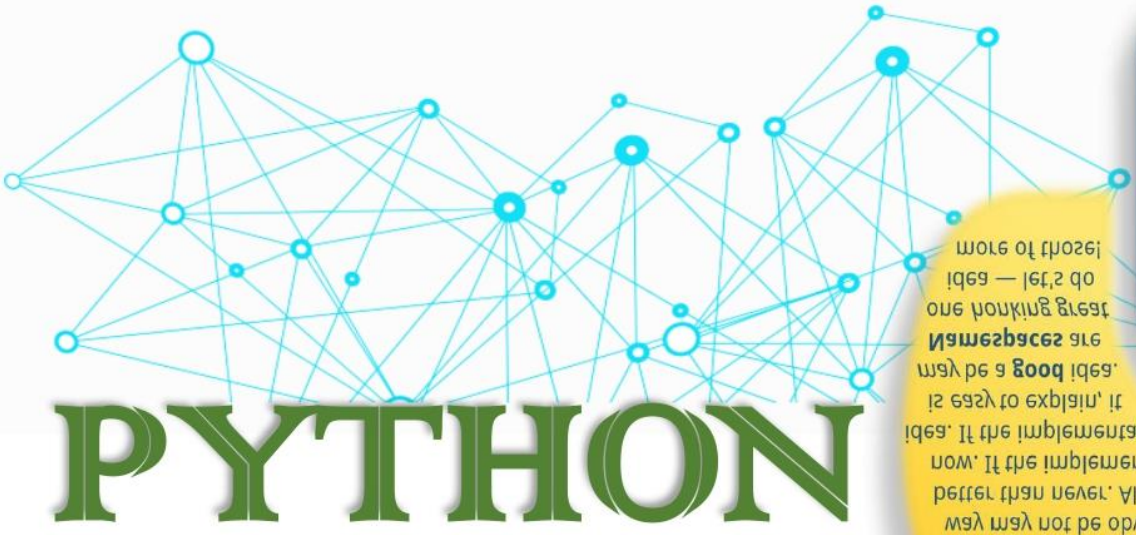

پایتون برای صفر کیلومترها

(مقدمه ای بر زبان برنامه نویسی پایتون)

علی نظری زاده

numbers
string
and / or / not
if / else
for
while
function
set
list

dictionary
tuple
Built-in Functions
all
any
zip
math
Lambda

LinkedIn linkedin.com/in/ali-nazarizadeh

رفقا سلام!! میخوایم یه مروری روی زبان برنامه نویسی پایتون داشته باشیم و بخش های مهم اون رو بررسی کنیم پس بدون هیچ حرف اضافه ای بریم که کار رو شروع کنیم. کدها رو توی محیط ژوپیتر نوتبوک (Jupyter Notebook) می نویسیم که یه محیط باحال برای برنامه نویس های پایتونه. ژوپیتر اینقدر کار راه انداز و مفیده که خیلی ها میگن در حد یه کیس ازدواج برای برنامه نویسای پایتونه که خب ما وارد جزییات نمیشیم. اگر چشم تون ضعیف نباشه، میتونید محیط ژوپیتر نوتبوک رو پایین ببینید که متشکل از تعدادی سلول هستش که توی هر سلول میتونید کدهاتون رو بنویسید و اجرا کنید. حالا این به چه دردی میخوره؟؟ به خیلی دردها که اینجا نمیتونم بگم اما همینو بدونید که تست و خطایابی برنامه ها خیلی ساده تره میشه. ژوپیتر نوتبوک به همراه تعدادی نرم افزار دیگه توی یک بسته نرم افزاری به اسم آناکوندا (Anaconda) قرار دارند که با یه سرچ کوچولو تو اینترنت میتونید نصبش کنید و برید حالشو ببرید.

خب دیگه شوخی بسه بریم سراغ درس، برای این که بخوایم داده های خودمون رو مثل عدد، رشته، متن و ... توی کامپیوتر و پایتون ذخیره کنیم، باید اونها رو داخل متغیرها بریزیم. خیلی از علما میگن که متغیرها مثل ظرف هایی هستند که داده ها رو تو خودشون ذخیره میکنن، ما هم این تعریف رو قبول میکنیم (من جزو علما نیستم!!). مثلا توی سلول اول به ترتیب داده های 4 و 3.5 و "A" و "Hi Friends" داخل متغیرهای a و b و c و d قرار گرفتند که مثلا اگر متغیر a رو توی یه سلول جداگانه بنویسیم و Shift + Enter یا Ctrl + Enter بزنیم، میتونیم مقدارشو یعنی عدد 4 رو ببینیم که همین هم برای ما هم چاپ کرده.

numbers & String

```
a = 4
b = 3.5
c = 'A'
d = "Hi Friends"
```

```
a
```

```
4
```

به کمک تابع type() میتونیم بفهمیم که نوع یک متغیر چیه مثلا type(a) شده int پس میفهمیم که متغیر a که عدد 4 داخلش ذخیره شده یک عدد صحیح است یا مثلا متغیر b که مقدارش 3.5 است، یک عدد اعشاری است اما متغیرهای c و d داخلشون رشته ذخیره شده پس تو خروجی نوشته str یعنی اینکه از نوع رشته هستند. توی سلول آخر هم تمام متغیرها رو توی یه سلول نوشتم و با کاما از هم جدا کردم تا یکجا خروجی شون رو ببینیم.

```
type(a)
```

```
int
```

```
type(b)
```

```
float
```

```
type(c)
```

```
str
```

```
type(d)
```

```
str
```

```
a, b, c, d
```

```
(4, 3.5, 'A', 'Hi Friends')
```

رفقا به کمک دستور print هم میتونیم مقادیر رو چاپ کنیم که نمونه های مختلف شو توی کدهای زیر میتونید ببینید.

```
print(a)
print(b)
print(c)
print(d)
```

```
4
3.5
A
Hi Friends
```

```
print("a: ", a)
print("b: ", b)
print("c: ", c)
print("d: ", d)
```

```
a:  4
b:  3.5
c:  A
d:  Hi Friends
```

به کمک "اف استرینگ ها" که توی صفحه بعد کدشو میبینید، میتونیم ترکیبی از دستورات و رشته ها رو داشته باشیم. به طور خلاصه هرچی داخل آکولاد ({}) باشه، محاسبه و مقدارش حساب میشه اما هرچی که بیرون از آکولاد باشد، عینا همون چاپ میشه.

```
print(f"a + b: {a + b}")
print(f"a - b: {a - b}")
print(f"9 * 6: {9 * 6}")
print(f"8 / 2: {8 / 2}")
```

```
a + b: 7.5
a - b: 0.5
9 * 6: 54
8 / 2: 4.0
```

یه وقت هایی لازمه که مقادیر موجود داخل متغیرها رو باهم مقایسه کنیم که این کار به کمک عملگرهای مقایسه ای امکان پذیره اما دقت کن که خروجی این عملگر ها True (درست) یا False (غلط) است. توی خط اول سلول زیر مشخصه که عدد 2 با عدد 3 برابر نیست پس خروجی شده False یا غلط. (پس میفهمیم که 2 با 3 برابر نیست). خط بعدی هم خب 2 مخالف 3 هستش پس خروجی شده True. و خط های بعدی هم واضحه دیگه، هر خطی که درست و صحیح بود، خروجی میشه True و هر خط یا دستوری که اشتباه بود، خروجی میشه False.

```
print(2 == 3)
print(2 != 3)
print(4 > 4)
print(4 >= 4)
print(8 < 2)
print(3 <= 9)
```

== : برابر بودن
=! : مخالف بودن
> : بزرگتر بودن
=> : بزرگتر مساوی
< : کوچکتر بودن
=< : کوچکتر مساوی

```
False
True
False
True
False
True
```

به کمک عملگر and می توان روی داده ها شرط گذاشت که خروجی این عملگر True یا False است. به صورت خلاصه زمانی خروجی عملگر and منجر به True میشه که همه شرط ها درست باشند یعنی اگر صدتا شرط داشتیم و 99 تاشون درست بود اما یکی غلط یا False بود، خروجی and میشه False. خب چه وقت خروجی میشه True ؟ الان گفتم دیگه، زمانی که همه شرط ها True باشند یعنی هر صدتا درست باشند.

and

```
print(f"False and False ==> {False and False}")
print(f"False and True  ==> {False and True}")
print(f"True and False  ==> {True and False}")
print(f"True and True   ==> {True and True}")
```

```
False and False ==> False
False and True  ==> False
True and False  ==> False
True and True   ==> True
```

```
(2 == 2) and (4 < 5)  # T and T
```

```
True
```

```
(2 == 2) and (4 < 5) and ( 3 != 3 )  # T and T and F
```

```
False
```

توی سلول اول از کد پایین دو تا شرط داریم که هر دو درست یا True هستند، میگی نه خب برسی کنیم. شرط اول که (2 == 2) است خب درسته و شرط دوم هم که (4 < 5) هستش اونم درسته و چون هر دو شرط درسته، پس خروجی and هم True یا درسته. توی سلول پایین اش سه تا شرط داریم که دو شرط اول True و آخری False هستش پس طبق تعریفی که برای and داشتیم، خروجی میشه False. خب گلم تا اینجا رو متوجه شدی؟ آفرین بزن بریم عملگر بعدی.

```
(2 == 2) and (4 < 5)  # T and T
```

```
True
```

```
(2 == 2) and (4 < 5) and ( 3 != 3 )  # T and T and F
```

```
False
```

عملگر or هم یه جورایی تو مایه های عملگر and هست و خروجی اش True و False هستش. ببینید بچه ها، خروجی عملگر or زمانی میشه True که حداقل یکی از شرط ها True یا درست باشه. پس اگر صد تا شرط داشته باشیم و 99 تا اون ها False باشن اما یکی True باشه، خروجی میشه True. توی سلول اول صفحه بعد، این عملگر رو مستقیم روی True و False ها اعمال کردم ولی سلول بعدی روی دو شرط (3 == 2) و (2 < 5) اعمال شده که چون یکی شون True هست پس خروجی میشه True. حالا خودت سلول های دیگه رو هم ببین.

or

```
print(f"False or False ==> {False or False}")
print(f"False or True  ==> {False or True}")
print(f"True or False  ==> {True or False}")
print(f"True or True   ==> {True or True}")
```

```
False or False ==> False
False or True  ==> True
True or False  ==> True
True or True   ==> True
```

```
(2 == 3) or (5 >= 2)  # F or T
```

```
True
```

```
(6 < 4) or (2 > 3)  # F or F
```

```
False
```

```
x = (2 * 3) > 5        # T
y = (3 * 8) == (8 * 3) # T
z = 2 == (16 / 2)      # F
```

```
x or y or z  # T or T or F
```

```
True
```

یه عملگر کوچولو مونده به اسم عملگر not که کاری که انجام میده اینه که شرط های True رو به False و شرط های False رو به True تبدیل میکنه. توی سلول های زیر میتونید باهاش آشنا بشید اما زیاد طولش ندید بیاید مبحث بعدی.

not

```
print(f"not False ==> {not False}")
print(f"not True  ==> {not True}")
```

```
not False ==> True
not True  ==> False
```

```
a = 2 == 3
a
```

```
False
```

```
not a # not(2==3)
```

```
True
```

```
not(a)
```

```
True
```

خب بریم سراغ رشته ها یا String که تو پایتون مبحث مهمیه. یک رشته شامل تعدادی کلمه است که هر کلمه هم خودش شامل تعدادی کاراکتر یا حروف الفباست. حالا این رشته رو میتونیم داخل دابل کوتیشن ← " " بزاریم تا پایتون هم بفهمه که این یک رشته یا String هست. مثلا "Hi googolia chetorid" یک رشته است که داخل دابل کوتیشن (" ") قرار دادیم. این رشته از تعدادی کاراکتر تشکیل شده و هر کارکتر هم دارای اندیسی است که اندیس ها از صفر شروع میشن. در سلول اول یک رشته داخل متغیر str1 قرار داده شده که به به کمک تابع len() می تونیم طول رشته رو بدست بیاریم. در سلول های بعدی از براکت ([ ]) استفاده کردیم جهت انتخاب بخشی از رشته. مثلا اونجایی که نوشته str1[0 : 13] یعنی کاراکترهای 0 تا 12 (13 کاراکتر یا حرف الفبا) انتخاب شده یعنی بخش "Data Analysis" که در خروجی هم همین نوشته شده. یا با دستور str2[32 :] می تونیم از کاراکتری که توی اندیس 32 تا آخر رشته هستش رو انتخاب کنیم.

string

```
str1 = "Data Analysis with Python using Numpy, Pandas and Matplotlib"
type(str1)
```

```
str
```

```
len(str1)
```

```
60
```

```
str1[0:13]
```

```
'Data Analysis'
```

```
str1[32:]
```

```
'Numpy, Pandas and Matplotlib'
```

```
str1[19:25]
```

```
'Python'
```

حالا که با رشته ها آشنا شدیم، بهتره که بدونید کلی تابع باحال و کاربردی برای کار باهاشون وجود داره که مهم ترین هاشون رو توی کدهای صفحه بعد میتونی ببینی. قبلش میخوام یه نکته خیلی خفن بهت بگم که هم خیر آخرت توشه و هم خیر این دنیا. اوه دمت گرم داش علی حالا اون نکته چیه؟ فعلا زیاد خودمونی نشو بیا صفحه بعد بهت میگم.

اگر شما یه رشته ایجاد کردی، یه ساختمان داده ساختی یا کلا از هر لایبرری و ابزای استفاده کردی، برای اینکه بدونی چه توابعی برای کار باهاش وجود داره، کافیه که اسمشو بنویسی بعد نقطه و در آخر نیز کلید tab کیبورد رو بزنی تا لیست کل توابعی که براش وجود داره رو ببینی. مثلا اگر بعد از همین رشته str1 نقطه و کلید tab بزنی به همه توابعی که برای کار با رشته ها در پایتون وجود داره دسترسی داری و میتونی تک تک اونا رو تست کنی.

به کمک تابع islower() می تونی بفهمی که آیا همه کاراکترهای یک رشته با حروف کوچک نوشته شده اند یا نه. در این مثال جواب خیر است و خروجی شده False. به کمک تابع isupper() هم میتونی چک کنی که آیا همه کاراکترهای یک رشته با حروف بزرگ نوشته شده اند یا نه که باز هم برای این مثال جواب False است. خیلی راحت هم به کمک دو تابع lower() و upper() می تونیم همه کاراکترهای یک رشته رو به حروف کوچیک و بزرگ تبدیل کنیم.

```
str1.islower()
```

```
False
```

```
str1.isupper()
```

```
False
```

```
str1.lower()
```

```
'data analysis with python using numpy, pandas and matplotlib'
```

```
str1
```

```
'Data Analysis with Python using Numpy, Pandas and Matplotlib'
```

```
str2 = str1.lower()
str2
```

```
'data analysis with python using numpy, pandas and matplotlib'
```

```
str2.islower()
```

```
True
```

به کمک تابع find() که تو صفحه بعد آوردم، میتونیم یک کلمه رو تو متن جستجو کنیم که ببینیم وجود داره یا نه. اینجا من کلمه "python" رو جستجو کردم که توی خرجی عدد 19 رو بهم داده، حالا این 19 یعنی چی؟ یعنی این که کلمه "python" داخل متن هست و اندیس شروعش هم اندیس شماره 19 هستش. با تابع replace() می تونیم یه مقداری رو توی متن با مقداری که مدنظر خودمون هست جایگزین کنیم که اینجا من بهش گفتم هرجا " data analysis" دیدی اونو با کلمه "python" جایگزین کن که خروجی هم همینو به ما داده.

```
str2.find('python')
```

```
19
```

```
S1 = "Hi My Friends"
S2 = "this is data analysis course"
```

```
S1 + S2
```

```
'Hi My Friendsthis is data analysis course'
```

```
S3 = S1 + " " + S2
S3
```

```
'Hi My Friends this is data analysis course'
```

```
S3 = S3.replace('data analysis', 'python')
S3
```

```
'Hi My Friends this is python course'
```

یکی از اون توابع خیلی باحال که عاشقشم (امیدوارم به دوس ... م بر نخوره، عهه من که اصلا ... ندارم.) تابع split() هستش حالا بگذریم. به کمک این تابع می تونیم رشته ها رو تیکه تیکه کنیم. به صورت پیش فرض روی "اسپیس" یا همون خط فاصله تنظیم شده پس رشته رو بر اساس خط فاصله میشکنه. اگه گفتی خروجی چی میشه؟ آفرین! خروجی میشه جداسازی کلمات یک متن. پس یکی از راه های ساده برای بدست آورن کلمات یک متن، استفاده از تابع split() هستش. یا مثلا توی سلول های پایینی اومدم رشته رو بر اساس کلمه "python" در واقع split() کردم که اینجوری زیر رشته قبل پایتون تو یه بخش و زیر رشته بعد پایتون تو بخش دیگه قرار میگیره.

```
S3.split()
```

```
['Hi', 'My', 'Friends', 'this', 'is', 'python', 'course']
```

```
S3.split()[1]
```

```
'My'
```

```
S3.split('python')
```

```
['Hi My Friends this is ', ' course']
```

```
S3.split('python')[0]
```

```
'Hi My Friends this is '
```

خیلی وقت ها برای حل یک مسئله لازم است که شرط هایی را در برنامه لحاظ کنیم و برای تک تک شرط ها نیز راهکارهایی ارائه بدیم. به کمک دستور if / else می توان این شرط ها را لحاظ کرد که نمونه ای ازش رو توی سلول های پایین میبینید. من میخوام یک شرط بزارم که اگر مقدارهای دو متغییر باهم برابر بودند، تو خروجی "Yes" چاپ بشه و اگر این شرط برقرار نبود یعنی دو عدد با هم برابر نبودند، توی خروجی "No" چاپ بشه. اگر شرط جلوی دستور if که همان a == b یعنی برابر بودن a , b برقرار باشد، برنامه وارد بدنه میشه و دستور print("Yes") چاپ میشه اما در اینجا مقدارهای a , b برابر نیستند در نتیجه برنامه وارد بدنه else میشه و دستور print("No") چاپ میشه که در این مثال هم این اتفاق افتاده.

if / else

```
a = 8
b = 12

if a == b:
    print("Yes")
else:
    print("No")
```

```
No
```

در مثال بالا فقط شرط برابری a , b را بررسی کردیم که اگر شرط درست بود وارد بدنه if و اگر شرط نادرست بود وارد بدنه else میشد. اما خیی اوقات تعداد شرط های بیشتری در نظر میگیریم و برای برقرار بودن هر شرط، یک راهکار یا کدی قرار میدهم. در مثال زیر حاصل جمع دو متغییر a , b را به عنوان شرط در نظر گرفتیم که اگر این حاصل جمع کوچکتر مساوی صفر بود، وارد بدنه if و در خروجی "a+b <= 0" چاپ می شود. اگر این حاصل جمع دو عدد بین صفر و ده بود، تو خروجی "0 <= a+b <= 10" چاپ می شود. اگر این حاصل جمع بین بازه ده و بیست بود، تو خروجی " 10 <= a+b <=20" چاپ می شود و اگر هیچ کدوم از این شرط ها برقرا نبود، در نهایت برنامه وارد بدنه else میشه و " a+b > 20" چاپ میشه.

```
if (a + b) <= 0:
    print("a+b <= 0")
elif (a + b) >= 0 and (a + b) <= 10:
    print("0 <= a+b <= 10")
elif (a + b) > 10 and (a + b) <= 20:
    print("10 < a+b <= 20")
else :
    print("a+b > 20")
```

```
10 < a+b <= 20
```

تو خیلی از مواقع لازم داریم که کاری را به دفعات انجام بدیم یا شمارش و گام هایی را در برنامه لحاظ کنیم. این کار به کمک دستور for امکان پذیره که البته برای اینکه این گام ها یا شمارش ها را لحاظ کنیم، از تابعی به اسم range() استفاده میکنیم. تو مثال پایین و در سلول اول، داخل تابع range() عدد 5 قرار دادم و این بدین معنیه که اعداد 0 تا 4 تولید و سپس حلقه for به این تعداد یعنی 5 بار میشماره. خب حالا نقش اون متغییر i چیه؟ آفرین اون i به تک تک این اعداد اشاره میکنه خب حالا بجوری، بزن بریم. ابتدا i برابر عدد 0 سپس برنامه میاد داخل بدنه for و خروجی متغییر i که همون صفر است چاپ میشه. دفعه بعد i به عدد 1 اشاره میکنه و برابر عدد 1 میشه، دوباره برنامه میاد داخل حلقه for و مقدار i که عدد 1 هستش چاپ میشه و این کار تا آخر یعنی عدد 4 ادامه دارد. حالا دیگه خودت میتونی هر کاری داخل بدنه for انجام بدی که توی سلول آخر اومدیم مقدار i ها رو در عدد 10 ضرب کردیم.

for

```
for i in range(5):
    print(i)
```

```
0
1
2
3
4
```

```
range(5)
```

```
range(0, 5)
```

```
for i in range(5):
    print(i * 10)
```

```
0
10
20
30
40
```

حالا اگر خواستیم گام های دوتا دوتا یا بیشتر برداریم چیکار کنیم؟؟ باشه عزیزیم صبر کن بهت میگم. تابع range() سه مقدار میگیره: (گام حرکت، پایان، شروع)range که توی مثال زیر شروع برابر 0، پایان 10 و گام حرکت برابر 2 است. در نتیجه اعداد 0, 2, 4, 6, 8 تولید و i تک تک و به ترتیب به همه این اعداد اشاره می کند و در خروجی چاپ شده. توی سلول پایین تر، حاصل جمع اعداد زوج یک رقمی محاسبه و در متغییر SUM قرار داده شده. حالا بجوری؟ خب واضحه دیگه، اعداد زوج یکی رقمی 0, 2, 4, 6, 8 است که اینم توی تابع range(0, 10, 2) تولید و سپس حاصل جمع داخل SUM ذخیره شده. فهمیدی؟ آره بابا.

```
for i in range(0, 10, 2):
    print(i)
```

```
0
2
4
6
8
```

```
SUM = 0

for i in range(0, 10, 2):
    SUM = SUM + i

SUM
```

```
20
```

توی مثالی که پایین میبینید، ترکیبی از for و if رو ساختیم که اعداد یک تا ده رو شمارش کردیم سپس با دستور if چک میکنیم که عدد فرد هستش یا زوج و تو خروجی میتونید ببنید.

```
for number in range(10):
    if number % 2 == 0:
        print(f"{number} is even")
    elif number % 2 != 0:
        print(f"{number} is odd")
```

```
0 is even
1 is odd
2 is even
3 is odd
4 is even
5 is odd
6 is even
7 is odd
8 is even
9 is odd
```

خب گلهای من تا اینجا که با for آشنا شدید، بهتره که یه مثال دیگه ازش بزنیم تا خوب جا بیفته. توی این مثال اومدیم با ترکیب دو حلقه for، جدول ضرب 0 تا 9 رو ساختیم.

```python
for i in range(10):
    print(f"********** {i}")
    for j in range(10):
        print(f"{i} * {j}: {i*j}")
```

```
********** 0
0 * 0: 0
0 * 1: 0
0 * 2: 0
0 * 3: 0
0 * 4: 0
0 * 5: 0
0 * 6: 0
0 * 7: 0
0 * 8: 0
0 * 9: 0

********** 1
1 * 0: 0
1 * 1: 1
1 * 2: 2
1 * 3: 3
1 * 4: 4
1 * 5: 5
1 * 6: 6
1 * 7: 7
1 * 8: 8
1 * 9: 9

********** 2
2 * 0: 0
2 * 1: 2
2 * 2: 4
2 * 3: 6
2 * 4: 8
2 * 5: 10
2 * 6: 12
2 * 7: 14
2 * 8: 16
2 * 9: 18

********** 3
3 * 0: 0
3 * 1: 3
3 * 2: 6
3 * 3: 9
3 * 4: 12
3 * 5: 15
3 * 6: 18
3 * 7: 21
3 * 8: 24
3 * 9: 27

********** 4
4 * 0: 0
4 * 1: 4
4 * 2: 8
4 * 3: 12
4 * 4: 16
4 * 5: 20
4 * 6: 24
4 * 7: 28
4 * 8: 32
4 * 9: 36

********** 5
5 * 0: 0
5 * 1: 5
5 * 2: 10
5 * 3: 15
5 * 4: 20
5 * 5: 25
5 * 6: 30
5 * 7: 35
5 * 8: 40
5 * 9: 45

********** 6
6 * 0: 0
6 * 1: 6
6 * 2: 12
6 * 3: 18
6 * 4: 24
6 * 5: 30
6 * 6: 36
6 * 7: 42
6 * 8: 48
6 * 9: 54

********** 7
7 * 0: 0
7 * 1: 7
7 * 2: 14
7 * 3: 21
7 * 4: 28
7 * 5: 35
7 * 6: 42
7 * 7: 49
7 * 8: 56
7 * 9: 63

********** 8
8 * 0: 0
8 * 1: 8
8 * 2: 16
8 * 3: 24
8 * 4: 32
8 * 5: 40
8 * 6: 48
8 * 7: 56
8 * 8: 64
8 * 9: 72

********** 9
9 * 0: 0
9 * 1: 9
9 * 2: 18
9 * 3: 27
9 * 4: 36
9 * 5: 45
9 * 6: 54
9 * 7: 63
9 * 8: 72
9 * 9: 81
```

خب تا اینجا با دستورات if / else و حلقه for آشنا شدیم، حالا بریم سراغ دستور کنترلی بعدی به نام while. به کمک این دستور کنترلی می توانیم تا زمانی که شرایطی برقرار بود، یک کار یا مجموعه ای از دستورات را انجام دهیم. در این جا شرط گذاشته ایم که متغیر x که مقدارش عدد 5 است، تا زمانی که بزرگتر از 0 باشد برنامه وارد بدنه while شود، مقدار x را چاپ کند و در نهایت نیز یک واحد از x کم شود. چون هر بار یک واحد از x کم می شود، در نتجه زمانی که مقدار x برابر 1- شود، شرط while برقرار نیست و برنامه از حلقه خارج می شود.

while

```
x = 5
while(x >= 0):
    print(x)
    #x -= 1
    x = x-1
```

```
5
4
3
2
1
0
```

یه مثال دیگه از while ببینیم که با دستور شرطی if / else ترکیب شده است. بزارید اینجا یه نکته مهم دیگه بهتون بگم که باز خیر دنیا و آخرت توش باشه!! آه آن چیست جناب نظری زاده؟؟ درسته گفتم خودمونی نشو اما در این حد هم لازم نیست. خب نکته اینه، برای این که بخوایم یک کاری را بی نهایت بار انجام بدیم و تا آخر دنیا ادامه داشته باشه، از while(True) استفاده می کنیم. حالا خب کی این حلقه به اتمام میرسه و از حلقه while خارج می شویم؟ برای این که از حلقه خارج بشیم، از دستور break استفاده می کنیم. در مثال زیر به کمک تابع input() از کاربر مقداری رو دریافت می کنیم، اگر اون مقدار مخالف 0 بود، عبارت "مقدار x مخالف صفر است" چاپ می شود و بی نهایت بار این کار تکرار می شود. اما زمانی که عدد 0 وارد شود، شرط if برقرار نمی شود و برنامه وارد بدنه else می شود، در آن جا عبارت "مقدار x صفر است" است چاپ می شود و بعدش که break نوشتم باعث می شود که برنامه به اتمام برسد.

```
x = 1
while (True):
    x = int(input("x : "))
    if x != 0:
        print("مخالف صفر است x مقدار")
    else:
        print("صفر است x مقدار")
        break
```

```
x : 6
مخالف صفر است x مقدار
x : 2
مخالف صفر است x مقدار
x : 0
صفر است x مقدار
```

خب بچه های عزیز بریم سراغ مبحث بعدی که خیلی هم مهمه و اسمش تابع یا function هست. اگه بخوام خیلی ساده تعریف تابع رو بگم اینجوریه : تابع یک ساختار، یک قالب یا چارچوبی است که به کمک آن می توانیم مسائل خودمون رو در قالب این ساختار مدل کنیم. دقت داشته باش که یک تابع از سه مولفه اصلی (ورودی، پردازش و خروجی) تشکیل شده است و هنگام تعریف یک تابع، باید این سه بخش رو مشخص کرد. ما توی ریاضیات هم تابع رو داشتیم که اونجا به صورت y = f(x) می نوشتیم که در اینجا y خروجی، f تابع یا همان عملیات پردازشی و x هم ورودی هستش. همون طور که تو سلول اول میبینید، برای ایجاد یک تابع ابتدا باید از کلمه کلیدی def استفاده کنیم، بعد اسم تابع که در اینجا اسم تابع ما fun_name است. خب حالا اون سه بخش اصلی یعنی ورودی، پردازش و خروجی کدوم ها هستند؟ ورودی تابع، Inputs است، خروجی Outputs است که حتما باید با کلمه کلیدی return بازگشت داده شود و عملیات پردازشی یا همان الگوریتم هم در بدنه تابع نوشته می شود. خب حالا با توجه به ورودی و خروجی می توانیم چهار حالت رو برای توابع در نظر بگیرم. 1- ورودی و خروجی نداشته باشد، 2- ورودی و خروجی داشته باشد، 3- ورودی نداشته باشد و خروجی داشته باشد، 4-خروجی داشته باشد و ورودی نداشته باشد. این دیگه چیه سید!!! تو که گفتی ورودی و خرجی حتما باید باشه اما تو این چهار موارد، تابع میتونه ورودی یا خروجی نگیره. اولا من سید نیستم، دوما آره درسته اما اون تو ریاضیات بود ولی توی برنامه نویسی ما میتونیم توابع رو بر اساس این چهار دسته ای که گفتم ایجاد کنیم. نکته آخر و تکمیلی هم بگم اینکه تو مرحله اول باید تابع رو بسازیم و تو مرحله دوم میتونم از تابعی که ایجاد کردیم بینهایت بار در برنامه استفاده کنیم. مثلا یک تابع برای محاسبه مساحت مستطیل میسازیم سپس هر تعداد بار که بخواهیم می تونیم ازش استفاده کنیم.

function

```
def fun_name(Inputs):
    # algorithm
    return Outputs
```

```
y = f(x)

y = خروجی
f = تابع / عملیات پردازشی
x = ورودی
```

```
Input  Output
False  False
True   False
False  True
True   True
```

```
مرحله اول :    ساختن تابع
مرحله دوم : استفاده از تابع
```

داخل سلول زیر تابعی به اسم F1 ایجاد کردیم که تنها کاری که انجام میده فقط یک پیغام چاپ میکنه. خب این تابع ورودی ندارد چرا؟ چون خط اول داخل پرانتز چیزی نوشته نشده و همچنین خروجی هم ندارد چرا؟ چون اصلا return ای وجود ندارد. پس F1 یک تابع بدون ورودی و خروجی است. تا اینجا مرحله اول انجام شد یعنی تابع رو ساختیم.

```
def F1():
    print("Hello Python")
```

حالا باید از تابع استفاده کنیم دیگه نه؟ یعنی همون مرحله دوم که برای این کار کافیه فقط اسم تابع رو بدون کلمه کلیدی def بنویسیم که خروجی "Hello Python" حاصل می شود.

```
F1()
```

```
Hello Python
```

داخل سلول اول تابعی به اسم F2 ایجاد کردیم حالا بهم بگید که وضعیت ورودی و خروجی های این تابع به چه صورته؟ آفرین درسته این تابع یک ورودی به نام x دارد اما خروجی ندارد. تنها کاری که انجام میده اینه که ورودی که براش فرستاده میشه رو تو خروجی چاپ میکنه همین. این حالا مرحله اوله که تابع رو ساختیم. سلول بعدی همون مرحله دومه که ما از تابع استفاده میکنیم. چون تابع ورودی دارد در نتیجه اینجا باید یک مقدار براش بفرستیم که در اینجا عدد 80 به عنوان ورودی فرستاده شده.

```
def F2(x):
    print(x)
```

```
F2(80)
```

```
80
```

در زیر تابعی به اسم F3 ایجاد شده که دارای ورودی است چرا؟ چون مقدار x داخل پرانتز است و همان ورودی تابع است. تابع خروجی ندارد چرا؟ چون دستور return ای وجود ندارد. حالا رسالت این تابع چیه؟ هیچی فقط باید عددی که به عنوان ورودی میگیره رو چک کنه که اگر کوچکتر از 0 بود چاپ کنه "منفی" و اگر بزرگتر از صفر بود چاپ کنه "مثبت". تو سلول بعدی هم از تابع استفاده شده است و عدد 3 به عنوان ورودی براش در نظر گرفته شده است.

```
def F3(x):
    if x < 0:
        print('منفی')
    else:
        print('مثبت')
```

```
F3(3)
```

```
مثبت
```

تابع زیر ورودی ندارد اما خروجی دارد چون کلمه کلیدی return وجود دارد. این تابع تا همیشه مقدار 2 * 8 یعنی 16 رو بازگشت می دهد. حالا که تابع رو ساختیم مرحله دوم باید ازش استفاده کنیم، چون تابع خروجی دارد در نتیجه می تونیم مقدارشو توی یه متغییر ذخیره کنیم که در سلول زیر داخل متغیر x ذخیره کردیم.

```
def F4():
    return 8 * 2
```

```
F4()
```

```
16
```

```
x = F4()
x
```

```
16
```

توی سلول زیر اومدیم روی تابع F4() شرط گذاشتیم حالا این یعنی چی؟ ببین عزیزم، بالا تابع F4() رو ساختیم اینجا توی شرط if گذاشتم و ازش استفاده میکنیم، خروجی اش میشه چی؟ خب میشه 16 دیگه حالا مثل این میمونه که داخل شرط داریم چک میکنیم که آیا 16 با 16 برابر است. خب بله برابره و خروجی True حاصل میشه.

```
if F4() == 16:
    print("True")
else:
    print("False")
```

```
True
```

تابع F5 هم ورودی داره هم خروجی، ورودی speed و خروجی x است که با return بازگشت داده شده. نحوه کارش ساده است حالا تو مرحله بعد از این تابع استفاده میکنیم و عدد 140 رو براش میفرستیم. توی سلول بعد عدد 110 رو براش میفرستیم با این تفاوت که چون تابع خرجی داره، مقدار خروجی اش رو میتونیم داخل متغیری مثل x ذخیره کنیم.

```
def F5(speed):
    if speed < 80:
        x = "Slow"
    elif speed >= 80 and speed < 120:
        x = "Normal"
    elif speed >= 120:
        x = "Fast"

    return x
```

```
F5(140)
```

```
'Fast'
```

```
S = F5(110)
S
```

```
'Normal'
```

تابع F6 نیز دقیقا مثل تابع F5 هستش فقط با این تفاوت که توی بدنه if ها، مقدارها با دستور return بازگشت داده شده است اما توی تابع f5 مقادیر داخل x ذخیره شده و در نهایت یک بار بازگشت داده شده است.

```
def F6(speed):
    if speed < 80:
        return "Slow"
    elif speed >= 80 and speed < 120:
        return "Normal"
    elif speed >= 120:
        return "Fast"
```

```
F6(115)
```

```
'Normal'
```

توی کد زیر به کمک تابع input از ورودی عددی دریافت میکنیم و سپس داخل تابع int قرار میدهیم تا مقدار ورودی به عدد صحیح تبدیل شود و در نهایت داخل متغییر S قرار می گیرد. حالا S به عنوان ورودی برای تابع F6 ارسال شده است و این تابع نیز یک خرجی میدهد حال بر اساس خروجی این تابع، دستور if اعمال می شود. در مثال زیر عدد 50 به عنوان ورودی به تابع F6 داده شده است که خروجی نهایی پیغام "رانندگی شما آهسته و با احتیاط است" چاپ شده است.

```
S = int(input("Speed : "))

if F6(S) == 'Slow':
    print("رانندگی شما آهسته و با احتیاط است")
elif F6(S) == 'Normal':
    print("رانندگی شما معمولی و با دقت است")
elif F6(S) == 'Fast':
    print("رانندکی شما پر خطر است")
```

```
Speed : 50
رانندگی شما آهسته و با احتیاط است
```

توابع می تونند چندین مقدار به عنوان ورودی بگیرند مثل تابع F7 که دو مقدار a , b رو میگیره و برابر بودن شون رو چک میکنه که دیگه واضحه و نیازی به توضیح نیست.

```
def F7(a, b):
    if a == b:
        return f"{a} = {b}"
    elif a > b:
        return f"{a} > {b}"
    elif a < b:
        return f"{a} < {b}"
```

```
F7(5, 8)
```

```
'5 < 8'
```

خب خب خب بریم سراغ بخش بعدی که هم خیلی جذابه و هم خیلی کاربردی، میخوایم در مورد "ساختمان داده ها" یا "داده ساختارها" صحبت کنیم. ببینید دوستان زمانی که ما تعداد زیادی داده، عدد، متن و ... داشته باشیم دیگه نمیتونیم داخل یه متغییر ذخیره شون کنیم بلکه باید داخل یک ساختار یا قالب استاندارد ذخیره کنیم که بتونیم روی این داده ها پردازش های مناسبی انجام بدیم. در واقع یک ساختمان داده یک قالب یا ساختاری است که می تواند تعداد زیادی داده را در خودش ذخیره کند. حالا یکی از این ساختمان داده ها، ساختمان داده "set" یا "مجموعه" می باشد که تعاریفش همان تعاریف مجموعه ها در ریاضیات است. دوستان در ساختمان داده set، عناصر تکراری قرار نمیگیرند مثلا اگر هزار تا عدد 3 داشته باشیم، فقط یک بار در مجموعه نوشته می شود. و نکته دوم اینکه ترتیب در مجموعه ها بی معنی است در نتیجه ترتیب قرار گیری داده ها همواره ممکنه تغییر کنه. به دو صورت می توان یک داده ساختار set ایجاد کرد: 1- قرار دادن داده ها داخل براکت ({}) 2- استفاده از تابع set و فرستادن داده ها برای آن. در مثال پایین S1 یک مجموعه یا set است که تایپ اش هم داره میگه اره حاجی این یه مجموعه است.

set

```
S1 = {2, 8, 6, 5, 6, 5, 9, 4}  # S1 = set( {2, 8, 6, 5, 6, 5, 9, 4} )
S1
```

```
{2, 4, 5, 6, 8, 9}
```

```
type(S1)
```

```
set
```

حالا هر یک از ساختمان داده ها دارای تعدادی توابع هستند که به کمک اون توابع میتونیم عملیات های پردازشی مختلفی روی داده ها داشته باشیم که در زیر تعدای از مهم ترین توابع مجموعه ها را با هم خواهیم دید. (دقت کن همه توابع با S1. + tab در دسترسن. بنویس S1، بعد نقطه و بعد دکمه tab کیبورد رو بزن). با تابع add میتونیم مقداری رو به مجموعه اضافه کنیم و همچنین با تابع remove میتونیم داده ای درون مجموعه رو حذف کنیم که در این جا برای تابع remove عدد 4 فرستاده شده در نتیجه عدد 4 از مجموعه حذف میشه.

```
S1.add(10)
S1
```

```
{2, 4, 5, 6, 8, 9, 10}
```

```
S1.remove(4)
```

```
S1
```

```
{2, 5, 6, 8, 9, 10}
```

به کمک تابع union می تونیم اجتماع و به کمک تابع intersection می تونیم اشتراک دو مجموعه رو بگیریم. اگه خیلی ساده بخوام بگم، اجتماع دو مجموعه یعنی همه داده های دو مجموعه روی هم ریخته میشه اما اشتراک دو مجموعه یعنی فقط عضوهایی که بین هر دو مشترک هستند انتخاب میشه. مثلا سلول آخر که اشتراک گرفتیم، حاصل شده اعداد 2 و 6 حالا چرا ؟ خب چون اعدد 2 و 6، هم توی S1 هستند هم توی S2.

```
S2 = {4, 13, 12, 2, 6, 7}
```

```
S1.union(S2)
```

```
{2, 4, 5, 6, 7, 8, 9, 10, 12, 13}
```

```
S1.intersection(S2)
```

```
{2, 6}
```

با تابع difference می تونیم تفاضل دو مجموعه رو بگیریم. S1.difference(S2) یعنی داده هایی که تو S1 هستند اما تو S2 نیستند یا مشابه شو میتونیم بنویسیم : S1 – S2. و همچنین S2.difference(S1) یعنی داده هایی که تو S2 هستند اما تو S1 نیستند. باز مشابه شو می تونیم S2 – S1 بنویسیم.

```
S1.difference(S2)  # S1 - S2
```

```
{5, 8, 9, 10}
```

```
S2.difference(S1)  # S2 - S1
```

```
{4, 7, 12, 13}
```

```
S1
```

```
{9, 10}
```

با تابع issubset می توان بررسی کرد که آیا یک مجموعه، زیر مجموعه دیگری است یا نه.

```
S3 = set({9, 6})
S4 = set({9, 6, 20})
```

```
S3.issubset(S1)
```

```
True
```

```
S4.issubset(S1)
```

```
False
```

```
S1
```

```
{2, 4, 5, 6, 8, 9}
```

خب دوستان شما میتونید به کمک حلقه ها، مجموعه ها رو پیماشین کنید حالا چجوری ؟ خب ساده است دیگه، تو مثال های پایین، متغییر i که داخل for هستش، به تک تک داده های داخل مجموعه S1 اشاره میکنه و اینجوری میشه تک تک داده ها رو پردازش کرد و هر بلایی سرشون آورد. توی اون سلول آخر هم با تابع clear()، مجموعه S1 رو پاک کردم.

```
for i in S1:
    print(i)
```

```
2
4
5
6
8
9
```

```
for i in S1:
    if i%2 == 0:
        print(f"{i} is even")
    else:
        print(f"{i} is odd")
```

```
2 is even
4 is even
5 is odd
6 is even
8 is even
9 is odd
```

```
S1.clear()
S1
```

```
set()
```

خب بزن بریم سراغ ساختمان داده بعدی یعنی لیست (list) که اینجا دیگه ترتیب قرار گیری داده ها مهمه و میتونیم هر چقدر دلمون خواست داده تکرای قرار بدیم. لیست ها رو به دو صورت میشه ایجاد کرد: 1- قرار دادن داده ها داخل براکت ([ ]) 2- قرار دادن داده ها داخل تابع list.

list

```
L1 = [5, 8, 5, 6, 9, 8, 4]  # L1 = list( [5, 8, 5, 6, 9, 8, 4] )
L1
```

```
[5, 8, 5, 6, 9, 8, 4]
```

```
type(L1)
```

```
list
```

حالا که فهمیدیم چجوری میشه لیست ها رو ایجاد کرد، میتونیم بخشی از داده های داخلش رو بکشیم بیرون و یقه شون رو بگیریم!! توی کدهای صفحه بعد این کار رو به کمک براکت ([ ]) و عملگر دو نقطه (:) انجام دادیم. قبل دو نقطه اندیس شروع و بعد دو نقطه اندیس پایان رو مینویسیم و اینجوری داده های بین این بازه استخراج میشن.

```
L1[0]
```

```
5
```

```
L1[5:]
```

```
[8, 4]
```

```
L1[2:6]
```

```
[5, 6, 9, 8]
```

با تابع append() میشه هر نوع داده ای رو به لیست اضافه کرد. به سلول دوم نگاه کنید، من خواستم اعداد 20 و 30 رو با تابع append به لیست اضافه کنم یعنی عدد 20 در اندیس 8 و عدد 30 در اندیس 9 اما این اتفاق نیفتاده و لیست [20, 30] در اندیس 8 قرار گرفته. حالا راه حل چیه ؟ ساده است، استفاده از تابع extend که سول پایین مثال شو زدم. به کمک این تابع میتونیم هر تعداد داده که خواستیم در اندیس های جداگانه داخل لیست قرار بدیم.

```
L1.append(20)
L1
```

```
[5, 8, 5, 6, 9, 8, 4, 20]
```

```
L1.append([20, 30])
L1
```

```
[5, 8, 5, 6, 9, 8, 4, 20, [20, 30]]
```

```
L1[8]
```

```
[20, 30]
```

```
L1[8][0]
```

```
20
```

```
L1.extend([20, 30, 40])
L1
```

```
[5, 8, 5, 6, 9, 8, 4, 20, [20, 30], 20, 30, 40]
```

تابع copy() : کپی گرفتن از یک لیست.

تابع count() : شمارش تعداد داده های داخل لیست.

تابع index() : پیدا کردن اندیس داده ای خاص. مثلا اینجا اندیس عدد 5، صفر هستش.

تابع insert() : قرار دادن مقداری در اندیسی خاص. در اینجا عدد 100 در اندیس 1 قرار داده شده است.

```
L1_copy = L1.copy()
L1_copy
```

```
[5, 8, 5, 6, 9, 8, 4, 20, [20, 30], 20, 30, 40]
```

```
L1.count(8)
```

```
2
```

```
L1.index(5)
```

```
0
```

```
L1.insert(1, 100)
L1
```

```
[5, 100, 8, 5, 6, 9, 8, 4, 20, [20, 30], 20, 30, 40]
```

با تابع ()pop می تونیم داده ای رو از لیست پاپ یا حذف کنیم. اگر به صورت پیش فرض هیچ عددی برای این تابع نفرستیم، آخرین مقدار لیست حذف می شود. اما می تونیم اندیس داده ای که میخوایم حذف بشه رو براش بفرستیم طبق مثال زیر که بهش گفتم عددی که تو اندیس 5 هسش رو حذف کن، یعنی عدد 9.

به کمک تابع ()remove هم می تونیم داده ای رو از لیست حذف کنیم فقط با این تفاوت که این بار به جای اندیس، خود عددی که میخوایم از لیست حذف بشه رو برای این تابع میفرستیم. وایسا ببینم اینجا که برای این تابع عدد 5 رو فرستادی، پس چرا 5 ها رو حذف نکرده ؟ سرکاریم ؟؟ نه عزیزم سرکار چرا. به اولین 5 که برسه حذف میکنه با بقیه شون کاری نداره.

```
[5, 100, 8, 5, 6, 9, 8, 4, 20, [20, 30], 20, 30, 40]
```

```
L1.pop()
```

```
40
```

```
L1
```

```
[5, 100, 8, 5, 6, 9, 8, 4, 20, [20, 30], 20, 30]
```

```
L1.pop(5)
```

```
9
```

```
L1
```

```
[5, 100, 8, 5, 6, 8, 4, 20, [20, 30], 20, 30]
```

```
L1.remove(5)
L1
```

```
[100, 8, 5, 6, 8, 4, 20, [20, 30], 20, 30]
```

تابع revers() : با این تابع، داده ها از آخر به اول نوشته میشن.

تابع sort() : مرتب شدن داده ها از کوچک به بزرگ.

تابع clear() : خالی کردن یک لیست.

```
L1.reverse()
L1
```

```
[30, 20, [20, 30], 20, 4, 8, 6, 5, 8, 100]
```

```
L1.remove([20, 30])
```

```
L1
```

```
[30, 20, 20, 4, 8, 6, 5, 8, 100]
```

```
L1.sort()
L1
```

```
[4, 5, 6, 8, 8, 20, 20, 30, 100]
```

```

```

```
L1.clear()
L1
```

```
[]
```

تو سلول زیر اومدم اعداد 0 تا 9 رو به کمک تابع range تولید کردم، حلقه for روشون زدم و با تابع append توی یک لیست ذخیره کردم. میخوام بگم که یه جور دیگه هم میشه این کار رو کرد یعنی کلا for رو تو یه خط نوشت مثل سلول آخر. پس کافیه for رو توی براکت یا همون لیست بنویسد، مقداری که قراره تو لیست باشه یعنی همون i رو قبل for بنویسید تا اینجوری خیلی ساده بتونید لیست ها رو مقدار دهی کنید.

```
L2 = list()
for i in range(10):
    L2.append(i)

L2
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
L2.clear()
L2
```

```
[]
```

```
L2 = [i for i in range(10)]
L2
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

تو سلول زیر روی حلقه for شرط if رو هم اعمال کردم تا اعداد بزرگتر مساوی 5 رو ذخیره کنم. روش دوم رو توی سلول آخر نوشتم یعنی کلا توی یه خط. پس دقت کن وقتی یه for و یه if داشتیم، اول باید for رو بنویسم بعد if رو جلوش حالا اون مقداری که میخوایم تو لیست ذخیره بشه رو سمت چپ مینویسم که اینجا مقدار ما همین i هستش.

```
L3 = list()
for i in range(10):
    if i >= 5:
        L3.append(i)

L3
```

```
[5, 6, 7, 8, 9]
```

```
L3.clear()
L3
```

```
[]
```

```
L3 = [i for i in range(10) if i >= 5]
L3
```

```
[5, 6, 7, 8, 9]
```

تو مثال پایین، از for و if و else استفاده شده که روش دوم نوشتن این قطعه کد رو سلول آخر آوردم براتون که حالشو ببرید. زمانی که مثل الان سه مولفه داشتیم (یعنی for و if و else)، سمت راست for و سپس میایم سمت چپ if / else می نویسم دقیقا مثل قطعه کد زیر.

```
L4 = list()
for i in range(10):
    if i >= 5:
        L4.append(i)
    else:
        L4.append(0)

L4
```

```
[0, 0, 0, 0, 0, 5, 6, 7, 8, 9]
```

```
L4.clear()
L4
```

```
[]
```

```
L4 = [i if i >= 5 else 0 for i in range(10)]
L4
```

```
[0, 0, 0, 0, 0, 5, 6, 7, 8, 9]
```

بزار حالا که دور همیم ساختمان داده بعدی یعنی دیکشنری ها رو هم بررسی کنیم که خیلی مهم هستند. تو دیکشنری ها داده ها رو تو value ها ذخیره و برای دستیابی به اون ها key ها رو ایجاد میکنیم. پس از key ها جهت دستیابی به value ها یا همون داده های خودمون استفاده میکنیم. اینجا هم میشه به دو صورت دیکشنری ها رو ایجاد کرد : 1- استفاده از آکولا ({ })، 2- استفاده از تابع dict که روش اول ساده تر و مرسوم تره. توی مثال زیر داده های ما : 'Mohammad', 'Ali', 32, 14 , 28 هستند که key ها یا کلیدهاشون به ترتیب 2 و 4 و 5 و 8 و 9 هستند. مثلا برای دستیابی به داده 'Ali' باید کلیدش یعنی 8 رو صدا بزنم.

dictionary

```
dic = {
    key: value,
}
```

```
d1 = dict( {2:28, 4:14, 5:32, 8:'Ali', 9:'Mohammad'} )
d1
```

```
d1 = {2 : 28,
      4 : 14,
      5 : 32,
      8 : 'Ali',
      9 : 'Mohammad'}
d1
```

```
{2: 28, 4: 14, 5: 32, 8: 'Ali', 9: 'Mohammad'}
```

```
type(d1)
```

```
dict
```

تو کد زیر برای دستیابی به مقدار 14، به دو صورت کلیدش رو صدا زدم یعنی استفاده از براکت و استفاده از تابع .get

```
d1[4]
```

```
14
```

```
d1.get(4)
```

```
14
```

ما می تونیم داده ساختارهای دیگه ای مثل لیست ها رو توی دیکشنری ها ذخیره کنیم مثل مثال پایین که کاملا واضحه. کلیدها تو این دیکشنری، 'Name' و 'Course' و 'Score' و Valueها هم به ترتیب لیست های 'Name' و 'Course' و 'Score' هستند.

```
Name = ['Ali', 'Mohammad', 'Javad', 'Mehran', 'Sina', ]
Course = ['data analysis', 'big data', 'Matlab', 'NLP']
Score = [19, 20, 17, 16, 18]

d2 = {
    'Name' : Name,
    'Course' : Course,
    'Score' : Score
}

d2
```

```
{'Name': ['Ali', 'Mohammad', 'Javad', 'Mehran', 'Sina'],
 'Course': ['data analysis', 'big data', 'Matlab', 'NLP'],
 'Score': [19, 20, 17, 16, 18]}
```

اکثر اوقات دیکشنری ها به همین سادگی و گوگولی نیستند، بلکه کمی پیچیده تر هستند مثل دیکشنری زیر. تنها نکته ای که اینجا وجود دارد، در مورد کلید students هستش که مقدارها یا valueهاش دوباره یک دیکشنری دیگه است !! جل الخالق یعنی چی؟؟ ساده ست، ببینید دیکشنری ها میتونند خیلی خیلی منعطف باشند. اینجا valueهای کلید students یک دکشنری دیگه است که کلیدهاش، Name و Course هستند.

```
d3 = {
    'Name' : 'Ali',
    'Family' : 'Nazarizadeh',
    'Skill' : ['Data', 'Python', 'AI', 'NLP'],
    'City' : ['Tehran', 'Kermanshah'],
    'students' : {
        'Name' : ['Zohreh', 'Fatemeh', 'Mohammad', 'Ali'],
        'Course' : ['Python', 'data analysis']
    }
}

d3
```

```
{'Name': 'Ali',
 'Family': 'Nazarizadeh',
 'Skill': ['Data', 'Python', 'AI', 'NLP'],
 'City': ['Tehran', 'Kermanshah'],
 'students': {'Name': ['Zohreh', 'Fatemeh', 'Mohammad', 'Ali'],
  'Course': ['Python', 'data analysis']}}
```

ما گفتیم که داده های درون یک دیکشنری به صورت کلید مقدار ذخیره میشن حالا چجوری میتونیم به این کلید مقدار ها دسترسی داشته باشیم؟ به کمک توابع keys() و values() می تونیم به ترتیب کلیدها و مقدارهای یک دیکشنری رو استخراج کنیم. یه تابع باحال دیگه به اسم items() وجود داره که خروجی این تابع، هم کلید ها و هم مقدارهای

یک دیکشنری هستش !! من اومدم رو این تابع for زدم و چون دو خروجی داره، توی for دو متغیر key و value در نظر گرفتم تا به سادگی بتونم یکجا به کلید مقدارهای دیکشنری d3 دسترسی داشته باشم.

```
d3.keys()
```

```
dict_keys(['Name', 'Family', 'Skill', 'City', 'students'])
```

```
d3.values()
```

```
dict_values(['Ali', 'Nazarizadeh', ['Data', 'Python', 'AI', 'NLP'], ['Tehran', 'Kermanshah'],
{'Name': ['Zohreh', 'Fatemeh', 'Mohammad', 'Ali'], 'Course': ['Python', 'data analysis']}])
```

```
d3.items()
```

```
dict_items([('Name', 'Ali'), ('Family', 'Nazarizadeh'), ('Skill', ['Data', 'Python', 'AI', 'NL
P']), ('City', ['Tehran', 'Kermanshah']), ('students', {'Name': ['Zohreh', 'Fatemeh', 'Mohamma
d', 'Ali'], 'Course': ['Python', 'data analysis']})])
```

```
for key, value in d3.items():
    print(key, ':', value)
```

```
Name : Ali
Family : Nazarizadeh
Skill : ['Data', 'Python', 'AI', 'NLP']
City : ['Tehran', 'Kermanshah']
students : {'Name': ['Zohreh', 'Fatemeh', 'Mohammad', 'Ali'], 'Course': ['Python', 'data analysi
s']}
```

توی سلول های زیر، کلیدهای 'Name' و 'Skill' رو تو براکت نوشتم تا بتونم به مقدارها یا valueهاش دسترسی داشته باشم.

```
d3['Name']
```

```
'Ali'
```

```
d3['Skill']
```

```
['Data', 'Python', 'AI', 'NLP']
```

```
d3['students']
```

```
{'Name': ['Zohreh', 'Fatemeh', 'Mohammad', 'Ali'],
 'Course': ['Python', 'data analysis']}
```

حالا مثلا اگر بخوام به 'Fatemeh' خانم دسترسی داشته باشم چجوریاس!؟ کافیه که مرحله به مرحله فیلتر کنم و برم داخل. یعنی اول کلید students رو انتخاب کنم تا به valueهاش برسم، بعد برم سراغ کلید Name ها و در اخر هم اندیس 'Fatemeh' خانم که 1 هست رو بنویسم. ما میتونیم داده ها درون یک دیکشنری رو تغییر بدیم، مثل مثال پایین که به جای data analysis مقدار ML رو قرار دادم.

```
d3['students']['Name'][1]
```

```
'Fatemeh'
```

```
d3['students']['Course'][0]
```

```
'Python'
```

```
d3['students']['Course'][1] = 'ML'
d3
```

```
{'Name': 'Ali',
 'Family': 'Nazarizadeh',
 'Skill': ['Data', 'Python', 'AI', 'NLP'],
 'City': ['Tehran', 'Kermanshah'],
 'students': {'Name': ['Zohreh', 'Fatemeh', 'Mohammad', 'Ali'],
  'Course': ['Python', 'ML']}}
```

به کمک عملگر in می تونیم مقداری رو توی کلید مقدارها جستجو کنیم. سلول اول گفتم که آیا Ali داخل d3 هست؟ خروجی False برگردونده اما خب چرا مگه ما Ali رو تو دیکشنری نداریم ؟ آره داریم اما اینجا به صورت پیش فرض جستجو روی کلیدها انجام میشه نه روی value یا مقدارها. برای این که به پایتون بفهمونیم که میخوایم توی value ها برامون جستجو کنه، کافیه که بنویسم d3.values().

```
'Ali' in d3
```

```
False
```

```
'Ali' in d3.keys()
```

```
False
```

```
'Ali' in d3.values()
```

```
True
```

```
'Ali' in d3['Name']
```

```
True
```

```
'Zohreh' in d3.values()
```

```
False
```

```
'Zohreh' in d3['students']['Name']
```

```
True
```

داده ساختار آخر هم بگیم که اسمش تاپل یا tuple هست البته سر نامش بین ملت دعواست، یه عده میگن تاپل، یکی دیگه میگه تیوپل و یه سری افراد دیگه هم میگن توپل. خلاصه که بین علما دعواست اما شما بگید تاپل شاید بهتر باشه. تاپل ها زیاد منعطف نیستند و نمی تونیم داده های داخل شون رو تغییر بدیم یا حتی مقداری بهشون اضافه کنیم. وقتی یه تاپل رو ایجاد میکنیم، تا آخر عمر همونجوریه نه مقدار میگیره و نه مقدارهاش تغییر میکنه !! ما رو مسخره کردی حالا این بی خاصیت به چه دردی میخوره ؟ مودب باش بهت میگم. توی خیلی از مواقع که داده های ما اصلا تغییر نمی کنند از این ساختمان داده استفاده می کنیم. مواردی مثل کد ملی، موقعیت های جغرافیایی و ... که همیشه ثابت هستند. تاپل ها رو هم میشه به دو صورت ایجاد کرد : 1- استفاده از پرانتز () 2- استفاده از تابع tuple. جالبه بدونید که برای تاپل ها فقط دو تابع وجود داره که پایین توضیح دادم.

تابع () count : تعداد تکرار یه داده رو برای ما مشخص میکنه. مثلا توی تاپل T2، دو تا عدد 2 وجود داره.

تابع () index : میشه با این تابع، اندیس داده ای رو بدست آورد. مثلا داده 'S' تو اندیس 1 هستش.

Tuple

```
T1 = (2, 8, 6, 9)  # T1 = tuple( [2, 8, 6, 9] )
T1
```

```
(2, 8, 6, 9)
```

```
type(T1)
```

```
tuple
```

```
T2 = (2, 'S', "Ali", [2, 3, 'Mohammad'], 2, 8)
T2
```

```
(2, 'S', 'Ali', [2, 3, 'Mohammad'], 2, 8)
```

```
T2.count(2)
```

```
2
```

```
T2.index(2)
```

```
0
```

```
T2.index('S')
```

```
1
```

به کمک براکت ([ ]) و مشخص کردن اندیس ها، میشه داده ها رو استخراج کرد.

```
T2[0]
```

```
2
```

```
T2[3]
```

```
[2, 3, 'Mohammad']
```

```
T2[3][2]
```

```
'Mohammad'
```

```
8 in T2
```

```
True
```

تو این بخش میخوایم درباره توابع داخلی پایتون یا Built-in Functions صحبت کنیم که خیلی کار راه انداز هستند. جریان از این قراره که پایتون اومده تعدادی تابع که هر کدوم کار خاصی رو انجام میدن برای ما ساخته که بتونیم از اون ها تو بخش های مختلفی استفاده کنیم. این توابع انعطاف پذیری بالایی دارن و میتونیم هر نوع داده ای با هر نوع ساختاری که دارد رو به عنوان ورودی براش بفرستیم. تعدای از مهم ترین توابع داخلی پایتون رو اینجا براتون آوردم که میتونید ببینید.

Built-in Functions

```
number = [7, 6, 12, 4, 3, 9, 18]
text = "data analysis with python using numpy, pandas and matplotlib"

print(f"min : {min(number)}")
print(f"min : {min(text)}")
print(f"max : {max(number)}")
print(f"max : {max(text)}")
print(f"sum : {sum(number)}")
print(f"abs : {abs(-20)}")
print(f"len : {len(number)}")
print(f"len : {len(text)}")
print(f"pow : {pow(2, 4)}")
print(f"round : {round(8.265, 2)}")
print(f"int: {int(4.5)}")
print(f"float: {float(3)}")
print(f"type : {type(6)}")
print(f"type : {type(number)}")
print(f"range : {range(4, 20, 2)}")
print(f"reversed : {list(reversed(number))}")
print(f"sorted: {sorted(number)}")
del(number[1])
print(f"del : {number}")
```

خروجی سلول بالا در زیر نشان داده شده:

```
min : 3
min :
max : 18
max : y
sum : 59
abs : 20
len : 7
len : 60
pow : 16
round : 8.27
int: 4
float: 3.0
type : <class 'int'>
type : <class 'list'>
range : range(4, 20, 2)
reversed : [18, 9, 3, 4, 12, 6, 7]
sorted: [3, 4, 6, 7, 9, 12, 18]
del : [7, 12, 4, 3, 9, 18]
```

تابع ()min : کوچک ترین مقدار رو بر میگردونه.

تابع ()max : بزرگ ترین مقدار رو بر میگردونه.

تا بع ()sum : حاصل جمع کل داده ها رو حساب میکنه.

تابع ()abs : عملیات قدر مطلق که تو ریاضیات هست رو انجام میده.

تابع ()len : طول داده هایی که داخل اون داده ساختار هستش رو حساب میکنه.

تابع ()pow : عملیات توان رو انجام میده. (عدد اول رو به توان عدد دوم میرسونه)

تابع ()round : عدد رو گرد میکنه.

تابع ()int : عدد رو تبدیل به عدد صحیح یا Int میکنه.

تابع ()float : عدد رو تبدیل به عدد اعشاری یا float میکنه.

تابع ()type : نوع داده رو مشخص میکنه.

تابع ()range : از عدد 4 تا 20، با گام دو تا دو تا عدد تولید میکنه.

تابع ()reversed : داده ها رو از آخر به اول مینویسه.

تابع sorted() : داده ها رو مرتب میکنه.

تابع del() : داده ساختار رو حذف میکنه.

بیاید در مورد دو تابع مهم به نام های عمه all و خاله any صحبت کنیم که خروجی این دوتا، درست و غلط یا True و False هستش. عمه all میگه زمانی خروجی من میشه True که همه شرط هایی که برام میفرستی، True باشه. اگر 999 تا شرط True وجود داشت اما یکی False بود، کار خراب میشه و خروجی عمه all میشه False. (یعنی رفتار all سختگیرانه است، دقیقا مثل عمه ها !!!).

all

```
L1 = [True, True, True, True]

all(L1)
```

```
True
```

```
L2 = [True, True, True, False]

all(L2)
```

```
False
```

حالا مثال عددی هم بزنیم، توی سلول زیر روی لیست L3 حلقه for زدم و اونایی که بزرگتر از 10 هستند میشه True و بقیه False. در نتیجه لیستی از True و False ها تولید میشه و من این لیست رو به عنوان ورودی برای تابع all فرستادم که خروجی all شده False، حالا چرا ؟؟ ساده ست من خواستم ببینم آیا همه داده هایی که توی لیست L3 هستش، بزرگتر مساوی 10 هستند یا نه ؟ خب برای این کار اول for میزنم و به کمک number >= 10 این True و False ها رو مشخص میکنم، حالا مرحله بعد باید از تابع all استفاده کنم چرا all ؟ چون دارم میگم "همه اعداد" پس باید یاد عمه all بیفتم که سخت گیره. خروجی نهایی شده False. بنده خدا راست هم میگه، همه اعداد داخل L3 بزرگتر مساوی 10 نیستند.

```
L3 = [2, 8, 6, 7, 9]

L4 = [number >= 10 for number in L3]
L4
```

```
[False, False, False, False, False]
```

```
all(L4)
```

```
False
```

من میتونستم مثال بالا رو با تغییر number <=10 در یک خط و به صورت زیر بنویسم که میبینیم خروجی True شده.

```
all([number <= 10 for number in L3])
```

```
True
```

آخرین مثال از عمه all رو ببینیم که بررسی کردم همه اعدادی که بین 0 تا 9 هستند آیا باقی مانده شون بر 2 میشه صفر یعنی همه شون زوج هستند؟ خب این درست نیست و جواب شده False.

```
all(i %2 ==0 for i in range(10))
```

```
False
```

بریم سراغ تابع بعدی یعنی خاله any که خیلی مهربونه. خاله any میگه اگر فقط یه دونه از شرط ها True باشه، خروجی منم میشه True. یعنی اگر هزار تا شرط داشته باشیم 999 تا False باشه اما فقط یکی شون True باشه، خروجی میشه True. (دقیقا مثل خاله ها که مهربونن)

any

```
L1 = [True, True, True, True]

any(L1)
```

```
True
```

```
L2 = [False, False, False]

any(L2)
```

```
False
```

```
L3 = [False, True, False, False]

any(L3)
```

```
True
```

خب بریم ببینیم که مثال عدد به چه صورته، تو سلول اول روی لیست L4 حلقه for زدم و بررسی کردم که کدوم ها بزرگتر مساوی 20 هستند که تعدادی True و False برای من تولید میشه که من دیگه چاپ شون نکردم و سریع فرستادمشون برای خاله any. خب چون توی داده های L4 حداقل یه دونه عدد پیدا میشه که بزرگتر مساوی 20 باشه، در نتیجه خروجی شده True. (یعنی حداقل یا لا اقل یه دونه عدد توی L4 وجود داره که بزرگتر مساوی 20 باشه)

تو سلول دوم بررسی کردم که آیا حداقل یا لا اقل یه دونه عدد تو L4 هست که بزرگتر مساوی 50 باشه، خب واضحه که نیست پس خروجی شده False.

سلول آخر هم بررسی کردم که آیا حداقل یا لا اقل یه دونه عدد بین بازه 0 تا 9 وجود داره که زوج باشه؟ خب واضحه که وجود داره و خروجی هم شده True.

```
L4 = [12, 17, 32, 14, 20, 26]

any([number >= 20 for number in L4])
```

```
True
```

```
any([number >= 50 for number in L4])
```

```
False
```

```
any(i %2 == 0 for i in range(10))
```

```
True
```

خب عمه و خاله بازی و اینا بسه بریم سراغ مبحث بعدی یعنی تابع ()zip که مهمه. بزارید خیلی ساده بهتون بگم، به کمک این تابع میتونیم داده ها رو جفت جفت یا هر چندتا مدنظرمون بود رو کنار هم بزاریم. من اینجا دو لیست L1 و L2 دارم که برای تابع zip اونا رو فرستادم حالا چه اتفاقی میفته ؟ داده اول از L1 یعنی 5 با داده اول از L2 یعنی صفر کنار هم قرار میگیرند، داده دوم از L1 یعنی 9 با داده دوم از L2 یعنی 7 کنار هم قرار میگیرند و این روال ادامه پیدا میکنه. حالا اون تابع list که قبل تابع zip نوشتم چی میگه ؟؟ هیچی نمیگه بنده خدا فقط خروجی رو دوباره به لیست تدیل میکنه که نوشتنش مهم و باید بنویسیم.

zip

```
L1 = [5, 9, 4, 8, 6]
L2 = [0, 7, 4, 6, 8]

list(zip(L1, L2))
```

```
[(5, 0), (9, 7), (4, 4), (8, 6), (6, 8)]
```

```
list(zip(L2, L1))
```

```
[(0, 5), (7, 9), (4, 4), (6, 8), (8, 6)]
```

حالا برعس این داستان هم می تونیم داشته باشیم، یعنی یه داده ای zip باشه و بخوایم از حالت zip خارجش کنیم که این کار خیلی ساده به کمک همان تابع zip و البته کاراکتر "*" امکان پذیره.

```
L3 = [(5, 4), (8, 0), (3, 9)]

List = list(zip(*L3))
print(List)
type(List)
```

```
[(5, 8, 3), (4, 0, 9)]
list
```

بیاید یه کم درمورد ریاضیات و توابع مهم اش صحبت کنیم. برای این کار لایبرری های زیادی وجود داره که یکی از معروف ترین های اون، لایبرری یا کتابخونه math هستش. حالا این لایبرری یا کتابخونه چیه؟ ببینید دوستان ما برای انجام هر کار سخت و پیچیده ای، لازم نیست که از صفر خودمون کد بزنیم، فقط کافیه که از لایبرری هایی که برای انجام کار ما وجود داره استفاده کنیم. خوشبختانه حضرت عشق یعنی پایتون، بیشترین تعداد کتابخونه بین زبان های برنامه نویسی رو داره و برای انجام هر کاری، کتابخونه یا لایبرری براش وجود داره. پس نکته خیلی خیلی خیلی مهم،، هر کاری خواستید انجام بدید اول تو گوگل سرچ کنید ببینید که چه کتابخونه هایی براش وجود داره و از اونا استفاده کنید. استفاده از کتابخونه ها توی پایتون هم خیلی خیلی ساده ست فقط کافیه طبق دستور زیر عمل کنیم: ا

Import اسم کتابخونه ➔ Import math

Math

```
import math
```

```
math.sin(90)
```

```
0.8939966636005579
```

```
math.cos(0)
```

```
1.0
```

```
math.tan(2)
```

```
-2.185039863261519
```

```
math.e
```

```
2.718281828459045
```

```
math.sqrt(25)
```

```
5.0
```

```
math.fabs(-8)
```

```
8.0
```

```
math.log10(100)
```

```
2.0
```

```
math.log2(32)
```

```
5.0
```

```
math.pi
```

```
3.141592653589793
```

```
math.pow(2, 3)
```

```
8.0
```

```
math.factorial(5)
```

```
120
```

```
math.ceil(3.9)
```

```
4
```

```
math.ceil(3.1)
```

```
4
```

```
from math import pow

pow(2, 3)
```

```
8.0
```

تابع ()sin : محاسبه سینوس.

تابع ()cos : محاسبه کسینوس.

تابع ()tan : محاسبه تانژانت.

تابع e : عدد e در ریاضیات.

تابع ()sqrt : محاسبه جذر.

تابع ()fabs : قدر مطلق.

تابع ()log10 : محاسبه لگاریم در مبنای 10.

تابع ()log2 : محاسبه لگاریتم در مبنای 2.

تابع pi : عدد پی در ریاضیات.

تابع ()pow : محاسبه توان.

تابع ()factorial : محاسبه فاکتوریل.

تابع ()ceil : گرد کردن به سمت بالا.

خب میرسیم به آخرین بخش و میخوایم توابع لامبدا (lambda) رو توضیح بدیم که خیلی خیلی مهم هستن و تو زمینه های مختلفی مخصوصا پردازش زبان طبیعی، تحلیل داده، کار با کتابخونه pandas و ... ازش استفاده میشه. دوستان لامبدا یک تابع بدون نام است که با کلمه کلید lambda تعریف میشه و حتما حتما باید توی یک خط کد اونو پیاده سازی کنیم. قالب کلی این تابع به صورت زیر است :

خروجی = lambda ورودی ها : عملیات پردازشی

توی سلول زیر من اومدم عملیات ساده a به توان 2 رو در قالب تابع معمولی که اسمش F1 است و تابع lambda که خروجی اش F2 است پیاده سازی کردم تا تفاوت ها و شباهت هاشو ببینید. با قالبی که بالاتر از لامبدا گفتم، فکر کنم همه چیز واضح باشه نه ؟ اما خب تا اینجا تابع رو ایجاد کردیم حالا مجبوری اونو فراخوانی کنیم و ازش استفاده

کنیم مگه نگفتیم این تابع بدون نام هست ؟ درسته اما اینجا F2 هم خروجی ماست و هم به عنوان نام این تابع ازش استفاده میکنیم که تو سلول بعدی برای F2 عدد 10 رو فرستادیم و خروجی مناسب رو تولید کرده.

lambda

```
def F1(a):
    return a**2

F2 = lambda a: a**2
```

```
print(F1(10))
print(F2(10))
```

```
100
100
```

تو مثال پایین یه دیکشنری ایجاد کردم که اسامی و نمرات پنج دانشجو رو داخلش ذخیره کردم. میخوام با لامبدا میانگین نمرات این کلاس رو بدست بیارم پس یه ورودی به اسم x تعریف میکنم که این ورودی میخوام value های داخل دیکشنری باشه پس با تابع sum حاصل جمع همه value ها رو میگیرم و تقسیم بر تعداد دانشجوها یعنی همون len(D1) می کنم. پس تا اینجا من تابع لامبدا رو ساختم و خروجی میره تو Result. حالا میتونم از همین Result استفاده کنم و به عنوان ورودی، دیکشنری D1 رو بفرستم.

```
D1 = {
    "Ali": 19,
    "Mogammad": 20,
    "Javad": 18,
    "Fatemeh": 20,
    "Zahra": 19
}

Result1 = lambda x: sum(x.values()) / len(D1)
```

```
Result1(D1)
```

```
19.2
```

یکی دو مثال دیگه باهم بررسی کنیم. تو مثال اول داخل سلول زیر، اومدم زوج بودن رو در قالب لامبدا پیاده سازی کردم. مثال دوم اومدم اعداد 0 تا 9 رو با استفاده از range() تولید و حلقه for رو روشون اعمال کردم. حالا تک تک اعدادی که توسط for پیمایش میشه و اسمش x هست رو به lambda میدم و lambda هم توان 2 شون رو حساب میکنه. به همین سادگی و خوشمزگی.

```
Result2 = lambda x: x % 2 == 0
```

```
Result2(9)
```

```
False
```

```
L2 = [(lambda x: x**2)(x) for x in range(10)]
L2
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

```
################################
```

```
watsapp = '0933 136 7233'
watsapp
```

```
'0933 136 7233'
```

LinkedIn https://www.linkedin.com/in/ali-nazarizadeh/

این دوره جامع رو توی لینک زیر میتونید ببینید :

https://bigdataworld.ir/product/pandas_numpy_matplotlib/

آموزش جامع پایتون هم خواستید اینجا هست :

https://bigdataworld.ir/product/python-programming-machine-learning-base/