! مطمئناً شما نمیتونین بگین ایندکس مثلاً منفی یک یه عدد رو بده! همیشه:

```
mylist = [1.28, 'John', -256, ['car', 'bus', 152], False]
print(mylist[3][-1][-1])
```

خروجی:

```
TypeError: 'int' object is not subscriptable
```

عددها (مثل اینجا که ۱۵۲ رو گفتم)، ایندکس ندارن. فعلاً فقط لیست‌ها و استرینگ‌ها ایندکس دارن.

- این به چه درد می‌خوره؟

+ فرض کنین می‌خوایم یه ماتریس بسازیم:

```
mylist = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

یا همونطور که قبلاً گفتیم، می‌تونیم بعد کاما، یه اینتر بزنین که تمیزتر شه:

```
mylist = [[1, 2, 3],
           [4, 5, 6],
           [7, 8, 9]]
```

حالا تمیزتر شد و دقیقاً شبیه ماتریس شد.

حواستون باشه که این لیست شامل ۳ عنصره. هر کدوم از اون عناصر هم خودشون یه لیستن.

```
mylist = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
print(len(mylist))
print(len(mylist[0]))
```

بار اول گفتم که طول خود لیستو بده. بعدش گفتم طول لیست ایندکس صفر رو بده.

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 به `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 هست که داره میاد، همونو `split` کن بریز توی `row1`. در آخر هم توی ماتریس قرارش دادیم.

خب نظرتون چیه یه ماتریس طور 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 لیست 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!^{۸۵}

^{۸۵} 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.

حتی می‌تونین اسم هم بدین بهش. بگین وقتی اضافه‌اش کردی، اسمش مثلاً بذار `deepc`. من هر وقت بخوام صدایش 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)
```

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 توی 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.^{۸۶} 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 (مثلث فیباچ):

<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]))
```

86 Import Statement Overhead → <https://wiki.python.org/moin/PythonSpeed/PerformanceTips>,
<https://moinmo.in/GPL>

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 می کنیم.

تمرین ۳:

دو دنباله از اعداد به شما داده میشه. دنباله اول sort شده هست. شما باید تک تک اعداد دنباله دوم رو دونه دونه به دنباله اول اضافه کنید که هنوزم دنباله اول sort باقی بمونه:

input:

1 2 4

3 2

output:

1 2 2 3 4

input:

7 11

12 0 -1 2

output:

-1 0 2 7 11 12

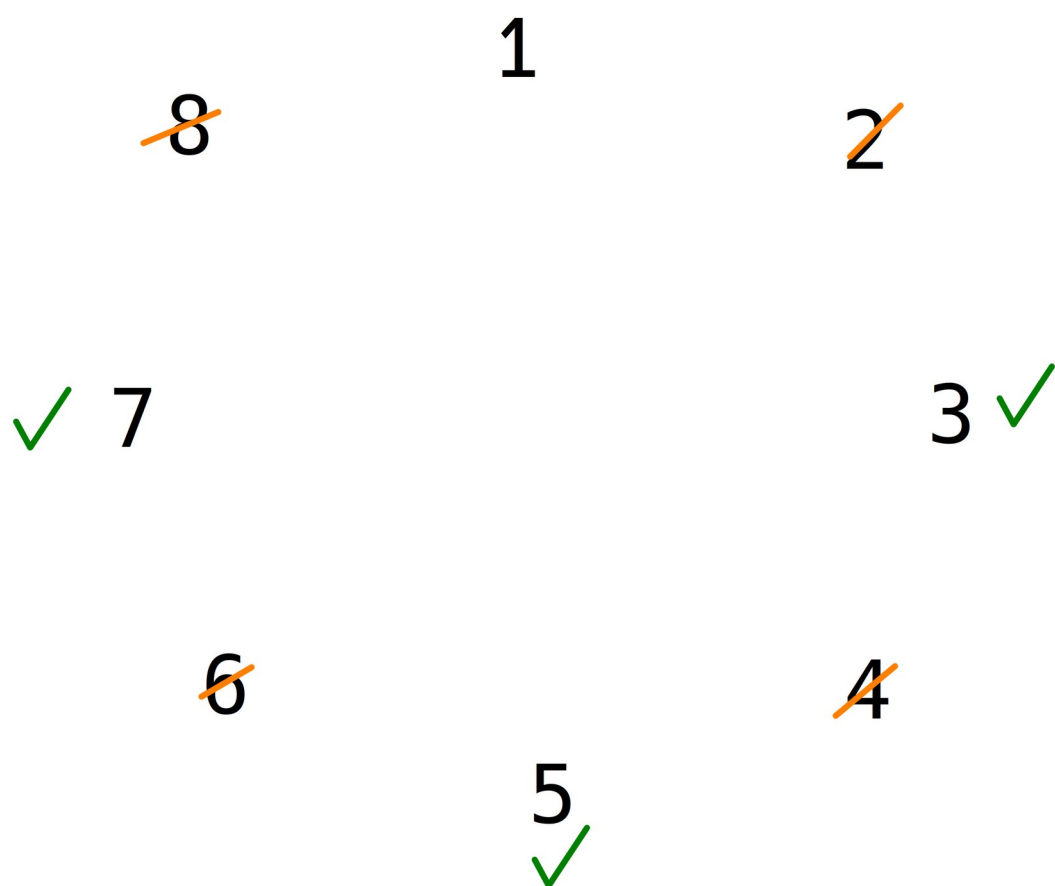
راهنمایی: سعی کنین یه تابع بنویسین که مسئول اضافه کردن یه عنصر به لیست باشه و بعدش با کمک اون تابع، تک تک عناصر جدید رو دونه دونه اضافه کنین به لیست.

تمرین ۴ (انتقابات دور میز):^{۸۷}

<https://quera.org/problemset/604/>

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.

87 Josephus Problem: https://en.wikipedia.org/wiki/Josephus_problem



بارو دوم: دفعه قبل ۸ خط خورد. پس از ۸ شروع می‌کنیم و افراد باقی‌مونده رو یکی در میون خط می‌زنیم. ۱ نیاز نیست خط بخوره. پس ۳ خط می‌خوره. (دوبار خطش زدم که مشخص باشه بار دوم خط خورده و افراد کوررنگ بهتر بتونن تشخیص بدن)
بعد ۳، ۷ خط می‌خوره. (۵ خط نمی‌خوره چون قرار بود یکی در میون از افراد باقی‌مونده خط بزنیم)

✓✓
1

~~8~~

~~2~~

~~7~~

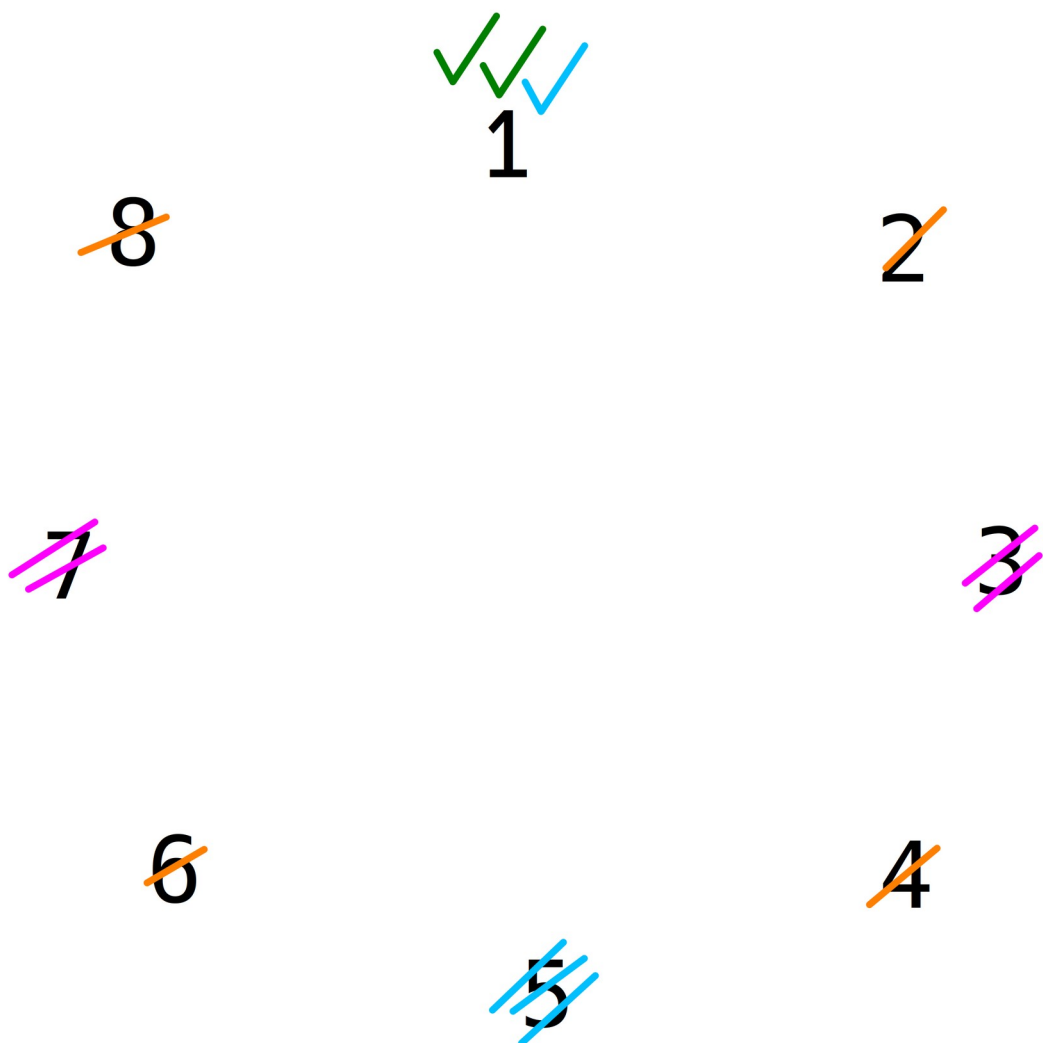
~~3~~

~~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 باقی‌مونده ۱ بود. پس ۱ باید چاپ شه.

تمرین ۵ (هلزون ماتریسی):

<https://quera.org/problemset/609>

توجه ۱: صفر هم عددی مربع کامل محسوب می‌شود.

توجه ۲: همیشه حواستون باشه که شرط پایان رو شرطی درست که مطمئنید صد درصد درسته انتخاب کنید. اینجا ممکنه یه نفر موقع گرفتن ورودی‌ها، جمعشون رو حساب کنه و مثلاً بگه جمع اعداد ۲۵ هست. حالا وقتی که داره به صورت حلزونی روی عناصر حرکت می‌کنه، می‌گه شرط پایان `while` مو (شرط پایان حرکتمو)، رسیدن به جمع ۲۵ می‌دارم. این شاید در نگاه اول درست باشه، اما ممکنه یه سری مشکلات به وجود بیاد. فرض کنید که ورودی اینطوری باشه:

```
2
10 10
0 5
```

مجموع چنده؟ ۲۵. خب اگر شرط پایان رو ۲۵ بذاریم، به پنج ردیف دوم که برسه، می‌گه خب مجموع شد ۲۵ و من خارج میشم. اما نمی‌گه که ممکنه یه سری صفر هم بعدش باشن که باعث نشدن مجموع تغییر کنه. $25 = 0 + 25$. پس بازم توان ۲ داریم. پس تعداد جمع توان ۲ ها تا این مرحله، ۲ تاست. پاسخ میشه ۲.

توجه ۳: وقتی می‌گیم اعداد در فلان رنج هستن، یعنی ممکنه مثلاً همش هم ۱ باشه:

```
2
1 1
1 1
```

یا از هر عددی، چندبار تکرار شه. (مثلاً چندتا صفر داشته باشیم).

تمرین ۷ (بمب بازی):

<https://quera.org/problemset/3407>

راهنمایی: یکی از راهها این هست که وقتی می‌خوانین یه بمب جدید اضافه کنین، تمام خونه‌های اطرافشو (در صورت اینکه بمب نیستن)، آپدیت کنین (یکی به عددشون اضافه کنین). خونه‌های اطراف اینا هستن:

- چپ بالا
- بالا
- راست بالا
- راست
- راست پایین
- پایین
- چپ پایین
- چپ

راهنمایی ۲: درواقع ابتدا بر اساس دو عدد اول، یک ماتریس خالی بسازین. مثلاً نمونه یک که گفته

```
4 3
```

یعنی تعداد سطرها چهارتا و ستون‌ها سه تا:

```
l = [[0, 0, 0],
      [0, 0, 0],
      [0, 0, 0],
      [0, 0, 0]]
```

بعدش بر اساس مختصات بمب‌ها، به اطراف بمب‌ها (در صورتی که خودشون بمب نیستن) یه دونه عدد اضافه کنین.

پاسفنامه:

پاسخ:

می‌دونیم ضرب ماتریس اینطوری که اول عنصرهای row1 با عناصر col1 نظیر به نظیر ضرب میشن.

و

```
res[0][0]
```

رو می‌سازن.

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 نیازه اولین for من، روی سطرهای row1 باشه. یه تابع می‌نویسم که ضرب رو انجام بده. چیا نیازه 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 کنیم 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 نوشتیم 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 جمعی که گفتیم رو هر مرحله انجام میدی 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` ششم 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 دارین 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) 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]

            rowl.append(product)

        res.append(rowl)

    return res
```

یعنی درواقع for سوم ما هر بار یه بار ضربا رو انجام میدی و جمع می‌کنی و یه عدد می‌سازه. For دوم ما میاد هر بار یکی از این عددا 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 ساخته‌شده، 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):  
        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)  
  
        res.append(rowl)  
    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  
    rowl = input().split()  
    rowl = [int(i) for i in rowl]  
    mat1.append(rowl)
```

```
for i in range(row2): # Get matrix 2  
    rowl = input().split()  
    rowl = [int(i) for i in rowl]  
    mat2.append(rowl)
```

```
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 نیازه 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 به `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)، به فاصله با `end` خالی چاپ کنه. یعنی بعد فاصله اینتر نزنه. (اگر `end` نمی‌گذاشتم، بعد هر پرینت مثل همیشه خودش به اینتر می‌زد) درواقع 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` برنیم. در پایان هر باری که `for` داخلی اجرا میشه، به ردیف چاپ میشه. پس نیاز به 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 دومیه 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:

خب بهتره اول یه تابع بنویسیم که بیاد یه عنصر بگیره و اضافه‌اش کنه به لیست. بعدش دیگه تابع رو که داشته باشیم، می‌تونیم روی لیست ورودی حرکت کنیم و دونه‌دونه که داریم حرکت می‌کنیم، با استفاده از تابع، اضافه‌اش کنیم به لیست اصلی.

خب برای اضافه کردن یه عنصر، باید دونه دونه روی اعضای لیست حرکت کنیم و بگیم خب کجا از عنصر لیست اول کوچکتر یا مساویه که اضافه‌اش کنیم به قبل عنصر لیست. یعنی درواقع لیست جدید، از ترکیب عنصرای قبلی + عنصر جدید + ادامه لیست تشکیل میشه:

```
def insert_sort(l, num):
    for i in range(len(l)):
        if num <= l[i]:
            l = l[:i] + [num] + l[i:]
            break
    return l
```

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 بابتش نیست که چکش کنه!) پس کدمون رو درست کنیم:

```
def insert_sort(l, num):
    for i in range(len(l)):
        if num <= l[i]:
            l = l[:i] + [num] + l[i:]
            break

    # if the num is bigger than all the numbers in l
    elif i == len(l)-1:
        l = l + [num]
        break

    return l

mainl = input().split()
mainl = [int(num) for num in mainl]
inputl = input().split()
inputl = [int(num) for num in inputl]
for num in inputl:
    mainl = insert_sort(mainl, num)
mainl = [str(num) for num in mainl]
res = ' '.join(mainl)
print(res)
```

برای بیرون تابع هم اول یه لیست گرفتیم. بعدش اعضاشو integer کردم. بعدش لیست دوم گرفتیم. بازم اعدادش رو صحیح کردم. بعدش یه for زدم روی لیست دومی و دونه‌دونه به لیست اولی اضافه‌اش کردم. بعدش یه لیست دارم که شامل اعداد صحیحه و مرتبم هست. می‌خوام که این لیستو تبدیل به string ای کنم که اعداد با فاصله از هم جدا شدن که چاپش کنم. از متد «join» کمک می‌گیرم. بعدش هم نتیجه رو چاپ می‌کنم.

البته میشد توی تابع، یه متغیر به نام طول تعریف کرد و طول لیست رو بریزیم داخلش که نخوایم دو بار طول رو حساب کنیم:

```
def insert_sort(l, num):
    n = len(l)
    for i in range(n):
        if num <= l[i]:
            l = l[:i] + [num] + l[i:]
            break

    # if the num is bigger than all the numbers in l
    elif i == n-1:
        l = l + [num]
        break

    return l

mainl = input().split()
mainl = [int(num) for num in mainl]
inputl = input().split()
inputl = [int(num) for num in inputl]
for num in inputl:
    mainl = insert_sort(mainl, num)
mainl = [str(num) for num in mainl]
res = ' '.join(mainl)
print(res)
```

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!

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 داریم 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 می خوره، برابر $len - 1$ و برای فردا، $len - 2$ هست:

1 2 3 4 5 6 7 → last one $i = 5 \rightarrow i = len - 2$

1 2 3 4 5 6 7 8 → last one $i = 7 \rightarrow i = len - 1$

حالا پس برای اینکه بدونم آخری چی بوده، اونو توی یه متغیر ذخیره می کنم:

```
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 داریم هی باید خط بزنیم تا فقط یک عنصر باقی بمونه؟ یا

درواقع صرفاً یک عنصر که «0» نیست باقی بمونه. همه عناصر لیست ما بشن «0» به جز یکیش. یعنی

اینو اینطور می تونیم 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 بگیم دوتا دوتا برو جلو و خط بزن! چون ممکنه دوتا بره جلو و

اونم خط خورده باشه.

پس فکر درست تر اینه که بگیم 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 نیاز به 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_found = False
        while first_found == False or l[i] == '0':
            if l[i] != '0':
                first_found = True
            i += 1
```

یعنی میگم اول از آخری که خط خورد یه ایندکس برو جلو. بعد یه فلگ به نام first_found 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 می پریم 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 هیچ وقت 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 کنیم. 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 نیاز به تعداد آدم‌ها رو بگیریم و همچنین یه لیست شامل عدد اون آدم‌ها بسازیم. در آخر هم نیاز به خروجی‌مون رو چاپ کنیم. چون تابع یه لیست به 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(l, people_count)[0])

```

تمام!

ببینین 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، یه 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);
}

```

پاسخ ۵:

ایده کلی آینه که پیام بگم ۴ حالت داریم. حالت راست، پایین، چپ، بالا. اینا به ترتیب تکرار میشن. هر بار که به سمت راست رفت، یه سطر رو کامل می‌کنه. پس سطر شروع یکی میره پایین. برای اینکه کنترل کنم که حلزون ما از بالا از کدوم ردیف شروع شه، از سمت راست به کدوم ستون نرسه، از پایین به کدوم ردیف نرسه، از چپ از کدوم ستون شروع شه، نیاز به چند متغیر داشتم. هر بار که میاد پایین، یه ستون کامل می‌کنه. ستون سمت چپو. پس ستون پایان یکی میاد سمت چپ.

```

def is_square(n):
    i = 0
    while i * i <= n:
        if i * i == n:
            return True
        i += 1
    return False

def snail_walk(mat, n):
    """
    @param mat: matrix of size n * n
    @param n: size of matrix

```

@return: how many squares appeared in the cumulative sum of the snail walk

increasing i means going down
increasing j means going right

go right. one row completed. start_row += 1
go down. one col completed. end_col -= 1
go left. one row completed. end_row -= 1
go up. one col completed. start_col += 1
"""

```
i = j = summ = cnt = 0
start_row = start_col = 0
end_row = end_col = n - 1
total_blocks = n * n
dir = 'r' # r, d, l, u
while total_blocks > 0:
    if dir == 'r':
        while j <= end_col: # go right until end_col
            summ += mat[i][j]
            if is_square(summ):
                cnt += 1
            j += 1
            total_blocks -= 1
        dir = 'd'
        i += 1 # after going right, go down
        j -= 1 # j is out of range after while (j > end_col)
        start_row += 1
```

```
    elif dir == 'd':
        while i <= end_row: # go down until end_row
            summ += mat[i][j]
            if is_square(summ):
                cnt += 1
            i += 1
            total_blocks -= 1
        dir = 'l'
        j -= 1 # after going down, go left
        i -= 1 # because i is out of range (i > end_row)
        end_col -= 1
```

```
    elif dir == 'l':
```

```

        while j >= start_col: # go left until start_col
            summ += mat[i][j]
            if is_square(summ):
                cnt += 1
            j -= 1
            total_blocks -= 1
            dir = 'u'
            i -= 1 # after going left, go up
            j += 1 # because j is out of range (j < start_col)
            end_row -= 1

    elif dir == 'u':
        while i >= start_row: # go up until start_row
            summ += mat[i][j]
            if is_square(summ):
                cnt += 1
            i -= 1
            total_blocks -= 1

    dir = 'r'
    j += 1 # after going up, go right
    i += 1 # because i is out of range (i < start_row)
    start_col += 1
    return cnt

n = int(input())
# create n * n matrix with input
matrix = []
for i in range(n):
    line_input = input().split()
    line_input = [int(x) for x in line_input]
    matrix.append(line_input)

print(snail_walk(matrix, n))

```

پاسخ ۷:

```

def mine_sweeper(matrix, row, col, no_of_bombs):
    for i in range(no_of_bombs):
        bomb = input().split()
        bomb_row = int(bomb[0]) - 1
        bomb_col = int(bomb[1]) - 1
        matrix[bomb_row][bomb_col] = '*'
        # update the cells around the bomb:
        # up:
        if bomb_row != 0:
            if matrix[bomb_row-1][bomb_col] != '*':
                matrix[bomb_row-1][bomb_col] += 1
        # down:
        if bomb_row != row-1:
            if matrix[bomb_row+1][bomb_col] != '*':
                matrix[bomb_row+1][bomb_col] += 1
        # left:
        if bomb_col != 0:
            if matrix[bomb_row][bomb_col-1] != '*':
                matrix[bomb_row][bomb_col-1] += 1
        # right:
        if bomb_col != col-1:
            if matrix[bomb_row][bomb_col+1] != '*':
                matrix[bomb_row][bomb_col+1] += 1
        # up-left:
        if bomb_row != 0 and bomb_col != 0:
            if matrix[bomb_row-1][bomb_col-1] != '*':
                matrix[bomb_row-1][bomb_col-1] += 1
        # up-right:
        if bomb_row != 0 and bomb_col != col-1:
            if matrix[bomb_row-1][bomb_col+1] != '*':
                matrix[bomb_row-1][bomb_col+1] += 1
        # down-left:
        if bomb_row != row-1 and bomb_col != 0:
            if matrix[bomb_row+1][bomb_col-1] != '*':
                matrix[bomb_row+1][bomb_col-1] += 1
        # down-right:
        if bomb_row != row-1 and bomb_col != col-1:
            if matrix[bomb_row+1][bomb_col+1] != '*':
                matrix[bomb_row+1][bomb_col+1] += 1
    return matrix

```



```

row_col = input().split()
row = int(row_col[0])
col = int(row_col[1])

# create all zero matrix (because we have no bombs yet)
matrix = []
for i in range(row):
    col_list = []
    for j in range(col):
        col_list.append(0)
    matrix.append(col_list)
# Create matrix with list comprehension:
# matrix = [[0 for i in range(col)] for j in range(row)]

no_of_bombs = int(input())
matrix = mine_sweeper(matrix, row, col, no_of_bombs)

for row in matrix:
    row = [str(i) for i in row]
    print(' '.join(row))

```

میشه گفت که لیست یکی از 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.^{۸۸} یعنی:

```

s = 'hello'
l = ['h', 'e', 'l', 'l', 'o']

```

پس خیلی مهمه که به خوبی یادش بگیرین!

• 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.

^{۸۸} در اصل بهشون میگن Array (آرایه)

مثال ::

```
def is_upper(c):  
    return 'A' <= c <= 'Z'  
  
def is_lower(c):  
    return 'a' <= c <= 'z'  
  
def is_letter(c):  
    return is_upper(c) or is_lower(c)  
  
print(is_letter('a'))
```

من بهش گفتم که برو is_letter رو برای «a» صدا بزن. میره توی تابع می‌بینه که عه! مگه میشه! یه تابع دیگه رو ریترن کنم؟ میگه باشه بذار ببینم که حاصل تابع is_upper چیه. می‌بینه False هست. پس جاش می‌ذاره False:

```
def is_upper(c):  
    return 'A' <= c <= 'Z'  
  
def is_lower(c):  
    return 'a' <= c <= 'z'  
  
def is_letter(c):  
    return False or is_lower(c)  
  
print(is_letter('A'))
```

بعدش می‌گه خب برم حاصل is_lower رو حساب کنم. حساب می‌کنه میشه True. پس جاش True می‌گذاره:

```
def is_upper(c):  
    return 'A' <= c <= 'Z'  
  
def is_lower(c):  
    return 'a' <= c <= 'z'  
  
def is_letter(c):  
    return False or True  
  
print(is_letter('A'))
```

حالا حاصل or بین True و False که میشه True رو بر می گردونه و True چاپ میشه.

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 به ازای ۵ چاپ کنه. خب میره سراغ تابع f. توی تابع f نوشته که حاصل یه تابع رو ریترن کن! - عه؟! مگه میشه؟! میشه بگیریم که حاصل یه تابع رو ریترن کن؟! + آره میشه! اینجا هم همین گفته. پس میره اول حاصل تابع increment رو برای ۵ حساب کنه. میره اونجا. اونجا نوشته که $num + 1$ رو ریترن کن. پس $5 + 1$ یعنی ۶ رو ریترن می کنه. حالا برمی گرده توی تابع f و حاصل تابع increment رو جاش قرار میده:

```
def f(num):  
    return 6
```

الآن ۶ رو ریترن می کنه. پس ۶ به سمت پرینت پاس داده میشه و پرینت هم برامون ۶ 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(2)` 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 ۲ هست. پس میرم توی تابع `square(2)` که ببینه حاصل چیه. یعنی:

```
def square(2): return 2 * 2
def f1(2): return square(2) * 2
def f2(2): return f1(2) + 1

print(f2(2))
```

اونجا میبینم که باید عددی که بهش پاس داده شده رو ضربدر خودش کنه و ریترن کنه. پس ۲ رو ضربدر دو میکنه. میشه چهار. حالا ۴ رو ریترن می‌کنه. 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 square(2): return 4
def f1(2): return square(2) * 2 # return 4 * 2 → return 8
def f2(2): return f1(2) + 1 # return 8 + 1 → return 9

print(f2(2))
```

اول توی `square`، مقدار ۴ رو پس میده. برمی‌گرده جای قبلیش. جای قبلیش توی `f1` بود، ۴ رو جایگذاری می‌کنه. ۴ ضربدر ۲ رو پس میده. برمی‌گرده جای قبلیش. جای قبلیش توی `f2` بود، ۸ رو جایگذاری می‌کنه.

حالا گفته شده $۸ + ۱$ رو پس بده. میگه باشه. ۹ رو پس می‌ده و پرینت ۹ رو چاپ می‌کنه وقتی که کد به تابعی بر خورد کنه، نمیتونه که تابع رو ریترن کنه! پس اول باید حاصل اون تابع رو حساب کنه و یه مقدار به دست بیاره. حالا که مقدار به دست آورد، می‌تونه جایگذاری کنه و ریترن کنه. اصطلاحاً هی میره تو تا وقتی که به یه مقدار (و نه یه تابع) برسه. بعدش بازگشت میزنه هی. حالا که فهمیدین چه جوریه، میریم سراغ توابع بازگشتی (:

به طور کلی، هر تابعی که خودش توای 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.

قبول دارین که ۲ به توان ۳ یعنی ۳ بار ۲ رو در خودش ضرب کن؟
پس یه راه دیگش اینکه من بگم:

```
def power(a, b):  
    return a * power(a, b-1)
```

مگه قرار نبود تابع توان رو حساب کنه، خب وقتی اومدی توای تابع، a رو ضربدر خود تابع ولی ایندفعه با b یه دونه کمتر کن.

- چرا یه دونه کمتر؟

+ چون که یه ضرب رو انجام دادیم. گفتیم عدد ضربدر عدد به توان یکی کمتر. خب این تا ابد ادامه پیدا می‌کنه. هر بار a رو ضربدر تابع a به توان یکی کمتر می‌کنه.

ما باید یه شرطی بگذاریم که این متوقف شه. یعنی بگیم دیگه بسه. نمیخواد باز تابع رو صدا بزنین.

- به نظرتون شرط پایان توان‌رسوندن چیه؟ (راهنمایی: b چه مقداری پیدا کنه، کار ما تموم شده؟)

+ قبول دارین که اگر b صفر شه، یعنی کامل به توان رسیده. چون هر بار داریم a رو ضربدر a^{b-1} می‌کنیم. پس وقتی b صفر شه، یعنی به تعداد b تابع صدا زده شده و b بار a در خودش ضرب شده. پس قبول دارین در مرحله آخر، توان‌رسوندن a کامل شده؟ پس باید چی برگردونه که ضربدر a^b شه و مقدارش همون a^b شه؟ قاعدتاً یک:

```
def power(a, b):  
    if b == 0:  
        return 1  
    return a * power(a, b-1)
```

بریم روی یه مثال توضیح بدیم. فرض کنین می‌خوام ۲ به توان ۵ رو حساب کنم. مثلاً:

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:

25:**

`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 یکیه به دست بیاریم:

`power(2, 3) → return 2 * power(2, 2)`

عه این هم همینطوره! پس بریم باز تا وقتی به مقدار نرسیدیم، تابع 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 مسئله‌ها رو به شدت ساده حل می‌کنه. حالا می‌رسیم بهش. فعلاً باید یکم تمرین کنیم که بهتر بتونین درکش کنین.

اصطلاحاً به اون $b == 0$ می‌گن «base case» یا «حالت پایه». یعنی کجا باید متوقف شه و یه مقدار (و نه یه تابع) رو برگردونه؟

همیشه سعی کنین خود منطق تابع رو بنویسین و بعدش برید سراغ نوشتن base case. مثل همین سؤال توان که اول منطق رو نوشتیم و در آخر base case رو نوشتیم.

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 از هردو یکی کم کنین و هروقت a برابر صفر و b نشد، 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)
```

آیا می‌تونم یه متن رو ریترن کنم؟ بله میشه! هرچیزی رو میشه ریترن کرد. چه عدد چه استرینگ چه لیست و ...

اول چک‌ها رو انجام میدم و در آخر تابع رو دوباره صدا می‌زنم با $a-1$ و $b-1$. درواقع 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. درواقع هنر اصلی شما، یافتن متوقف‌کننده‌هاست. اگر درست انتخاب نکنین، ممکنه بی‌نهایت بار تکرار شه! مثل loop بی‌نهایت.

نکته! همیشه سعی کنین که ریترن آخری رو بنویسین و سپس base case ها رو بنویسین.

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. موقوف‌کننده 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)^{۸۹} رفت و رنگ آبی (رو به بالا) برگشت رو نشون میده. (اول کد مسیر هایلات‌های 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.

^{۸۹} جهت هم مشخص کردم که اگر یکی سیاه سفید پرینت گرفته یا کوررنگ هست، بهتر بتونه تشخیص بده (:

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 نخوایم 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 ریترن شده (یعنی None) چاپ نمیشه! ریترن میشه ولی چیزی رخ نمیده. چیزی چاپ نمیشه. چون پرینت مقدار ریترن شده 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 رباته! شما نمی‌تونین بهش بگین که یه عدد رندوم بده. نمی‌فهمه! باید یه چیز مشخص باشه. مثلاً ۲ ضربدر ۲. اینو می‌تونه جواب 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 عدد بزرگ بوده و باید یه عدد کوچکتري وارد کنم. اگر هم عدد کوچکتري 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:

ایده کلی: جمع عنصر اول + جمع باقی عناصر (صدازدن تابع روی بقیه عناصر).

```
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 قاطبی نشه. مروسومه که وقتی یه اسمی می‌خواین استفاده کنین که با اسم چیزای خود پایتون خاطبی نشه، آخرش یه «underscore» می‌ذارین. Base case چیه؟ خب بذارین فکر کنیم. همینطور تابع رو روی عنصر ۱ به بعد صدا می‌زنه. حالا کجا به پایان می‌رسه؟ وقتی یه عنصر داخل لیست باشه. صرفاً نیازه همون یه عنصر رو ریترن کنه.

```
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 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! 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  
aaabaaa  
hih  
abcba  
aaaa
```

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.

راهنمایی: دو جمله اول دنباله ۱ و ۱ هستن و راحت توی هوا و حفظی می‌تونیم گزارششون کنیم.

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^{۹۰} 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 که از یه ستون که پر دیسک هست، دیسکا رو به همین ترتیب که هستن (به ترتیب بزرگ به کوچیک 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 کوچکتر رو روی دیسک بزرگ‌تر بذارین و دیسک بزرگ‌تر رو نمی‌تونین روی دیسک کوچکتر 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]
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.

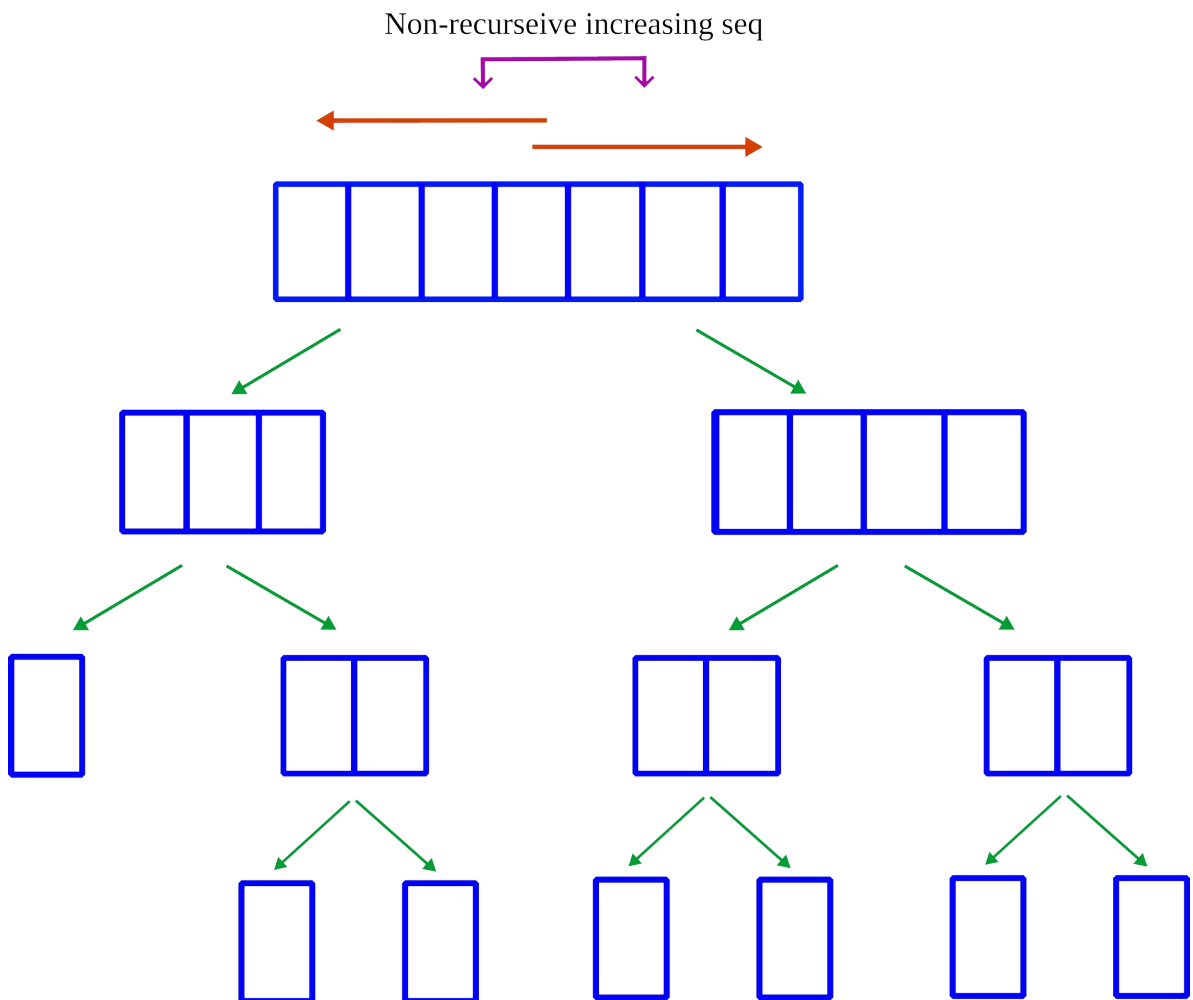
تمرین ۱۰.۲:

longest positive sequence

90 English: 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

راهنمایی: سعی کنید به صورت بازگشتی هی نصف کنید و حلش کنید. اما حواستون باشه که وقتی به صورت بازگشتی حل می کنید، باعث میشین که از وسط بشکنه و عملاً اون دنباله ای که عدد وسطی ممکنه توش جا می گرفت، جا نگیره! پس یه دور هم باید غیر بازگشتی، از وسط به چپ و از وسط به راست حلش کنید. و ببینین تا کجا که برین چپ و تا کجا که برین راست، هنوز دنباله صعودیه. این کار رو تا جایی که اعداد صعودی هستن ادامه بدین.



تمرین ۱۰.۱:

longest increasing sub-sequence:

طولانی ترین بازه ی پشت هم که می تونیم در یه سری اعداد پیدا کنیم که اعضاش صعودی باشن، طولش چنده؟

Input:

1 2 3 4

output:

1 2 3 4

بزرگترین زیردنبال صعودی، ۴ عضو داره. یعنی همین «۱ ۲ ۳ ۴».

Input:

1 3 -1 3

output:

2

بزرگترین زیردنبال صعودی، ۲ عضو داره. یا ۱ و ۳. یا ۱- و ۳.

Input:

7 8 0 1 3 -1

output:

3

راهنمایی: سعی کنین به صورت بازگشتی هی نصف کنین و حلش کنین. اما حواستون باشه که وقتی به صورت بازگشتی حل می‌کنین، باعث میشین که از وسط بشکنه و عملاً اون دنباله‌ای که عدد وسطی ممکنه توش جا می‌گرفت، جا نگیره! پس یه دور هم باید غیر بازگشتی، از وسط به چپ و از وسط به راست حلش کنین. و ببینین تا کجا که برین چپ و تا کجا که برین راست، هنوز دنباله صعودیه. این کار رو تا جایی که اعداد صعودی هستن ادامه بدین.

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 جواب دادن ۱ و ۳ و ۴ و ۵ برای این سؤال کافیه. دومی تایم لیمیت می خورین 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!^{۹۱}

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:

```
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 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 یکسانن یا نه. و این شرط باید برای تمام ایندکسها باشه. یعنی هی and باشه. حالا می‌تونیم اینطوری پیاده‌سازیش کنیم:

۹۱ البته سیستمای مختلف (کامپیوترهای) تفاوت دارن. ممکنه یکی با چیزای کمتر ارور بده.


```
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 دوتای قبلیه. دقیقاً همین رو نوشتیم. اما خب همینطور باید بره عقب تا به یه base case و یه چیز مشخص عددی بخوره.

طبق صورت سوال، جمله اول و دوم همیشه مشخصه و ۱ هست. پس base case رو همین می نویسم:

```
def fib(n):
    if n == 1 or n == 2:
        return 1
    return fib(n-1) + fib(n-2)
```

حالا دیدین که چقدر بازگشتی می تونه کمک کنه که راه حل ساده شه؟ با سه خط کد و عیناً نوشتن مفهوم، تونستیم فیبوناچی رو حل کنیم!

پاسخ ۷:

خب قبول داریم 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 داریم که اگر از سمت چپ بخوایم شروع کنیم (چون **string** شروعش از سمت چپه)، هر چی 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"⁹²

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.⁹³ البته اگر تازه دارین با برنامه‌نویسی آشنا میشین، نیاز نیست حلش کنین. سخته :

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 longest_increasing_subsequence(arr):  
    if len(arr) == 0:  
        return 0  
    if len(arr) == 1:  
        return 1  
    if len(arr) == 2:  
        if arr[0] < arr[1]:  
            return 2  
        else:  
            return 1  
    mid = len(arr) // 2  
    left = arr[:mid]  
    right = arr[mid:]  
    rec = max(longest_increasing_subsequence(left),  
              longest_increasing_subsequence(right))  
    not_rec = 0  
    mid_copy = mid - 1  
    while mid > 0 and arr[mid-1] < arr[mid]:  
        mid -= 1  
        not_rec += 1  
    while mid_copy < len(arr)-1 and arr[mid_copy] <  
arr[mid_copy+1]:  
        mid_copy += 1  
        not_rec += 1  
    return max(rec, not_rec)
```

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 زبون‌های 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 نخوانم 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:

<https://www.geeksforgeeks.org/recursion-practice-problems-solutions/>

• 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.

```
usernames = ['Alex', 'James', 'Maria']  
passwords = ['Alex256', 'James512', 'Maria1024']
```

اما این یکم سخته. باید حواسمون باشه که ایندکس‌ها متناظر اشتباه نشه. اینجا پایتون یه چیزی داره به نام Dictionary. اینطوری می‌تونیم پیاده‌سازی کنیم:

```
database = {  
    'alex': 'alex256',  
    'James': 'James512',  
    'Maria': 'Maria1024'  
}
```

به سمت چپا می‌گیریم کلید (key) و به راستیا می‌گیریم مقدار (value). اما برخلاف لیست، نمی‌تونه داخل خودش دو چیز یکسان داشته باشه. یعنی اگر به صورت زیر بنویسیم:

```
database = {  
    'alex': 'alex256',  
    'James': 'James512',  
    'Maria': 'Maria1024',  
    'James': '1234'  
}  
print(database)
```

چون James دوبار تکرار شده، صرفاً مقدار دومی رو در نظر می‌گیره. چاپش کنیم بهتر متوجه شین:


```
{24: 'alex256', 'James': '1234', 'Maria': 'Maria1024'}
```

اینجا ایندکس نداریم. پس اگر بخوایم به یکی از عناصرش دسترسی پیدا کنیم، اسم کلید رو می‌بریم. مثلاً:

```
database = {
    'alex': 'alex256',
    'James': 'James512',
    'Maria': 'Maria1024',
    'James': '1234'
}

print(database['Maria'])
```

کلیدها هرچیزی جز «دیکشنری، لیست یا set» می‌تونن باشن.^{۹۴} بله! حتی یه عدد اعشاری هم می‌تونه کلید باشه! ولی برای value ها شرط خاصی نداریم. هرچیزی می‌تونن باشن. مثال:

```
database = {
    'alex': 'alex256',
    'James': 'James512',
    2: 'Maria1024',
    'James': '1234'
}

print(database)
```

تابع len رو هم داریم و تعداد رو میده.

متدهای dictionary هم چیزای جالبین برای کار باهاش:

https://www.w3schools.com/python/python_ref_dictionary.asp

update(), copy(), get(), keys(), pop(), items()

مثلاً بخوایم یه چیز آپدیت یا اضافه کنیم:

```
database = {
    'alex': 'alex256',
    'James': 'James512',
    'Maria': 'Maria1024',
}

database.update({'James': 'hi'})
print(database)
```

^{۹۴} اما Tuple می‌تونه کلید باشه.

درواقع نگاه می‌کنه که اگر کلید James نبود، اضافه‌اش می‌کنه. اگر هم از قبل 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. مثل string و لیست، اینجا هم for ... in داریم:

```
database = {
    'alex': 'alex256',
    'James': 'James512',
    'Maria': 'Maria1024',
}

for key in database:
    print(key)
```

این کار میاد روی کلیدها حرکت می‌کنه:

```
alex
James
Maria
```

قبول داریم وقتی که کلید رو داشته باشیم، عملاً می‌تونیم به value ها هم دسترسی داشته باشیم؟

```
database = {
    'alex': 'alex256',
    'James': 'James512',
    'Maria': 'Maria1024',
}

for key in database:
    print(f'{key}: {database[key]}')
```

یعنی روی کلیدها که داری حرکت می‌کنی و چاپشون می‌کنی، در کنارش value هم چاپ کن. خروجیش:

```
alex: alex256
James: James512
Maria: Maria1024
```

البته می‌تونستیم از متد items هم استفاده کنیم:

```

database = {
    'alex': 'alex256',
    'James': 'James512',
    'Maria': 'Maria1024',
}

for pair in database.items():
    print(pair)

```

یعنی `database.items` بود که یه لیست که شامل یه سری `pair` بود رو بهت میداد،^{۹۵} برو دونه‌دونه روی اون لیست حرکت کن و `item` هاشو پرینت کن:

```

('alex', 'alex256')
('James', 'James512')
('Maria', 'Maria1024')

```

حتی می‌تونیم یکم با این بازی کنیم:

```

database = {
    'alex': 'alex256',
    'James': 'James512',
    'Maria': 'Maria1024',
}

for key, value in database.items():
    print(key, value)

```

مگه متد `items` به ما یه لیست شامل `pair` (دوتایی) کلید و مقدارها رو نمیداد؟ حالا من میگم برای کلید و `value` های درون `items` (یعنی هر بار میاد یکی از اون دوتایی‌ها رو انتخاب می‌کنه و متناظر به `key` و `value` انتساب میده. حالا می‌تونم اعضا رو پرینت کنم.

همونطور که قبلاً گفتیم، روی چیزایی مثل لیست که چند عضوین، اگر می‌خوان تعداد رو تغییر بدین، مثلاً وقتی می‌خوان یه چیزی رو حذف کنین و همزمان حرکت کنین، روی یه لیست هردو کار رو با هم نکنین. دیکشنری هم همینه. یعنی نباید این کار کنین. بلکه باید یه کاپی بگیرین و روی کاپی حرکت کنین و تغییرات رو توی لیست عادی اعمال کنین:

⁹⁵ [('alex', 'alex256'), ('James', 'James512'), ('Maria', 'Maria1024')]

```

database = {
    'alex': 3,
    'James': 2,
    'Maria': 3,
    'Me': 1,
}

for name in database.copy():
    if database[name] == 3:
        database.pop(name)
print(database)

```

یکم با چیزای بالا بازی کنین و تمرین کنین که خیلی مهمن.

تمرین!

۱- فرض کنین دو لیست زیر رو داریم:

```

name_list = ['Tom', 'Lily', 'Rose', 'Sarah', 'Jack']
last_name_list = ['Smith', 'Jackson', 'Johnson', 'Lee', 'Taylor']

```

درواقع یه دیتابیس رو با دو لیست پیاده‌سازی کردم. اما کار با این سخته. می‌خوام با استفاده از این دو لیست، یه دیکشنری بسازین که کلیدها نام، و فامیل‌ها value باشن.

۲- فرض کنین دیکشنری زیر رو داریم:

```

d = {
    'key1': 'Hello world! Hello Python! Hello coding!',
    'key2': 'Use programming languages as a tool to solve problems',
    'key3': 'Python is an easy to learn, powerful programming language.'
}

```

می‌خوایم تک‌تک کلمات داخل value ها رو چاپ کنیم. (قاعدتاً باید با فاصله جدا شن).

output

```
Hello
world!
Hello
Python!
Hello
coding!
Use
programming
languages
as
a
tool
to
solve
problems
Python
is
an
easy
to
learn,
powerful
programming
language.
```

این رو پیاده‌سازی کنین.

۳- فرض کنین دیتابیس زیر رو داریم.

```
database = {
    'alex': '1234',
    'James': '2345',
    'Maria': '3456',
    'Me': '4567',
}
```

می‌خوایم جای کلیدها و value ها رو عوض کنیم. این کار رو انجام بدین.

پاسخنامه:

پاسخ ۱:

```
name_list = ['Tom', 'Lily', 'Rose', 'Sarah', 'Jack']
last_name_list = ['Smith', 'Jackson', 'Johnson', 'Lee', 'Taylor']
name_database = {}
for i in range(len(name_list)):
    name_database.update({name_list[i]: last_name_list[i]})
```

یه `for` زدم روی دو تا لیست (طول دو لیست برابر) و در هر بار حرکت روی ایندکس، ایندکس‌های متناظر نام و فامیل رو قرار دادم. نام کلید و فامیل `value` اش.

پاسخ ۲:

```
d = {
    'key1': 'Hello world! Hello Python! Hello coding!',
    'key2': 'Use programming languages as a tool to solve problems',
    'key3': 'Python is an easy to learn, powerful programming language.'
}
inverted_index = {}
for key, val in d.items():
    words = val.split()
    for word in words:
        print(word)
```

اول یه `for` زدم روی دیکشنری. بعد هر بار که به `val` دسترسی پیدا کردم، میام `split` اش می‌کنم و یه لیست از کلمات می‌سازم. بعد روی لیست حرکت می‌کنم و کلمات رو پرینت می‌کنم.

پاسخ ۳:

```
database = {
    'alex': '1234',
    'James': '2345',
    'Maria': '3456',
    'Me': '4567',
}

rev_database = {}
for key, value in database.items():
    rev_database.update({value: key})

database = rev_database.copy()
print(database)
```

مقدار به کلید دیکشنری می‌تونه خودش به 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)
```

این دیکشنری دو تا کلید داره. اما value های اون کلیدها، خودشون دیکشنری هستن.

چون دیکشنری دو بعدیه، پرینت کردنش خیلی نامرتبه. برای همین پایتون به فانکشن پرینت دیگه‌ای به نام «pprint» داره که میشه چیزای دوبعدی و سه بعدی رو قشنگ‌تر پرینت کرد:

```
from pprint import pprint  
pprint(school_dict)
```

حالا بیایم مثلاً به value خاصی رو پرینت کنیم:

```
print(school_dict['teachers']['Hannah'])
```

به syntax توجه کنین که اشتباه ننویسینش!

درواقع می‌گیم به school_dict داریم که شامل دو عنصره. یکی students یکی teachers. حالا هرکدوم شامل به دیکشنری هستن!

بین این دو عنصر به 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 به دیکشنری هست که value برای students هست

و دومی به 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 برنامتون 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 ارور نمی‌خوره و کرش نمی‌کنه و این خیلی بهتره!

• Dictionary Comprehensions

مثل لیست، میشه for رو درون dictionary هم به کار برد. مثلاً:

```
first_names = ['Bruce', 'Edward', 'Ronald']
```

```
last_names = ['Schneider', '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.^{۹۶}

^{۹۶} اسمایی که نوشتیم، از بزرگترین افراد در زمینه کامپیوتر و رمزنگاری (cryptography) هستن (:

تمرین:

۱- فرض کنیم ما می‌خواهیم تابعی بنویسیم که به استرینگ بگیرد و تعداد تکرار کرکتهای مختلف رو بهمون بده. مثلاً بگه که «a» چند بار تکرار شده. «b» چند بار تکرار شده. «c» چند بار توش تکرار شده و (همچنین برای کرکتهای بزرگ هم بگه).
این کار رو هم با لیست و هم با دیکشنری پیاده‌سازی کنیم.

input:

```
KJTvsVMhAbazTCLzRpyenTfBFsKVAewAYyVAoNvA
```

output:

```
===== List =====
```

```
--- Upper ---
```

```
A 5
```

```
B 1
```

```
C 1
```

```
D 0
```

```
E 0
```

```
F 1
```

```
G 0
```

```
H 0
```

```
I 0
```

```
J 1
```

```
K 2
```

```
L 1
```

```
M 1
```

```
N 1
```

```
O 0
```

```
P 0
```

```
Q 0
```

```
R 1
```

```
S 0
```

```
T 3
```

```
U 0
```

```
V 3
```

```
W 0
```

```
X 0
```

مثلاً Ronald Rivest نویسنده یکی از معروف‌ترین کتاب‌های الگوریتم، برنده جایزه Turing که به نوبل کامپیوتر معروفه، استاد دانشگاه MIT
و ...

```
Y 1
Z 0
--- Lower ---
a 1
b 1
c 0
d 0
e 2
f 1
g 0
h 1
i 0
j 0
k 0
l 0
m 0
n 1
o 1
p 1
q 0
r 0
s 2
t 0
u 0
v 2
w 1
x 0
y 2
z 2
===== Dict =====
--- Upper ---
a 1
b 1
c 0
d 0
e 2
f 1
g 0
h 1
i 0
j 0
```

k 0
l 0
m 0
n 1
o 1
p 1
q 0
r 0
s 2
t 0
u 0
v 2
w 1
x 0
y 2
z 2
--- Lower ---
A 5
B 1
C 1
D 0
E 0
F 1
G 0
H 0
I 0
J 1
K 2
L 1
M 1
N 1
O 0
P 0
Q 0
R 1
S 0
T 3
U 0
V 3
W 0
X 0

Y 1

Z 0

همونطور که می بینین، خروجی رو هم برای لیست و هم برای دیکشنری نشون دادیم.

راهنمایی:

قبول دارین که تعداد حروف انگلیسی ۲۶ تاست؟ پس می تونیم اینطوری پیاده سازی کنیم:

```
def is_upper(c):
    return 'A' <= c <= 'Z'

def is_lower(c):
    return 'a' <= c <= 'z'

def letter_freq(txt) -> tuple[list[int], list[int]]:
    upperl = [0 for i in range(26)]
    lowerl = [0 for i in range(26)]
    for c in txt:
        if is_upper(c):
            upperl[ord(c) - ord('A')] += 1
        elif is_lower(c):
            lowerl[ord(c) - ord('a')] += 1
    return upperl, lowerl

def letter_freq2(txt) -> tuple[dict[str, int], dict[str, int]]:
    lowerd = {char: 0 for char in 'abcdefghijklmnopqrstuvwxyz'}
    upperd = {char: 0 for char in 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'}
    for c in txt:
        if is_upper(c):
            upperd[c] += 1
        elif is_lower(c):
            lowerd[c] += 1
    return lowerd, upperd

s = input()
upperl, lowerl = letter_freq(s)
upperd, lowerd = letter_freq2(s)

print('==== List =====')

print('--- Upper ---')
for i in range(26):
    print(chr(ord('A') + i), upperl[i])

print('--- Lower ---')
for i in range(26):
    print(chr(ord('a') + i), lowerl[i])

print('==== Dict =====')

print('--- Upper ---')
for key, val in upperd.items():
    print(key, val)

print('--- Lower ---')
for key, val in lowerd.items():
    print(key, val)
```

تعداد

۹۷

تولید قسمت لیست:

اول دو لیست ساختم. یکی برای کرکتهای کوچک و دیگری برای کرکتهای بزرگ. این لیست طولش ۲۶ هست. (ایندکس صفر تا ۲۵). این لیست برای آینه که به طور متناظر تعداد هر کرکتر رو ذخیره کنم. در ابتدا ۲۶ تا صفر می ریزیم تو هر دو. چون فعلاً تعداد رو نشمردم و تعداد هر کدوم صفر هست فعلاً. بعدش روی متن حرکت می کنم. خب اول باید چک کنم که ببینم حرف ما کوچیکه یا بزرگه. اگر کوچیک بود باید لیست کوچیکا رو آپدیت کنم. اگر بزرگ بود، باید لیست بزرگا رو آپدیت کنم. حالا چطور لیست رو آپدیت کنم؟

باید ایندکس صفر تعداد «a» ها باشه. ایندکس ۱ تعداد «b» ها. ایندکس ۲ تعداد «c» ها و ایندکس ۲۵، تعداد «z» ها باشه. پس میام اسکی رو منهای «a» می کنم. که رنج هر کرکتر بیاد روی ۰ تا ۲۵. این اینطوری به دست میاد:

`ord(c) - ord('A')`

درواقع با این می تونیم جایگای ایندکس هر ایندکس رو بیابیم. در پایان هم میایم هر دو لیست رو ریترن می کنیم. - عه! مگه میشه دو تا چیز رو با هم ریترن کرد؟ + بله میشه! فقط باید حواسمون باشه که اونور وقتی می خوایم تابع رو انتساب بدیم به متغیر، دو تا متغیر بذاریم. به ترتیب موارد ریترن شده رو می ذاره توشون. همونطور که دیدین منم همین کار کردم. یعنی دو تا متغیر با کاما جدا کردم و تابع رو انتساب دادم بهشون.

تولید قسمت ریکشنری:

میایم یه دیکشنری با کلید حروف الفبا و value صفر برای حروف بزرگ و یکی برای حروف بزرگ می سازم. پایینشم معلومه. اگر بزرگ بود، به بزرگا اضافه می کنیم.

پرینت قسمت لیست:

روی ایندکس های ۰ تا ۲۵ حرکت می کنیم و هر بار پرینتش می کنیم. می گیم خود مقدار اون ایندکس و همچنین کرکتر متناظر رو چاپ کن. کرکتر متناظر رو با اسکی می سازم. ایندکس رو بعلاوه مقدار عددی اسکی «A» کنم. بعد کرکترش کنم.

به قسمتای

```
-> tuple[list[int], list[int]]
```

هم کاری نداشته باشین. نیاز نیست بلدش باشین. این رو قبلاً دربارش توی کادرای آبی صحبت کردیم و یکم پیشرفتس.

• Try and Except

یکی از مهم‌ترین هنرهای یه برنامه‌نویس آینه که نذاره برنامش به ارور بر بخوره.
- چرا؟

+ چون ارور خوردن غیرمنتظره، عملاً منجر میشه که برنامه متوقف و تموم بشه. خب خیلی بده که یهو کاربری که داره با برنامه کار می‌کنه، برنامش یهو بسته بشه. مثلاً کد زیر رو در نظر بگیریم:

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

این کد یه عدد می‌گیره. اما شما به عنوان برنامه‌نویسی، همیشه باید یه چیز رو مدنظر قرار بدین: ((هیچ‌وقت به کاربر اعتماد نکنید! همیشه امکان این وجود داره که کاربر یه چیز اشتباه (به صورت عمدی یا غیرعمدی)، بده.))
مثلاً اینجا ممکنه به جای یه عدد، یه متن بده. خب مطمئناً به ارور بر می‌خوریم:

```
ValueError: invalid literal for int() with base 10:
```

و عملاً برنامه تموم میشه.
شما نباید بذارین این اتفاق بیوفته. یکی از راهکارهایی که میشه جلوی ارور خوردن رو گرفت، استفاده از ساختاری به نام «Try and Except» هست. به طور خلاصه اینطوری ترجمه میشه که:

اینو امتحان کن:

...

اگر نشد اینو امتحان کن:

...

مثال:

```
try:
    print(age)
except:
    print("age is not defined")
```

میگه که اول تست کن بین آیا می‌تونی age رو پرینت کنی؟ اگر نشد و به ارور خوردی، بیا قسمت «except» رو اجرا کن.

مثال:

```
try:
    age = int(input("Enter your age: "))
except: # ValueError
    print("You entered an invalid value.")
```

یعنی اول امتحان کن ببین که میشه آیا به عدد بگیری؟ اگر به وقت ارور خوردی، نمی‌خواد ارور رو بنویسی. به جاش بیا قسمت «except» و متن رو چاپ کن. می‌تونیم با کمک این ورودی رو به صورت بهتری بگیریم. بگیریم تا وقتی که try کامل اجرا نشده (به ارور می‌خوری)، بمون و همینطور ورودی بگیر. اینو می‌تونیم اینطوری بنویسیم:

```
while True:
    try:
        age = int(input("Enter your age: "))
        break
    except: # This will catch any error
        print("You entered an invalid value.")
```

بهش می‌گیریم همیشه داخل while بمون. اول بیا به عدد صحیح بگیر. اگر ارور خورد، از همون خطی که عدد گرفته، می‌پره میره توی «except» و «except» اجرا میشه. و باز داخل while میاد. اما اگر به ارور نخورد، میره بعد خط گرفتن ورودی، می‌بیننه نوشتیم break. پس از while خارج میشه. عملاً اینطور می‌تونیم بگیریم تا وقتی ورودی درستی نداده، نذار بره جای دیگه‌ای و نگهش دار تا ورودی درست رو بده.

پایتون می‌گه شاید برای ارورهای مختلف، بخوای کارهای مختلفی کنی. برای همین جلوی except می‌تونی بنویسی که اگر چه اروری رخ داد، چیکار کنم. توی پایتون هر نوع گروه از ارورها، به نام خاص دارن. مثلاً بیاین چند موردشو ببینیم:

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

اگر مقدار عددی ندیم:

```
ValueError: invalid literal for int() with base 10:
```

اسم این ارور «ValueError» هست.
یا مثلاً:

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

چون x تعریف نشده، ارور می‌ده:

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

اسم این نوع ارور، «NameError» هست.

از همین اسما می‌تونیم استفاده کنیم برای بازنویسی try and except:

```
try:
    print(age)
except NameError:
    print("age is not defined")
```

یعنی اگر دیدی «NameError» رخ داد، بیا سراغ except.

یا مورد دیگه:

```
try:
    age = int(input("Enter your age: "))
except ValueError:
    print("You entered an invalid value.")
```

یعنی اگر این نوع ارور رخ داد بیا اینجا.

خب اما گاهی ما ممکنه صرفاً تو فکرمون، یکی دو نوع ارور بیاد و همه ارورهایی که ممکنه رخ بدن رو تو ذهنمون بهش پی نبریم. پایتون گفته ایرادی نداره! می‌تونی اینطوری بنویسی:

```
try:
    age = int(input("Enter your age: "))
except ValueError:
    print("You entered an invalid value.")
except:
    print("Something went wrong.")
```

یعنی اگر «ValueError» داد اون کار کن. وگرنه یه پرینت دیگه کن.

مثال دیگه:


```

while True:
    try:
        num1 = int(input("Enter a number: "))
        num2 = int(input("Enter another number: "))
        result = num1 / num2
        print("Result:", result)
        break

    except ValueError: # If the user enters invalid input
        print("Invalid input. Enter a valid number.")

    except ZeroDivisionError: # if num2 is 0
        print("Cannot divide by zero. Enter a non-zero number.")

    except: # if any other exception occurs
        print("An error occurred:")

```

اگر دقت کنیم، توی کد ممکنه دو ارور رخ بده. یکی اینکه کاربر مقداری که اینتیجر نیست رو وارد کنه برای ورودی، یا ممکنه عدد دوم رو صفر داده باشه و تقسیم بر صفر رخ بده. برای این دو مجزا تعریف کردم که چه پیامی چاپ شه. بعدشم گفتم باز ممکنه اروری رخ بده که من نمی‌دونم. پس یه `except` بدون اسم ارور هم می‌ذارم براش که هر ارور دیگه‌ای که رخ داد بره سمت این.

```

try:
    f = open("file.txt")
    try:
        f.write("hi")
    except:
        print("Something went wrong when writing to the file")
    finally:
        f.close()
except:
    print("Something went wrong when opening the file")

```

Finally مهمه!!!!

```

while True:
    try:
        num1 = int(input("Enter a number: "))
        num2 = int(input("Enter another number: "))
        result = num1 / num2
        print("Result:", result)
        break

    except ValueError: # If the user enters invalid input
        print("Invalid input. Enter a valid number.")

    except ZeroDivisionError: # if num2 is 0
        print("Cannot divide by zero. Enter a non-zero number.")

    except: # if any other exception occurs
        print("An error occurred:")

    finally:
        print('=====')

```

فرقش با نداشتن finally.

به سری مشکلات رایج

• else اضافی:

```

def foo(n: int) -> bool:
    if n > 5:
        return True
    else:
        return False

```

- به نظرتون آیا اصلاً اینجا else نیاز یا نه؟

+ نه نیاز نیست. چون دو حالت هست.

• وارد if میشه و ریترن صورت می گیره و تابع تموم میشه.

• وارد if نمیشه و میره وارد else میشه.

اگر وارد if بشه، تابع تموم میشه. اگر نه، قطعاً وارد else میشه. پس فرقی نداره else بذارین یا نه.

چه بذارین چه نذارین یه اتفاق میوفته. قطعاً اون خط بعدی اجرا میشه.

مثال دیگر:

```
def foo() -> None:
    for i in range(5):
        if i == 2:
            break
        else:
            print('Not yet')
    print('end')
```

چه `else` باشه یا نباشه یه چیز رخ میده. یا وارد `if` میشه و `for` تموم میشه و اصلاً سراغ خط بعدی نمیره. یا نمیشه و صددرصد `else` اجرا میشه.

• match case (switch case)

فرض کنین که یه سن می گیریم. بعد میگی که 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 پیاده سازی کنیم:

```
age = int(input())
if age == 18:
    print(1)
elif age == 20:
    print(2)
elif age == 22:
    print(3)
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 بگیریم. 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.

Type hint

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.

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 هست که `int` هست. خروجی هم عبارتی `boolean` هست. تذکر! این صرفاً یه راهنماس. وگرنه باز می‌تونیم هر تایپی بخوایم به تابع پاس بدیم. این کلمات صرفاً راهنمایی هستن. یعنی می‌تونین `string` هم پاس بدین. این صرفاً یه راهنماس که خودمون بهتر بفهمیم. یعنی مثلاً بعد یه ماه برگشتیم به کد، سریع با نگاه کردن به پارامترها تشخیص بدیم چیکار می‌کنه. همچنین ابزارهایی هستن که می‌تونن با خوندن این‌ها، مشکلات رو پیدا کنن. مثلاً می‌گن گفته بودی `int` می‌گیره ولی اشتباهی بهش `string` پاس دادی. یا مثلاً اشتباهی فلان متغیر رو ریختی تو فلان چیز. شاید در نگاه اول سخت بیاد ولی بهم اعتماد کنین. در پروژه‌های بزرگ، به شدت کمکتون می‌کنه که کدتون خوانا شه.

مثال ۳:

اگر بخوایم بگیریم هیچی بر نمی‌گردونه، می‌گیم «None» (هیچی) بر می‌گردونه:

```
def foo(t1: tuple[int, int], l1: list[int],  
        d1: dict[str, int], l2: list[list[int]]) -> None:  
    pass
```

همونطور که دیدن وقتی بخوایم بگیریم که تایپ لیست چیه، می‌نویسیم `list` و چیزایی که توشه رو می‌نویسیم. یا تایپ تاپل هم می‌گیم تاپل هست و چی توشه. یا تایپ دیکشنری هم مطابق شکل. اولی تایپ کلید و دومی `value`. بعدش `l2` لیست دو بعدی. که می‌گه یه لیسته. که توش لیستی شامل اینت‌جراهاست.

مثال ۴:

اگر بخوایم بگیریم که تایپش ممکنه هرچیزی باشه، از کلیدواژه «Any» استفاده می‌کنیم:

```
from typing import Any  
  
def add(a: Any, b: Any) -> Any:  
    return a + b
```

فقط حواسمون باید باشه که تایپ Any رو از typing ایمپورت کنیم.

مثال ۵:

اگر از raise استفاده کنیم و بگیم واقعاً هیچی برنمی‌گردونه و فقط raise می‌کنه، از کلیدواژه «noReturn» استفاده می‌کنیم:

```
from typing import NoReturn

def foo() -> NoReturn:
    raise ValueError("This function never returns")
```

• Static code analysis

هرچقدر هم ما برنامه‌نویس خوبی باشیم، بالاخره ممکنه یه جاهایی از کد حواسمون نباشه. یا یه کارهایی کنیم که ندونیم اشتباهه.

خب یه راهش آینه که کتاب بخونیم تا با اشتباهات رایج آشنا شیم.

این خوبه ولی کافی نیست! چرا؟ چون خیلی از چیزایی که می‌خونیم رو بعداً یادمون میره. همچنین خیلی از چیزایی که می‌خونیم هم شاید به کارمون نیاد. پس درسته کمک‌کننده، اما کافی نیست! چه خوب بود که وقتی که داریم یه پروژه می‌زنیم، یه نفر بغل دستمون می‌نشست و بهمون راهنمایی می‌کرد. مثلاً می‌گفت اینجا یادت رفته این متغیر رو تعریف کنیا! اونجا شاید فلان مشکل پیش بیاد! اینجا اگر کاربر یه چیز اشتباه بده کدت به ارور می‌خوره‌ها! درستش کن.

خب اگر بهتون بگم یه سری برنامه هستن که میان کدتون رو چک می‌کنن و مشکلاتتون رو میگن چی؟ خیلی خوب نیست؟!

بله یه سری آدم نشستن برنامه‌هایی نوشتن که کدهای شما رو اتوماتیک می‌خونن و چک می‌کنن. درواقع شما می‌تونن توی Notepad هم کد بزنی! اما خوبه آیا؟! نه! بهتره بری در یه سری محیط که مخصوص برنامه‌نویسی هستن کد بزنی. این محیطا یه سری از این راهنمایی‌ها رو به شما می‌کنن. معروف‌تریناش هم:

- Pycharm
- Visual Studio Code + Extensions

هستن.

خب ما اینجا فعلاً با دومی که ساده‌تره کار می‌کنیم.^{۹۸} شما باید اکستنشن‌های اولیه که گفتیم و همچنین اکستنشن «Pylance» رو نصب کنیم.

همچنین توی cmd، دو دستور

دو دستور زیر رو به ترتیب بنویسین و اینتر بزنین و منتظر بمونین تا با موفقیت اجرا بشه (اول یکی انجام شه و بعد دیگری رو اجرا کنین):

```
pip install pylint
pip install bandit
```

این برنامه‌ها اتومات هستن. ممکنه یه وقتایی اشتباه هم بگن! پس اینطور نیست که هر وقت ایراد گرفتن، ۱۰۰ درصدِ ۱۰۰ درصد درست گفته باشن ولی خب به عنوان یه دستیار خیلی می‌تونن بهتون کمک کنن. در نهایت این شمایی که تصمیم می‌گیرین که آیا این اشتباهه یا نه و درستش کنین یا نه.

- Pylance

این افزونه به صورت پیش‌فروش و خودکار و به صورت آن واحد میاد بهتون مشکلات کد رو می‌گه. مثلاً بیایم یه کد بنویسیم:

```
def age_group(age):
    if 0 < age < 18:
        res = 'child'
    elif 18 <= age < 65:
        res = 'adult'
    elif age >= 65:
        res = 'senior'
    return res
```

خب همونطور که می‌بینیم، زیر res خط کشیده. و اگر موس رو روش ببریم، نوشته:

```
"res" is possibly unbound Pylance(reportUnboundVariable)
```

خب به نظرتون مشکل کد چیه؟

خب قبول دارین که متغیر res داره توی if و elif ها ساخته میشه؟ خب فرض کنین سن پاس داده‌شده به تابع، در بازه هیچکدوم از if و elif ها نباشه. خب پس وارد هیچکدوم از if و elif ها نمیشه. تازه else هم نداریم که وارد اون بشه. پس عملاً «res» ما ساخته نمیشه. پس توی خط ریترن، به مشکل بر خواهیم خورد. چون می‌گه که من وارد هیچ if و elif ای نشدم و res ساخته نشده. اصلاً res تعریف نشده! من چطور چیزی که تعریف نشده رو ریترن کنم؟ مثلاً بیایم یه کد بنویسیم که بفهمیم کی ارور میده:

^{۹۸} که البته Pycharm بسیار ابزار قدرتمند و خوبیه که می‌تونه خیلی کمک کنه. درواقع vscode یه نوت‌پد خیلی پیشرفته‌تره که برای هر زبونی می‌تونه استفاده بشه. اما Pycharm یه ابزار خیلی قدرتمند مخصوص پایتونه که دقیقاً برای پایتون ساخته شده.


```
def age_group(age):
    if 0 < age < 18:
        res = 'child'
    elif 18 <= age < 65:
        res = 'adult'
    elif age >= 65:
        res = 'senior'
    return res

print(age_group(-1))
```

خب دیدین؟ ارور می‌ده:

```
line 9, in age_group
    return res
UnboundLocalError: local variable 'res' referenced before
assignment
```

می‌گه قبل اینکه بهش چیزی انتساب بدی (assign کنی - یا همون تعریفش کنی)، خواستی بهش ریفرنس بدی (خواستی بهش دسترسی پیدا کنی).

پس دیدین؟ این آنالیزورهای کد خیلی می‌تونن جلوی ارورها و مشکلات برنامه رو بگیرن.

حالا نمونه‌های دیگه، با pylint:

مثال:

یا مثلاً یه کد دیگه رو ببینیم:

```
row_cnt = 3
col_cnt = 3
l = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
for i in range(row_cnt):
    for j in range(col_cnt):
        print(l[i][j], end=' ')
    print()
```

یه ماتریس ساختم. بعد تعداد ردیف و ستون هم تعریف کردم. بعدش ماتریسو چاپ کردم. حالا اگر Pylint رو اجرا کنیم، می‌گه که:

```
***** Module main
main.py:1:0: C0114: Missing module docstring (missing-module-
docstring)
main.py:1:0: C0103: Constant name "row_cnt" doesn't conform to
UPPER_CASE naming style (invalid-name)
main.py:2:0: C0103: Constant name "col_cnt" doesn't conform to
UPPER_CASE naming style (invalid-name)

-----
Your code has been rated at 5.71/10 (previous run: 0.00/10, +5.71)
```

تهش یه امتیازی به کد داده که کدت چقدر خوبه. حالا بیایم دقیق تر تحلیلش کنیم که چی می‌گه.

```
main.py:1:0: C0103: Constant name "row_cnt" doesn't conform to
UPPER_CASE naming style (invalid-name)
```

گفته در فایل main.py، خط یک، کرکتر صفر، یه چیزی پیدا کردم. حالا C و اینا چی هستن؟ طبق داکيومنت خود ⁹⁹PyLint:

```
Output:
Using the default text output, the message format is :
MESSAGE_TYPE: LINE_NUM:[OBJECT:] MESSAGE
There are 5 kind of message types :
* (C) convention, for programming standard violation
* (R) refactor, for bad code smell
* (W) warning, for python specific problems
* (E) error, for probable bugs in the code
* (F) fatal, if an error occurred which prevented pylint from
doing
further processing.
```

حالا زیاد نیست درگیرش بشین. خلاصه بدونین که یه مشکل پیدا کرده. عدد جلوشم کد مشکله. حالا بعدش نوشته که من حس می‌کنم که این row_cnt، یه مقدار ثابت هست. یعنی تعداد سطرها در طول کد تغییر نمی‌کنه و ثابت. چون ثابت هست، پایتون نویسا یه قرارداد دارن که می‌گن برای اینکه بهتر کدها رو متوجه بشیم، تمام کرکترهای متغیرهای ثابت رو حروف بزرگ بذارین که کد خواناتر باشه. پس بیایم اصلاحش کنیم:

```
ROW_CNT = 3
COL_CNT = 3
l = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
for i in range(row_cnt):
    for j in range(col_cnt):
        print(l[i][j], end=' ')
    print()
```

99 <https://pylint.readthedocs.io/en/stable/tutorial.html#getting-started>

حالا دوباره اجراش کنیم:

```
***** Module main
main.py:1:0: C0114: Missing module docstring (missing-module-
docstring)

-----
Your code has been rated at 8.57/10 (previous run: 5.71/10, +2.86)
```

می‌بینین؟ درست شد. امتیاز کدم هم از ۵.۷۱ شد ۸.۵۷.
اون یکی باقی‌مونده هم می‌گه که ابتدای کدت خوبه توضیح بدی که این کد اصلاً داره چیکار می‌کنه.
خب یه توضیح درباره فایل هم میدیم:

```
"""Print a 2D list in matrix format."""

ROW_CNT = 3
COL_CNT = 3
l = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
for i in range(row_cnt):
    for j in range(col_cnt):
        print(l[i][j], end=' ')
    print()
```

دوباره Pylint رو اجرا کنیم:

```
Your code has been rated at 10.00/10 (previous run: 8.57/10, +1.43)
```

خب کدم شد ۱۰ از ۱۰ :)
Mypy و Pytype و Pylance هم مشکلی پیدا نکرده بودن.

مثال:

```
import cryptography
import math
```

خروجی Pylint:

```
***** Module main
main.py:1:0: C0114: Missing module docstring (missing-module-
docstring)
main.py:2:0: C0411: standard import "import math" should be placed
before "import cryptography" (wrong-import-order)
main.py:1:0: W0611: Unused import cryptography (unused-import)
main.py:2:0: W0611: Unused import math (unused-import)

-----
Your code has been rated at 0.00/10
```

میگه خط ۲ به مشکل پیدا کردم. اینکه شما باید لایبرری‌هایی که استاندارد پایتون هستن (مثل لایبرری math) رو اول import کنی. یعنی جاشون باید برعکس می‌بود:

```
import math
import cryptography
```

چون math به لایبرری استاندارد هست ولی cryptography به لایبرری عادی هست که استاندارد نیست. چیزیه که به آدم عادی نوشتنش و جزء استاندارد حساب نمیشه. (این لزوماً چیز بدی نیست که جزء استاندارد نباشه. صرفاً می‌گه پیشفرض و استاندارد زبون نیست. به لایبرری معمولیه و لزومی هم نداره که بد باشه!)

+ یا مثلاً همینجا هم گفته که در خط ۱ و ۲، import کردی ولی استفادشون نکردی. حواست باشه.

مثال:

```
my_list = [1, 2, 3]
out_of_band_access = my_list[5]
```

این کد خیلی واضح مشکل داره! اونم اینکه ما اصلاً ایندکس ۵ نداریم که! پس ارور می‌خوریم. اما این مشکل رو نه «pylint» نه «mypy» نه «pytype» نه «pyre» نه اکستنشن «pylance» پیدا نکردن! درسته اکثراً برای مشکلات نگارشی و clean code ساخته شدن، اما این به چیز خیلی واضحی هست که انتظار می‌رفت بگنش.

مثال:

```
num = 0
print(2 / num)
```

این هم تقسیم بر صفر داره و خیلی واضحه. اما هیچکدوم تشخیص ندادن. این رو هم هیچکدوم تشخیص ندادن.

آره درسته اگر اجزاش کنیم ارور می‌خوریم و معلوم میشه، ولی این خیلی ساده بود. اینطوری هم نبود که num از اول صفر نباشه و توی روند برنامه صفر شده باشه که عملاً نیاز به چک و اجرای کد توی پس‌زمینه

دقیقاً اینجاست که یک ابزار به تنهایی کافی نیست! خوبه از چند تا ابزار استفاده کنین و کداتون رو آنالیز کنین. چون ممکنه یکیشون به مشکل رو پیدا کنه ولی دیگری نکنه. حتی ممکنه مشکلی وجود داشته باشه که هیچ ابزاری پیدا نکنه. خب اینا ابزارن دیگه! معجزه نیستن که همه چیز رو پیدا کنن!

- ## Security Code analysis

کد قدیمی! لایبرری قدیمی مشکل امنیتی و... توضیح بده

ما کلی کد می نویسیم که ممکنه مشکل امنیتی داشته باشن و ندونیم. درسته ابزارهایی مثل «Pylint» می تونن بعضی مشکلات که ممکنه منجر به مشکلات امنیتی بشه رو پیدا کنن ولی تخصصشون نیست و در یافتن مشکلات امنیتی ضعیف عمل می کنن. چه خوب بود که همونطور که ابزارهایی برای مشکلات نوشتاری و ارورها بود، ابزارهایی برای یافتن مشکلات امنیتی هم بود. خب بله هست! دو ابزار معروف:

- ### bandit

```
# Install with:  
pip install bandit  
  
# Usage:  
bandit main.py
```

- ### snyk¹⁰⁰

```
# Usage:  
snyk code test main.py
```

مثال:

فرض کنیم من یه کد نوشتم:

```
username = input()
password = input()
if username == 'admin' and password == '123456':
    print('Login successful')
```

خب بیایم «bandit» رو روش اجرا کنیم:

```
Test results:
>> Issue: [B105:hardcoded_password_string] Possible hardcoded
password: '123456'
Severity: Low Confidence: Medium
CWE: CWE-259 (https://cwe.mitre.org/data/definitions/259.html)
More Info:
https://bandit.readthedocs.io/en/1.7.6/plugins/b105_hardcoded_passw
ord_string.html
Location: main.py:3:39
2     password = input()
3     if username == 'admin' and password == '123456':
4     print('Login successful')

-----

Code scanned:
Total lines of code: 4
Total lines skipped (#nosec): 0

Run metrics:
Total issues (by severity):
  Undefined: 0
  Low: 1
  Medium: 0
  High: 0
Total issues (by confidence):
  Undefined: 0
  Low: 0
  Medium: 1
  High: 0
Files skipped (0):
```

خب گفته یه مشکل امنیتی پیدا کردم! ریسکش متوسطه. تو حواست نبوده و رمز واقعی سیستم رو توی کدت گذاشتی! این خیلی بده! چون هرکسی دسترسی به کد داشته باشه (هرکسی که می‌خواد کد رو اجرا کنه)، راحت می‌تونه کد رو باز کنه و رمز سیستم رو در بیاره! این خیلی بده! رمز سیستم و اینا هیچ‌وقت نباید تو کد باشه!

- چیکار کنیم پس؟

+ فعلاً نمی‌خواد کاری کنیم. ولی اگر می‌خواهین بدونین اینو سرچ کنیم:

“evn Python tutorial”

همچنین اگر snyk رو هم اجرا کنیم، همچین مشکلی رو بهمون می‌گه:

```
Testing main.py ...
```

```
X [Low] Use of Hardcoded Credentials
```

```
Path: main.py, line 3
```

```
Info: Do not hardcode credentials in code. Found hardcoded  
credential used in a condition.
```

```
X [Medium] Use of Hardcoded Credentials
```

```
Path: main.py, line 3
```

```
Info: Do not hardcode passwords in code. Found hardcoded  
password used in a condition.
```

گفته دو تا مشکل پیدا کردم یکی اینکه هم `username` (جزء اطلاعات حقوقی)، توی کد هست که این `impact` و تأثیر امنیتش پایینه (`low`) و هم رمز هست که این یکی ریسک و `impact` اش `medium` هست.

مثال:

فرض کنیم من یه کدی نوشتم که از یه لایبرری استفاده کردم:

```
import Crypto.Cipher.AES as AES
```

خب بیایم `bandit` رو اجرا کنیم:

```

Test results:
>> Issue: [B413:blacklist] The pyCrypto library and its module
Crypto.Cipher.AES are no longer actively maintained and have been
deprecated. Consider using pyca/cryptography library.
    Severity: High    Confidence: High
    CWE: CWE-327 (https://cwe.mitre.org/data/definitions/327.html)
    More Info:
https://bandit.readthedocs.io/en/1.7.6/blacklists/blacklist_imports
.html#b413-import-pycrypto
    Location: main.py:1:0
1      import Crypto.Cipher.AES as AES
-----

Code scanned:
    Total lines of code: 1
    Total lines skipped (#nosec): 0

Run metrics:
    Total issues (by severity):
        Undefined: 0
        Low: 0
        Medium: 0
        High: 1
    Total issues (by confidence):
        Undefined: 0
        Low: 0
        Medium: 0
        High: 1
Files skipped (0):

```

خب می‌گه یه مشکل امنیتی پیدا کردم! که ریسکش هم «High» هست. (یعنی ریسکش خیلی بالاست و خیلی بده!)

می‌گه تو از لایبرری «pyCrypto» استفاده کردی، اما این لایبرری خیلی وقته که بروزرسانی نمیشه و قدیمیه و نباید دیگه استفاده بشه! اگر یه نگاهی هم به وبسایتش خود pyCrypto بندازیم، می‌بینیم که خود نویسندگان هم نوشتن که دیگه ازش استفاده نکنین!^{۱۰۱} دیگه در حال توسعه و بروزرسانی نیست و مشکلات امنیتی داره.

“PyCrypto 2.x is unmaintained, obsolete, and contains security vulnerabilities.

Please choose one of the following alternatives:

Cryptography

- Recommended for new applications.

PyCryptodome

- Recommended for existing software that depends on PyCrypto.”

خب بیایم از همون لایبرری اولی استفاده کنیم:

¹⁰¹ <https://www.pycrypto.org/>


```
import cryptography
```

دوباره bandit رو اجرا کنیم:

```
Test results:
  No issues identified.

Code scanned:
  Total lines of code: 1
  Total lines skipped (#nosec): 0

Run metrics:
  Total issues (by severity):
    Undefined: 0
    Low: 0
    Medium: 0
    High: 0
  Total issues (by confidence):
    Undefined: 0
    Low: 0
    Medium: 0
    High: 0
Files skipped (0):
```

می بینین؟ مشکل رفع شد!
این مشکل رو snyk پیدا نکرد.

مثال:

```
file_name_inp = input("Enter the file name: ")
file_name = file_name_inp + ".txt"
with open(file_name, "r") as f:
    lines = f.readlines()
    for line in lines:
        print(line)
```

اینجا bandit مشکلی پیدا نکرد ولی «snyk» می‌گه که:

```
x [Medium] Path Traversal
Path: main.py, line 3
Info: Unsanitized input from user input flows into open, where
it is used as a path. This may result in a Path Traversal
vulnerability and allow an attacker to read arbitrary files.
```

می‌گه تو نام یه فایل رو از کاربر خواستی. بعدش اومدی محتویات فایل رو پرینت کردی. اما وایسا ببینیم! بررسی نکردی که اسم فایل یه درستی رو داده یا نه! ممکنه کاربر اسم یه فایل دیگه توی سیستم رو داده باشه و عملاً بتونه اطلاعات یه فایل یه که نباید بهش دسترسی پیدا می‌کرد رو بخونه! این خیلی بده!

پس باید کنترل می‌کردی که اسم فایل رو درست داده یا نه.

مثال:

حالا بیایم یه کد دیگه بنویسیم:

```
def is_password_valid(pass1, pass2):  
    return pass1 == pass2
```

این رو با «mypy» و «pylint» و «pylance» و «pytype» و «pyre» و همچنین با اونایی که مخصوص بررسی مشکل امنیتی هستن مثل «bandit» و «snyk» چک کردم و هیچکدوم مشکل امنیتی متوجه نشدن. پس فقط تکیه به ابزارها نمی‌تونه کافی باشه. خوبن ولی کافی نیستن. خوبه خود برنامه‌نویس کتاب یا داکيومنت‌هایی درباره اینکه مشکلات امنیتی رایج چی هستن و چطور کد امن بنویسیم بخونن.^{۱۰۲}

- خب مشکل امنیتی چی بود؟

Timing Attack

یادتونه علامت «==» توی پایتون (یا زبونای دیگه مثل جاوا) چیکار انجام میداد؟ میومد چک می‌کرد آیا دو تا چیزی که دادیم عیناً با هم برابرن؟ مثلاً:

```
'abcd' == 'bbbb'
```

درواقع پایتون میاد از اولین کرکتر چپ و راست شروع می‌کنه. می‌گه اولین کرکتر متن اول، آیا با اولین کرکتر متن دوم برابره یا نه؟ اگر برابر نبود، همون اول می‌گه خب پس لابد متنا یکسان نیستن! پس همون موقع می‌گه که دو متن برابر نیستن. (دیگه بقیش رو نگاهم نمی‌کنه. چون می‌گه برای چی نگاه کنم؟ وقتی یه کرکتر برابر نیست پس برابر نیستن دیگه! الکی چک کنم که سرعت برنامه کم شه؟!)

حالا اگر کرکتهای اولی دو متن برابر بودن چی؟

```
'abcd' == 'aaaa'
```

می‌گه خب اولی برابر بود. میره کرکتر دوم رو چک می‌کنه ببینه آیا دومیا چی برابرن یا نه؟^{۱۰۳}

^{۱۰۲} می‌تونین تو اینترنت سرچ کنین «common python code vulnerabilities»
^{۱۰۳} درواقع کدش اینه:

```
int isPassValid(char pass[], char inp_pass[])  
{  
    if (len(pass) != len(inp_pass))  
        return 0;  
  
    for (int i = 0; i < len(pass) && i < len(inp_pass); i++)  
    {  
        if (pass[i] != inp_pass[i])  
            return 0;  
    }  
}
```

حالا فرض کنیم پسورد سایت **استرینگ** '۱۲۳۴۵۶۷۸۹' هست. خب من می‌خوام پسورد رو حدس بزنم. چیکار می‌کنم؟

به ترتیب موارد زیر رو امتحان می‌کنم:

'000000000', '111111111', '222222222', ...

و همینطور ادامه میدم. خب شاید بگین برای چی؟

اگر کد رو نگاه کنیم، می‌بینیم که دونه‌دونه روی کرکتر حرکت می‌کنه و اولین چیزی mismatch و چیزی که یکسان نباشه رو ببینه، می‌گه یکسان نیستن..

حالا مشکل چیه؟ مشکل اینه که من مثلاً می‌بینم مثلاً ۱ میلی‌ثانیه طول کشیده و بعدش می‌بینم که ۱۱۱۱۱۱۱۱، مثلاً ۲ میلی‌ثانیه طول کشیده و بقیه چیزا یعنی ۲۲۲۲۲۲۲۲ و... هم همگی ۱ میلی‌ثانیه طول کشیدن. پس من می‌گم عه چرا بیشتر طول کشید؟ بذار فکر کنم. همم قاعدتاً کرکتر اول درست بوده و مجبور شده زمان بیشتری رو طرف کنه که کرکتر دوم رو هم چک کنه. خب درواقع کرکتر اول پسورد شما پیدا شد. خب با ۱۰ تا تست می‌تونم کرکتر اول رو حدس بزنم. برای هر کرکتر ۱۰ تا حدس نیازه. پس برای یه رمز ۱۰ کرکتری، ۱۰*۱۰ تا حدس نیازه. یعنی صرفاً ۱۰۰ تا حدس. اما در حالت عادی که بخوایم همه حالات رو تست کنیم 10^{10} حالت نیازه. یعنی به شدت کارمون ساده شد.

به این نوع حملات می‌گن «Timing Attack». یعنی از **تفاوت** زمان مصرفی، بتونم اطلاعاتی از سیستم بگیرم.

درواقع دقیق‌تر بگیم؛ وقتی صحبت از انجام کار و یا بررسی روی داده‌های حساس و پنهون میشه، تفاوت زمانی بین ورودی‌های مختلف، می‌تونه یه سری اطلاعات از اون داده حساسی که یه جای کد استفاده شده (بیشتر برای بررسی و چک یا انجام یه سری کار بر حسب اون داده حساس)، پی برد.

- خب پس چیکار کنیم؟

+ یادتونه اسم این نوع حملات چی بود؟ حملات «Timing Attack» یا حملات مبتنی بر **تفاوت** زمان. خب پس برای رفع مشکل، ما نباید تفاوت زمانی داشته باشیم! یعنی مهم نباشه که کرکترای اول دوتاشون یکسان یا نه! فارغ از اون، کرکتر رو تا ته چک کنه و نگه اولی یکسان نبود پس قطع می‌کنم و می‌گم یکسان نیست. بلکه بره تا تهش و وقتی تا ته رفت و توی تعداد چک‌ها تفاوتی رخ نداد (هردو برنامه به خاطر اجرا شدن تعداد دستور یکسان، یک زمان صرف کردن)، بعدش بگه یکسان نبودن. درواقع شما به عنوان یه مهندس امنیت کدها رو مطالعه می‌کنی و میگی سازمان محترم، اینجا کدت مشکل داره. بیا اینطور بنویس که امن باشه:

```
bool isPassValid(char pass[], char inp_pass[],
                 size_t lenPass, size_t lenInp)
{
    return 1;
}
```

```

volatile bool isValid = true;
isValid &= (lenPass == lenInp);
for (size_t i = 0; i < lenPass && i < lenInp; i++)
{
    isValid &= (pass[i] == inp_pass[i]);
}
return isValid;
}

```

اینجا دیگه ما در هر صورت، چه طول یکسان باشه چه نباشه، چه اولین کرکتر mismatch بخوره چه آخرین، یه زمان طی میشه.^{۱۰۴}

هر بار `isValid` رو `and` می‌کنم با چیزی که می‌خوام. پس اگر mismatch باشه، `isValid` مقدارش `false` میشه ولی بازم روند ادامه پیدا می‌کنه و روند قطع نمیشه. (که زمان یکسانی صرف شه)

- اگر به صورت رندوم تاخیر اضافه کنیم که متوجه نشن دقیق چه زمانی طی شده، مشکل حل میشه؟ + سخت می‌کنه ولی ناممکن نمی‌کنه!^{۱۰۵}

درواقع فرض کنیم شما تاخیرهای کاملاً رندوم اضافه کنیم. قاعداً باید یه بازه‌ای انتخاب کنیم. مثلاً بازه (۱۰۰ و ۲۰۰) نانو ثانیه. خب قاعداً هر دفعه به صورت رندوم بین ۱۰۰ تا ۲۰۰ نانو ثانیه یه تأخیر اضافه میشه.

این مشکل رو حل نمی‌کنه صرفاً مشکل‌تر می‌کنه. چون حمله‌کننده باید تعداد بار بیشتری رو تست کنه. مثلاً ۱۰۰۰ بار یه چیزی ثابت رو تست کنه و بعد نمودار بکشه و بفهمه که به طور میانگین ۱۵۰ نانو ثانیه تأخیر اضافه شده.^{۱۰۶} حالا با نمودار آنالیز رو پیش می‌بره. هردفعه این عملیات رو با کرکترای مختلف تست می‌کنه و در نهایت هی دونه‌دونه کرکتر رو پیدا می‌کنه.

سایت زیر توضیح خیلی خوبی داده و راه‌حل‌هایی هم پیشنهاد داده. حتماً بخونیدش:

<https://www.chosenplaintext.ca/articles/beginners-guide-constant-time-cryptography.html>

یکی از دلیلی استفاده از اسمبلی برای رمزنگاری اینه که ما می‌دونیم instruction ها هرکدوم آیا constant time هستن یا نه؟ چقدر clock cycle طول می‌کشه تا اون instruction اجرا شه؟ درواقع با اسمبلی، ما دسترسی خوبی روی سخت‌افزار داریم.

نمونه‌های دیگه‌ای از Timing Attack (پیشرفته):

^{۱۰۴} البته این صد درصد constant time نیست!

105 i) <https://stackoverflow.com/questions/28395665/could-a-random-sleep-prevent-timing-attacks>

ii) Remote Timing Attacks are Practical: <https://crypto.stanford.edu/~dabo/papers/ssl-timing.pdf>

iii) https://fahrplan.events.ccc.de/congress/2012/Fahrplan/attachments/2235_29c3-schinzel.pdf

106 Law of Large Numbers: https://en.wikipedia.org/wiki/Law_of_large_numbers

+ Timing Attacks on WhatsApp, Signal, and Threema can Reveal User Location¹⁰⁷ → Paper: Hope of Delivery: Extracting User Locations From Mobile Instant Messengers¹⁰⁸

+ Security Best Practices for Side Channel Resistance (*Highly recommended!*)¹⁰⁹

توجه مهم!

- خب چرا تمام ابزارهای موجود توی اینترنت رو دانلود نکنیم تا کدمون رو باهاشون اسکن کنه؟
+ آیا شما هر برنامه‌ای رو روی سیستمتون نصب می‌کنین؟ از کجا معلوم همشون خوب باشن و مخرب نباشن؟ (که بعضیاشون اتفاقاً هستن!)
پس شما باید صرفاً از اونایی که معتبرن استفاده کنین.
+ همچنین بعضی از این ابزارها، ممکنه کد شما رو به cloud ارسال کنن و اونجا آنالیز کنن. حواستون باشه بعضی وقتا نیازه که کد خیلی شخصی یا کد سازمانی که مشخص شده رو ندین بهشون. معمولاً میگن که به چه چیزایی دسترسی دارن. مثلاً توی توضیحات اکستنشن «Red Hat Dependency Analytics» نوشته که:

The Red Hat Dependency Analytics extension is an online service hosted and maintained by Red Hat. Red Hat Dependency Analytics only accesses your manifest files to analyze your application dependencies before displaying the vulnerability report.

این ابزارها کافی نیستن! درواقع پلتفرم‌های بهتر (و البته پولی) هستن که خیلی بهتر مشکلات کدتون رو میگن و مشکلات امنیتی بیشتری پیدا می‌کنن. پس مطمئناً سازمان‌ها صرفاً روی Bandit و اینا تمرکز نمی‌کنن.
همچنین گیت‌هاب و گیت‌لب و اینا هم بعضی از این ابزارها رو در خودشون جا دادن.

107 <https://restoreprivacy.com/timing-attacks-on-whatsapp-signal-threema-reveal-user-location/>

108 <https://arxiv.org/abs/2210.10523>

109 <https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/secure-coding/security-best-practices-side-channel-resistance.html>

More reading:

+ snyk python security best practices cheat sheet¹¹⁰

قدرت پایتون به سادگی و موجود بودن لایبرری‌های زیاد 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/Mohammad-Kamal-mk/Books>

¹¹⁰ <https://snyk.io/blog/python-security-best-practices-cheat-sheet/>