



دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)
دانشکده مهندسی صنایع

برنامه ریزی و کنترل موجودی ۱

دکتر بهروز کریمی

تدریس‌یار: پدram پیرو اصفیا



به کارگیری پایتون در حوزه پیشبینی

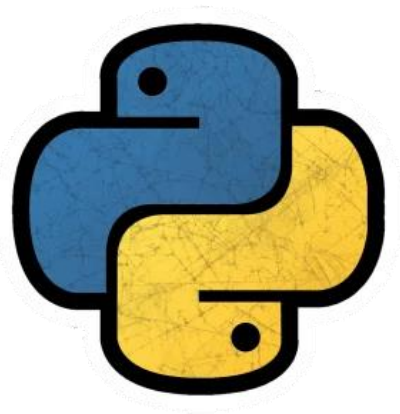
مقدمه ای از پایتون



- این زبان برنامه نویسی توسط Guido van Rossum در اوایل سال ۱۹۹۱ در هلند اختراع شد.
- به علت یکی از برنامه های تلویزیونی کمدی که در سال ۱۹۷۰ پخش میشد (به نام Monty Python's Flying Circus)، اسم این زبان برنامه نویسی، توسط Guido van Rossum، پایتون گذاشته شد.
- این زبان برنامه نویسی، یک زبان برنامه نویسی منبع باز (open source) است. به این معنا که برای عموم مردم باز بوده و قابل استفاده است.
- از همان ابتدا که اختراع شد، با استقبال زیادی مواجه شد و استفاده های خیلی زیادی در حوزه های مختلف از آن شد.
- گوگل از همان ابتدا از پایتون حمایت کرد و از آن استفاده زیادی کرد.

چرا پایتون؟

- یادگیری آن بسیار آسان است، از این بابت که نوع نوشتار این زبان برنامه نویسی، شباهت زیادی با صحبت کردن انسان دارد و دستورات آن ساده است.
- از دیگر مزیت های آن جامعه بزرگی است که دارد، به این صورت که میتوانید سوالات خود را در سایت های مختلفی پیدا کنید، یا حتی آن را بپرسید و بعد از چند ساعت/روز، پاسخ آن را بگیرید.
- مهم نیست که در چه زمینه ای میخواهید فعالیت کنید! پایتون پکیج و کتابخانه مربوط به آن را دارد! حتی میتوانید کتابخانه و پکیج خودتان را بنویسید!



از جمله زمینه هایی که پایتون در آن کاربرد زیادی دارد:

- علوم داده و تحلیل داده
- یادگیری ماشین و هوش مصنوعی
- توسعه نرم افزار
- طراحی وب
- محاسبات مهندسی

نحوه نصب پایتون

- برای نصب این نرم افزار روش های زیادی وجود دارد که در این کورس به متد اصلی آن میپردازیم:
- به سایت <https://www.python.org/downloads/> رفته و آخرین نسخه پایتون را دانلود کنید.
- در هنگام نصب توجه داشته باشید که تیک Add Python to PATH زده شده باشد که بتوانید از طریق command prompt به آن دسترسی داشته باشید.
- در نهایت Install Now را بزنید.



- توجه کنید اگر از قبل، نسخه دیگری از پایتون بر روی کامپیوتر شما نصب شده است و توسط cmd فراخوانی نمیشود (نحوه فراخوانی کردن پایتون توسط cmd در اسلاید بعد وجود دارد) آن را uninstall کنید.

نحوه استفاده از پایتون

- برای استفاده از پایتون، روش های زیادی وجود دارد که از طریق (integrated development environment) IDE های مختلف، این مهم اتفاق می افتد.
- هرچند که این تنها راه نیست. شما میتوانید از پایتون به صورت مستقیم و فراخوانی آن در cmd استفاده کنید. برای اینکار نیاز است که تنها دستور cmd را در کامپیوتر خود سرچ کنید، و سپس python را نوشته و enter بزنید.

Command Prompt - python

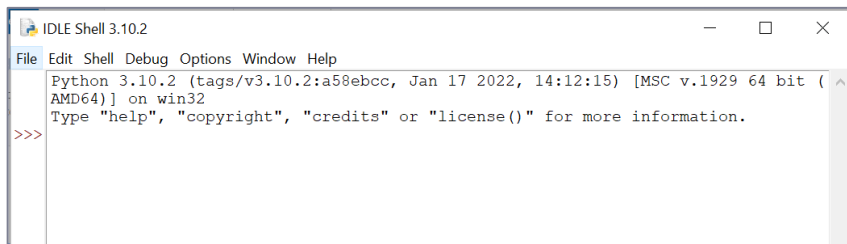
```
Microsoft Windows [Version 10.0.19041.1415]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Mr. Pedram>python
Python 3.10.2 (tags/v3.10.2:a58ebcc, Jan 17 2022, 14:12:15) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
```

- با این کار، پایتون برای شما اجرا شده و میتوانید کد های خود را وارد نمایید و از این زبان استفاده نمایید. هرچند کمتر کسی را پیدا میکنید که از پایتون به این طریق استفاده کند!

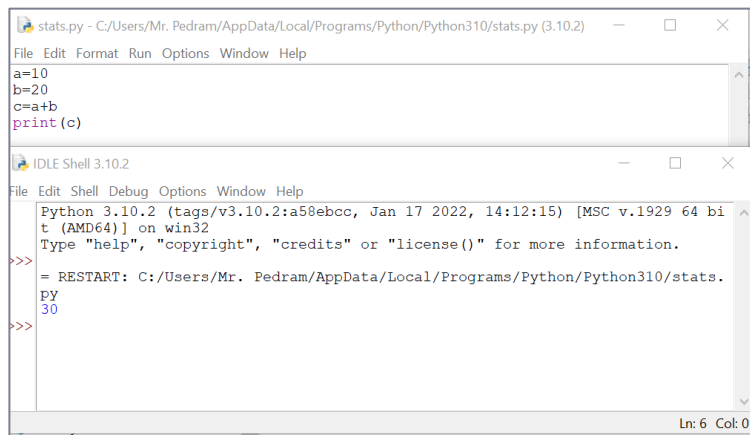
نحوه استفاده از پایتون

- نحوه دیگر استفاده از پایتون، اجرا کردن IDLE خود پایتون میباشد که در این بستر نیز میتوانید به دو صورت کد های خود را اجرا کنید.



- در این بخش، کد هایتان به صورت خط به خط اجرا شده و خروجی را میتوانید بعد از زدن هر `enter` مشاهده کنید.
- برای اجرای پروژه های طولانی، این روش کارآمد نیست و نیاز است که

چندین خط کد نوشته شده و سپس کل برنامه اجرا شود. برای این کار از بخش File، یک New File ایجاد کنید و بعد از `Save` کردن آن، کد های خودتان را آنجا نوشته و در نهایت با زدن `Run Module` آن را اجرا نمایید.



نحوه استفاده از پایتون

- پایتون IDE های زیادی دارد و به ندرت کسی از IDLE خود پایتون استفاده میکند. از جمله IDE های معروف آن:

Visual Studio ○

Spyder ○

Pycharm ○

Vscode ○

Atom ○

Jupyter Notebook ○

- در این کورس، از IDE ژوپیتِر نوتبوک استفاده میشود.

- نحوه نصب Jupyter Notebook:

1. cmd را باز کنید. دستور `pip install jupyter notebook` را وارد کنید. Jupyter notebook در حال نصب شدن است. اگر بعد از نصب اخطار به روز نبودن داد، `python.exe -m pip install --upgrade pip` را نیز اجرا کنید. حال با سرچ کردن jupyter notebook میتوانید آن را اجرا کنید و از مزایای این IDE بهره مند شوید.

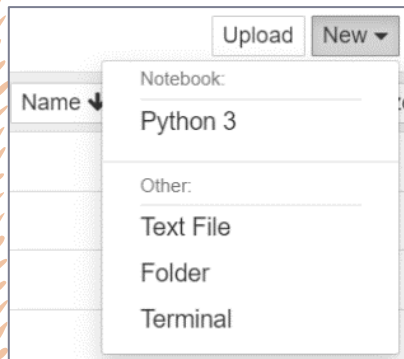
2. نصب Anaconda از طریق لینک <https://www.anaconda.com/products/individual> و اجرا کردن Jupyter Notebook از طریق آن. Anaconda

بستری برای استفاده از IDE های مختلف است که همگی بر روی آن قرار دارند.

نصب کتاب خانه های مختلف در پایتون

- همانطور که گفته شد، کتابخانه های زیادی در پایتون ایجاد شده اند و در اختیار عموم برای استفاده قرار گرفته اند.
- برای نصب این کتاب خانه ها باید دستور `pip install ****` را در `cmd` بنویسید، توجه کنید که به جای ستاره ها باید اسم کتابخانه مد نظر را بنویسید.
- به طور مثال برای نصب کتابخانه `numpy` روی پایتون، `pip install numpy` را در `cmd` بنویسید.

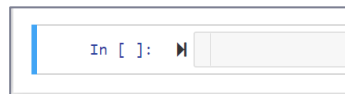
استفاده از Jupyter Notebook



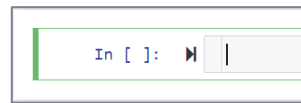
- Jupyter Notebook را در search bar کامپیوتر خود جست و جو کنید و آن را اجرا کنید. اگر نبود در `python -m notebook` jupyter notebook را سرچ کنید. اگر به این نحو نیز باز نشد، `python -m notebook` را در cmd بزنید.
- برای نوشتن کد های خود، از بخش New، Python 3 را انتخاب کنید و میتوانید در این بخش شروع به کد زدن کنید.

- در ژوپیتر نوت بوک، شما سلول های مختلفی در اختیار دارید و میتوانید کد هایتان را در این سلول ها بنویسید و هر سلول را اجرا کنید. توجه کنید که با اجرا کردن هر سلول و رفتن به سلول های بعد، تمام متغیر ها و ... که در سلول های اجرا شده قبل نوشته اید، در memory میماند و میتوانید از متغیر های سلول های پیشین استفاده کنید.

- هر سلول دو حالت edit mode و command mode دارد. به رنگ گوشه سمت چپ توجه کنید.



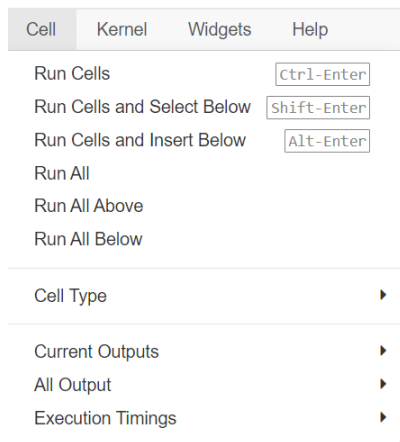
command mode



edit mode

بخش های مختلف Jupyter Notebook

- برای وارد شدن به حالت edit mode میتوانید یا در داخل محل کد زدن کلیک کنید یا enter بزنید.
- برای خارج شدن از حالت edit mode میتوانید esc را بزنید یا اینکه گوشه سمت چپ سلول را کلیک کنید.
- از خوبی های ژوپیتتر نوتبوک این است که میتوانید سلول های متنی ایجاد کنید و نوشته های خودتان را (همانند یک دفترچه یادداشت) بنویسید.

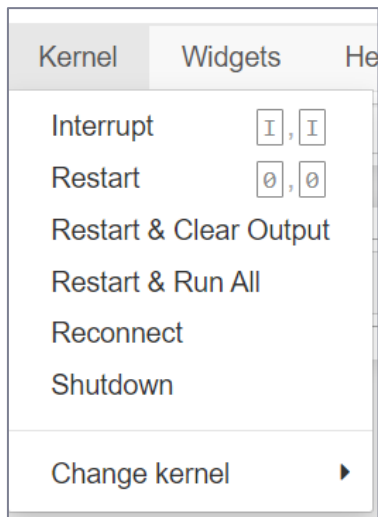
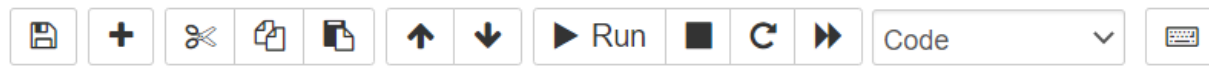


- برای این کار، در بخش Cell میتوانید با انتخاب بخش Cell Type، نوع سلول را به Markdown تغییر دهید و متن خود را بنویسید.
- برای زدن Header برای متن خود (یا بخشی از کد خود) اول از همه نوع سلول را به Markdown تغییر داده و سپس # با یک فاصله استفاده کنید. با افزایش تعداد # ها، header ها کوچک تر میشود.
- برای جابجا شدن بین سلول های مختلف باید در حالت command mode باشید و از up/down arrow keys استفاده کنید.

بخش های مختلف Jupyter Notebook

- برای اجرا کردن کد های خود، روی Run بزنید. همچنین میتوانید این کار را از طریق Cell هم انجام

دهید.



- اگر میخواهید کد خود را از اجرا شدن متوقف کنید، در بخش Kernel، Interrupt را بزنید.
- اگر میخواهید متغیرهایی که قبلا تعریف کرده اید، از روی memory حذف شوند، در بخش Kernel، Restart را بزنید.

کلید های میانبر در استفاده از Jupyter Notebook

- اجرا کردن کد و ماندن در همان سلول:
- Ctrl+Enter
- اجرا کردن کد و رفتن به سلول بعدی:
- Shift+Enter
- توجه کنید برای انجام دستورات زیر باید در حالت command mode باشید. اگر نیستید، Esc بزنید.
- حذف کردن سلول:
- D,D
- عوض کردن تایپ سلول به Markdown:
- M
- ایجاد سلول در زیر سلول فعلی:
- B
- ایجاد سلول بالای سلول فعلی:
- A
- برای دیدن تمامی میانبر ها، در بخش Help، Keyboard shortcuts را بزنید.

دیتاتایپ ها در پایتون

• پایتون از ۵ دیتاتایپ اصلی تشکیل شده است و دیگر دیتاتایپ ها با استفاده از این ۵ دیتاتایپ اصلی ساخته میشوند.

1. bool: Boolean, e.g. True , False
2. int: Integer, e.g. 12 , 12345
3. float: Floating point number, e.g. 3.1415 , 1.1e-5
4. string: Character string, e.g. "This is a string"
5. complex: real+imaginary(j), e.g. 5+3j

• برای دیدن data type یک چیز، از تابع type استفاده میکنیم.

• دیتاتایپ های پیچیده تر و اصلی که کاربرد زیادی دارند و در بستر خود پایتون وجود دارند (دیتاستراکچرها):

1. List
2. Tuple
3. Dictionary
4. Set

در این کورس از آنجایی که بیشتر با لیست ها سر و کار داریم به معرفی آن ها میپردازیم. هرچند در اواخر کورس نیز گذری به ۳ دیتاستراکچر دیگر نیز میزنیم.

لیست (List)

- کالکشنی از دیتاتایپ های مختلف
- همیشه به اعضای آن دسترسی داریم و محدود نیستیم.
- عناصر داخل آن قابل تغییر (mutable) هستند.
- موقعیت اعضای آن از صفر شروع میشود.
- نحوه تعریف آن ها به صورت زیر است که باید اعضای آن را بین دو براکت بنویسیم و آن ها را با , جدا کنیم:

```
lis=[ 1 , 2 , 7 , 54.7333 , 'Inventory' , 'DS' , True,[10,20,30]]
```

0 1 2 3 4 5 6 7

- برای دسترسی به اعضای آن به صورت زیر عمل میکنیم:
- `name_of_the_list[position_of_the_element]`
- مثلا در لیستی که بالا تعریف کردیم، `lis[3]` باید به ما عدد 54.7333 را بدهد.
- درباره لیست ها جلوتر بیشتر صحبت خواهیم کرد.

عملگرها (Operators)

عملگرهای ریاضی:

Opertator	Operation	Example
+	Addition	$5 + 3 = 8$
-	Subtraction	$4 - 3 = 1$
*	Multiplication	$3 * 6 = 18$
/	Division	$20 / 5 = 4$
//	Integer Division	$23 / 6 = 3$
%	Modulus/Remainder	$23 \% 6 = 5$
**	Power/Exp	$2 ** 3 = 8$

```
25**0.5
```

(1)

```
import math as mt  
mt.sqrt(25)
```

(2)

برای جذر گرفتن دو راه وجود دارد: (جذر عدد ۲۵)

در روش دوم، از کتابخانه `math` استفاده میکنیم و آن را در محیط پایتون با دستور `import` وارد میکنیم. با دستور `as`، میتوانید اسم کتابخانه مورد نظر را، به صورت اختصاری تعریف کنید و از این به بعد با آن کتابخانه را فراخوانی کنید.

عملگر های مقایسه ای و منطقی

- کوچکتر مساوی: \leq
- بزرگتر مساوی: \geq
- نامساوی: \neq
- کوچکتر: $<$
- بزرگتر: $>$
- مساوی: $=$

• عملگر های مقایسه ای:

- موجودیت چیزی در چیز دیگر: in
- شرط مساوی بودن قوی: is
- برعکس کردن نتیجه T/F: not/\sim
- "یا" منطقی: $\text{or}/|$
- "و" منطقی: $\text{and}/\&$

• عملگر های منطقی:

گزاره های شرطی (if statement)

- گاهی اوقات میخواهیم که اگر شرطی برقرار بود، اتفاقی بیفتد یا به طور مثال چیزی انجام شود، در این صورت از گزاره های شرطی استفاده میکنیم.
- گزاره های شرطی با if شروع میشوند، بعد از if شرط مورد نظر را باید بنویسیم و سپس : باید بگذاریم و به سطر بعدی برویم (هرچند الزامی نیست، ولی باعث خوانا تر شدن کد می شود).
- در خطوط بعد اتفاقی که در صورت برقرار بودن شرط میخواستیم بیفتد را مینویسیم. (توجه کنید که باید حتما خط بعدی، از اول خط به اندازه ۴ space فاصله داشته باشد، به این موضوع indentation میگویند و برای پایتون مهم و معنا دار است).

```
a=10
if a>5:
    print('a is greater than 5')
```

- در مثال بالا، چون که a برابر ۱۰ است، نوشته 5 is greater than a چاپ میشود.
- اگر a کمتر از ۵ باشد، چه اتفاقی می افتد؟ هیچی!
- برای اینکه متوجه بشویم که کد اجرا شده است یا نه و بخواهیم که در این صورت نیز خروجی بگیریم، از دستور else استفاده میکنیم.

گزاره های شرطی (if statement)

```
a=12
if a>15:
    print('a is greater than 15')
elif a>10:
    print('a is between 10 and 15')
else:
    print('a is less than 10')
```

- در مثال روبرو چون که a از ۵ کوچک تر است، کد های زیرِ بلاک else اجرا میشود.

- حال اگر بخواهیم تعداد شرط ها را زیاد کنیم باید چه کنیم؟

- از دستور elif استفاده میکنیم که از دو بخش else و if تشکیل شده است و به تعداد دلخواه میتوانیم بیاوریم:

```
a=3
if a>5:
    print('a is greater than 5')
else:
    print('a is less than 5')
```

- در مثال روبرو چون که a برابر ۱۲ است، بلاک دوم که همان elif باشد اجرا میشود و a is between 10 and 15 چاپ میشود.

```
if condition :
    do_something
elif condition2 :
    do_something_else1
else:
    do_something_else2
```

- سینتکس:

حلقه while

```
while condition :  
    statement1  
    statement2  
...
```

- در این حلقه ها، تا وقتی که یک شرط برقرار است، کاری انجام میشود.
- سینتکس:

```
count=0  
while count<10:  
    print('iteration',count)  
    count+=1
```

- در استفاده از این دستور دقت کنید که در یک حلقه بینهایت و بدون توقف نیفتید!
- در مثال روبرو، خروجی های ما 0 iteration تا 9 iteration خواهد بود.

- در این مثال اگر متغیر count، هر دفعه به مقدارش اضافه نمیشد، یک حلقه بینهایت میداشتیم و همیشه 0 iteration چاپ میشد.

حلقه for

- در این حلقه ها، به ازای یک سری متغیر که مقدارشان توسط عناصر یک دنباله تغییر میکند، یک سری اتفاقات می افتد.

```
for variable in sequence :  
    statement1  
    statement2  
    ...
```

- سینتکس:

- در مثال زیر، هر دفعه متغیر word به یکی از اعضای داخل لیست (به ترتیب موقعیت) میرسد و مقدارش عوض میشود. به ازای هر مقدار، یک سری کد که زیر حلقه for نوشته شده اند، اتفاق می افتد. در مثال زیر هر دفعه متغیر word یک مقدار مثل 'hello'، 'apple' تا 125.5 گرفته و هر کدام از این اعضا، با چاپ شدن متغیر word، چاپ میشوند.

```
lis1=['hello' , 'apple' , 'Inventory' , 234 , 125.5]  
for word in lis1:  
    print(word)
```

output →

```
hello  
apple  
Inventory  
234  
125.5
```

حلقه for

- نحوه دیگر رسیدن به همان خروجی مثال قبل، این است که ایندکس عوض شود و عنصر مطابق با آن ایندکس چاپ شود
- برای این کار باید یک متغیر مثل `i` بسازیم که اعدادش از ۰ شروع شود و تا یکی کمتر از تعداد اعضای لیستمان ادامه داشته باشد.

```
lis1=['hello' , 'apple' , 'Inventory' , 234 , 125.5]  
lis2=[0,1,2,3,4]  
for i in lis2:  
    print(lis1[i])
```

output →

```
hello  
apple  
Inventory  
234  
125.5
```

- آیا اگر تعداد اعضای لیستمان زیاد باشد، این کار بهینه است؟ خیر
- برای این کار از تابع `range` استفاده میکنیم و دو پارامتر اصلی آن که `range(start,step)` میباشد. این تابع به ما لیستی از اعداد میدهد که ابتدای آن `start` و انتهای آن `stop-1` است. پس اگر `range(0,5)` بدهیم، به ما اعداد ۰ تا ۴ را برمیگرداند.

```
lis1=['hello' , 'apple' , 'Inventory' , 234 , 125.5]  
for i in range(0,5):  
    print(lis1[i])
```

output →

```
hello  
apple  
Inventory  
234  
125.5
```

دستورات انتقال کنترل (Transfer Statements)

1. break:

```
count=0
while True:
    print('iteration',count)
    count+=1

    if count==10:
        print('the code will be stopped')
        break
```

- گاهی اوقات میخواهید اگر شرطی برقرار شد، دیگر ادامه حلقه for یا while انجام نشود و با یک دستور این حلقه بشکند.
- در اینجور مواقع از break استفاده میکنید.
- با یک مثال این دستور را بیشتر بررسی میکنیم.
- در مثال روبرو، همیشه while ما برقرار است. چرا که شرط آن True است. با این اوصاف ما در یک لوپ بینهایت افتاده ایم و کد هیچوقت از اجرا شدن متوقف نمیشود.
- حال فرض کنید ما تمایل داریم اگر عدد count برابر با ۱۰ شد، کد متوقف شود، یکی از راهکارها این است که از break استفاده کنیم (نحوه دیگر کد نویسی این بخش در مثال مربوط به while وجود دارد).
- به این صورت که اگر عدد count برابر با ۱۰ شد از چرخه while بیرون بیا و دیگر این چرخه را ادامه نده.

دستورات انتقال کنترل (Transfer Statements)

- برای این که اطمینان حاصل کنیم که آیا دستور `break` ما اجرا میشود یا خیر و چه زمانی میشود، از دستور `print` در داخل `if` استفاده میکنیم و خروجی را در زیر میبینید:

```
iteration 0  
iteration 1  
iteration 2  
iteration 3  
iteration 4  
iteration 5  
iteration 6  
iteration 7  
iteration 8  
iteration 9  
the code will be stopped
```

- به عنوان مثال دیگر، فرض کنید میخواهیم توان دوم اعداد ۱ تا ۱۰ را با یک دستور `for` اگر این عدد کوچک تر از ۸۰ بود، چاپ کنیم.

```
for num in range(1,11):  
    num2=num**2  
    if num2>80:  
        print('num^2 is greater than 80')  
        break  
    print(num2)
```

output →

```
1  
4  
9  
16  
25  
36  
49  
64  
num^2 is greater than 80
```

دستورات انتقال کنترل (Transfer Statements)

2. continue:

- گاهی اوقات میخواهید اگر شرطی برقرار شد، یک سری محاسبات (کد) مشخص روی ورودی انجام نشود، همچنین نمیخواهید که حلقه کاملاً متوقف شود و قصد دارید محاسبات برای دیگر اعضای دنباله ای که دارید انجام شود.
- در این مواقع از دستور continue استفاده میشود.
- فرض کنید میخواهید توان دوم اعداد ۱ تا ۱۰ بدون عدد ۵ را دستور for چاپ کنید:

```
for num in range(1,11):  
    if num==5:  
        print('number' , num , 'is passed')  
        continue  
  
    print(num**2)
```

output →

```
1  
4  
9  
16  
number 5 is passed  
36  
49  
64  
81  
100
```

- وقتی عدد num برابر با ۵ میشود، ادامه کد (که دستور `print(num**2)` در آنجا است)، اجرا نمیشود و به اول لوپ for برمیگردد و تکرار بعدی را آغاز میکند. مشابه این دستور با while در نوتبوک آمده است.

توابع در پایتون (functions)

- چه زمانی از تابع استفاده کنیم؟ وقتی که می‌خواهیم از انجام یک کار تکراری پرهیز کنیم. به صورت مثال فرض کنید در پروژه تان می‌خواهید چندین جا، روی یک ورودی، یک سری تغییرات انجام دهید، به جای اینکه هر دفعه این تغییرات را به ازای یک ورودی کد بزنید، به صورت کلی یک کدی می‌نویسید (که همان تابع ما باشد) و سپس با فراخوانی کردن این تابع، مقدار ورودی را عوض کرده و خروجی می‌گیرید.
- سینتکس:

```
def function(p1,p2,...):  
    do_something  
    return ...
```

- در مثال زیر، تابع add دو ورودی a و b دارد که این دو عدد را جمع کرده و به عنوان خروجی (که جلوی دستور return آمده است) به ما برمیگرداند. اگر بعد از تعریف تابع، آن را فراخوانی کنیم و به جای پارامترهای آن مقدار بگذاریم، داریم:

```
def add(a,b):  
    return a+b  
  
print(add(3,4))
```

output →

7

بیشتر درباره List

- اگر بخواهیم به بیشتر از یک عضو در لیست دسترسی پیدا کنیم چه کنیم؟ با مفهوم slicing باشد آشنا شویم. برای slicing باید بعد از آوردن اسم لیست و براکت، ابتدا و انتهای ایندکسی که می‌خواهیم عضو متناظر با آن به ما نمایش داده شود را بیاوریم. به مثال زیر خوب دقت کنید.

```
l1=[1 , 2 , 7 , 54.73333 , 'Stats' , 'DS' , True]  
l1[3:6]
```

- خروجی به نظر شما چیست؟ ☐ [54.73333, 'Stats', 'DS', True] ☒ [54.73333, 'Stats', 'DS']

- موقع slice کردن، محدوده ای که به پایتون می‌دهید، اولی بسته و دومی باز است، به این معنا که عضو مربوط به آخرین عدد محدوده برای شما نمایش داده نخواهد شد و یکی قبل آن نمایش داده میشود.

- همچنین میتوان محدوده را از یک طرف باز گذاشت:

```
l1[4:]  
l1[:4]
```

output → ['Stats', 'DS', True]
output → [1, 2, 7, 54.73333]

بیشتر درباره List

- تابع های کاربردی در لیست ها:

```
l2=[4,6,2,9,4,66,774,33333]  
len(l2)  
max(l2)  
min(l2)  
sorted(l2)  
sorted(l2,reverse=True)
```

output



8

output



33333

output



2

output



[2, 4, 4, 6, 9, 66, 774, 33333]

output



[33333, 774, 66, 9, 6, 4, 4, 2]

- توابع مختص به لیست: (method)

- lis.sort() مرتب کردن عناصر لیست
- lis.append(element) اضافه کردن عنصر به لیست
- lis.pop() حذف آخرین عنصر
- lis.insert(position,element) اضافه کردن عنصر به موقعیت مشخص
- lis.reverse() برعکس کردن ترتیب لیست
- lis.index(element) پیدا کردن موقعیت یک عنصر خاص
- lis.count(element) تعداد تکرار های عنصر خاص

کاربرد آمار توصیفی در پایتون

• برای این کار ابتدائاً بهتر است که مختصری با کتابخانه ی numpy آشنا شویم:

- NumPy is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, Fourier transform, and matrices. NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely. NumPy stands for Numerical Python.



```
import numpy as np
```

• برای کارکردن با این کتابخانه اول باید آن را وارد کنیم:

• ۲۰ عدد تصادفی صحیح از ۰ تا ۹۹ تولید میکنیم: (خروجی آن یک آرایه است).

```
lis=np.random.randint(0,100,size=20)  
lis
```

output →

```
array([36, 91, 59, 89, 90, 18, 55, 70, 19, 13, 43, 88, 15, 25, 7, 67, 39,  
       83, 15, 89])
```

• برای بدست آوردن میانگین، واریانس، انحراف معیار، میانه، صدک های مختلف (اعم از چارک اول، دوم که همان میانه باشد و چارک سوم) از

```
np.mean(lis)  
np.var(lis)  
np.std(lis)  
np.median(lis)  
np.quantile(lis , [0.25, 0.5, 0.75])
```

output →

```
50.55  
914.4475  
30.239833002184387  
49.0  
[18.75 49.    84.25]
```

دستورات زیر استفاده میکنیم:

کاربرد آمار توصیفی در پایتون

- برای دیگر معیار های آمار توصیفی (مثل کشیدگی، چولگی و ...) از کتابخانه SciPy استفاده میکنیم:

- SciPy is a scientific computation library that uses NumPy underneath. SciPy stands for Scientific Python. It provides more utility functions for optimization, stats and signal processing. Like NumPy, SciPy is open source so we can use it freely. SciPy was created by NumPy's creator Travis Olliphant.



```
import scipy as sp
```

- برای کارکردن با این کتابخانه اول باید آن را وارد کنیم:

- برای بدست آوردن مد، چولگی و کشیدگی دستورات زیر را میزنیم:

```
sp.stats.mode(lis)  
sp.stats.skew(lis)  
sp.stats.kurtosis(lis)
```

output

```
ModeResult(mode=array([15]), count=array([2]))  
0.04962058969269815  
-1.5557990129381396
```

- برای محاسبه ضریب همبستگی دو لیست (یا آرایه): (فعلا با عدد اول خروجی سر و کار داریم و مفهوم عدد دوم را بعدا بررسی میکنیم)

```
lis1=[1,2,3,4,5,6,7,8]  
lis2=[-1,-2,-3,-4,-5,-6,-7,-8]  
sp.stats.pearsonr(lis1,lis2)
```

output

```
(-0.9999999999999998, 2.736911063134408e-47)
```

```
lis1=np.random.randint(0,100,size=20)  
lis2=np.random.randint(0,100,size=20)  
sp.stats.pearsonr(lis1,lis2)
```

→

```
[33 76 57 79 47 85 14 63 95 65  4 62 62 18 23  0 56 85 81 95]  
[56 87 17 66 67 66 47 97 61 66 87 89 51  1 82 25 53 77 36 72]  
(0.2963592540598178, 0.20452837800785195)
```

کاربرد آمار توصیفی در پایتون

- خیلی اوقات نیاز است با یک سری دیتا که از منبع خارجی آمده است، کار کنیم، یا اینکه با جداول گسترده تر از یک لیست کار کنیم. در این شرایط استفاده از کتابخانه pandas ضروری میشود.

- Pandas is a Python library used for working with data sets. It has functions for analyzing, cleaning, exploring, and manipulating data. The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis" and was created by Wes McKinney in 2008.



- برای کارکردن با این کتابخانه اول باید آن را وارد کنیم:

```
import pandas as pd
```

- یک جدول که در اکسل میتوان ساخت را تصور کنید این جدول شامل یک سری اسم ستون، ایندکس سطر و اعضا است که در هر ستون و سطر خاص قرار گرفته است.

- برای ساخت همچنین جدولی در پایتون باید از پانداز استفاده کنیم.

- جدول زیر که در اکسل ساخته شده است و راجع به احتمال پذیرش دانشجویان با شرایط تعیین شده است را در نظر بگیرید.

index	Student ID	GRE Score	TOEFL Score	CGPA	Research	Chance of Admit
0	25001	337	118	9.65	1	0.92
1	25002	324	107	8.87	1	0.76
2	25003	316	104	8	1	0.72
3	25004	322	110	8.67	1	0.8
4	25005	314	103	8.21	0	0.65
5	25006	330	115	9.34	1	0.9

نحوه کار با دیتافریم ها

- به همچین جدولی در پایتون، Dataframe میگویند و آن را اکثرا با علامت اختصاری df نشان میدهند.
- حال میخواهیم همچین جدولی را در پایتون بسازیم:

```
students=[25001 , 25002 , 25003 , 25004 , 25005 , 25006]  
GRE=[337 , 324 , 316 , 322 , 314 , 330]  
Toefl=[118 , 107 , 104 , 110 , 103 , 115]  
CGPA=[9.65 , 8.87 , 8 , 8.67 , 8.21 , 9.34]  
research=[1 , 1 , 1 , 1 , 0 , 1]  
admission=[0.92 , 0.76 , 0.72 , 0.8 , 0.65 , 0.9]  
df=pd.DataFrame({'Student ID':students , 'GRE Score':GRE , 'TOEFL Score':Toefl , 'CGPA':CGPA,  
                  'Research':research , 'Chance of Admit':admission})  
df
```

↓
indno

	Student ID	GRE Score	TOEFL Score	CGPA	Research	Chance of Admit
0	25001	337	118	9.65	1	0.92
1	25002	324	107	8.87	1	0.76
2	25003	316	104	8.00	1	0.72
3	25004	322	110	8.67	1	0.80
4	25005	314	103	8.21	0	0.65
5	25006	330	115	9.34	1	0.90

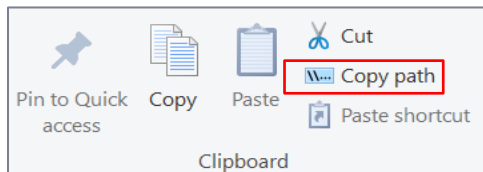
که ستون اول همان ایندکس ما است که به صورت اتوماتیک ساخته شده است.

نحوه کار با دیتافریم ها

- اگر از یک منبع خارجی (مثال اکسل) بخواهیم همچنین دیتایی را وارد کنیم باید چه کنیم؟
- اگر فایل مورد نظر CSV باشد، داریم:

```
df=pd.read_csv(r"D:\uni\Teaching Assistant\Inventory Planning & Control I\Slides & Notebooks\Admission_Predict.csv")
```

- آدرس فایل را با کلیک کردن روی فایل مورد نظر و سپس زدن ^{آدرس فایل} copy path انتخاب کنید.



- خروجی دیتافریم به صورت زیر است:

Serial No.	Student ID	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit	
0	1	25001	337	118	4	4.5	4.5	9.65	1	0.92
1	2	25002	324	107	4	4.0	4.5	8.87	1	0.76
2	3	25003	316	104	3	3.0	3.5	8.00	1	0.72
3	4	25004	322	110	3	3.5	2.5	8.67	1	0.80
4	5	25005	314	103	2	2.0	3.0	8.21	0	0.65
...
395	396	25396	324	110	3	3.5	3.5	9.04	1	0.82
396	397	25397	325	107	3	3.0	3.5	9.11	1	0.84
397	398	25398	330	116	4	5.0	4.5	9.45	1	0.91
398	399	25399	312	103	3	3.5	4.0	8.78	0	0.67
399	400	25400	333	117	4	5.0	4.0	9.66	1	0.95

نحوه کار با دیتافریم ها

- برای دیدن یک ستون `df['SOP']`
- برای دیدن چند ستون `df[['SOP' , 'LOR' , 'CGPA']]`
- برای حذف ستون خاص `del df['Serial No.']` / `df.drop('Serial No.' , axis=1)`
- برای عوض کردن ایندکس با مقادیر یک ستون دیگر `df.set_index('Student ID')`
- برای دیدن سطر خاص با استفاده از اسم ایندکس `df.loc[25002]`
- برای دیدن سطر خاص با استفاده از شماره سطر (اینکه سطر چندم است) `df.iloc[1]`
- برای دیدن عضو سطر با اسم i و ستون با اسم j `df.loc[25002 , 'LOR']`
- برای دیدن عضو سطر شماره i و ستون شماره j `df.iloc[1 , 5]`
- برای دیدن بازه ای از اطلاعات `df.loc[25002:25007 , ['GRE Score' , 'SOP']]` / `df.iloc[1:7 , [1,4]]`
- برای دیدن چند سطر اول (به صورت دیفالت ۵ تا) `df.head()`
- برای دیدن چند سطر آخر (به صورت دیفالت ۵ تا) `df.tail()`
- برای دیدن اسم ستون ها `df.columns`
- برای دیدن اسم ایندکس ها `df.index`
- برای دیدن اندازه دیتافریم `df.shape`
- برای دیدن آمار توصیفی مختصر `df.describe()`
- برای دیدن اطلاعات ستون ها `df.info()`

نحوه کار با دیتافریم ها

- اگر بخش خاصی از دیتافریم که شرط خاصی بر آن صدق کند را بخواهیم، باید چه کنیم؟
- باید با مفهوم dataframe projection آشنا شویم:

○ به این صورت است که شرط مورد نظر را داخل براکت بعد از اسم دیتافریم می آوریم و خروجی دیتافریمی است که در آن شرایط صدق کند. مثلا اطلاعات افرادی که شانس پذیرش بالای ۹۰٪ دارند:

```
df[df['Chance of Admit']>0.9].head()
```

output

	Serial No.	Student ID	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	1	25001	337	118	4	4.5	4.5	9.65	1	0.92
22	23	25023	328	116	5	5.0	5.0	9.50	1	0.94
23	24	25024	334	119	5	5.0	4.5	9.70	1	0.95
24	25	25025	336	119	5	4.0	3.5	9.80	1	0.97
25	26	25026	340	120	5	4.5	4.5	9.60	1	0.94

نحوه کار با دیتافریم ها

○ یا مثلا اگر بخواهیم هم شانس پذیرش بالای ۹۰٪ باشند هم تافل بالاتر از ۱۱۶:

```
df[(df['Chance of Admit']>0.9) & (df['TOEFL Score']>116)].head()
```

output

	Serial No.	Student ID	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	1	25001	337	118	4	4.5	4.5	9.65	1	0.92
23	24	25024	334	119	5	5.0	4.5	9.70	1	0.95
24	25	25025	336	119	5	4.0	3.5	9.80	1	0.97
25	26	25026	340	120	5	4.5	4.5	9.60	1	0.94
32	33	25033	338	118	4	3.0	4.5	9.40	1	0.91

```
df.sort_values('Chance of Admit')
```

- مرتب کردن دیتافریم بر اساس ستون خاص به صورت صعودی:

```
df.sort_values('Chance of Admit',ascending=False)
```

- مرتب کردن دیتافریم بر اساس ستون خاص به صورت نزولی:

نحوه کار با دیتافریم ها

- ترانهاده (transpose) کردن دیتافریم `df.T`
 - حذف کردن سطر خاص (با اسم) `df.drop(25002)`
- توجه کنید با اینکار، دیتافریم شما واقعا تغییر نمیکند و صرفا دیتافریمی به شما برمیگرداند که سطر ۲۵۰۰۲ را ندارد، در صورتی که دیتافریم اصلی شما عوض نشده و هنوز آن سطر را دارد. برای تغییر باید دوباره مقدار آن را به `df` تخصیص دهید، یا اینکه پارامتر `inplace` را برابر `True` قرار دهید.
- (1) `df.drop(25002,inplace=True)` (2) `df=df.drop(25002)`

- ریست کردن ایندکس (ستون ایندکس قبلی را جز ستون های اصلی دیتافریم میبرد و اتوماتیک ایندکس میزند) `df.reset_index()`
- مرتب کردن ایندکس `df.sort_index()`
- دیدن مقادیر منحصر به فرد یک ستون `df['SOP'].unique()`
- دیدن اینکه آیا داده NaN داریم یا نه `df.isnull()`
- حذف کردن سطر هایی که در آن مقدار NaN وجود دارد `df.dropna()`
- عوض کردن یک مقدار خاص در دیتافریم با یک مقدار دیگر (در این مثال، هر جا که در ستون TOEFL Score ۱۱۸ وجود دارد، ۱۱۹ جایگذاری میشود).
`df['TOEFL Score'].replace(118,119)`

رسم نمودار



- کتابخانه های زیادی در پایتون میتوانند برای رسم نمودار های مختلف به کار گرفته شوند.

1. Matplotlib

2. Seaborn

3. Plotly

4. Ggplot

- کتابخانه های دیگری نیز وجود دارند، در این کورس به بررسی مختصری از کتابخانه Matplotlib میپردازیم.
- نحوه استفاده از این کتابخانه به دو صورت است:

1. مبتنی بر MATLAB است و از یک رابط مبتنی بر حالت (state-based interface) استفاده می کند.

2. مبتنی بر شی گرایی و Object-Oriented-Programming

- در این کورس حتی الامکان از حالت اول استفاده میکنیم، چرا که ساده تر است و قابل درک تر میباشد. همچنین نحوه کد زدن در این بخش شباهت زیادی با متلب دارد.

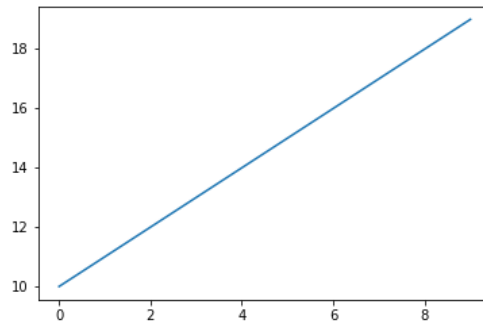
رسم نمودار

- اول از همه باید کتابخانه مربوطه را وارد کنیم:
- همچنین پیشنهاد میشود که از magic function که فقط و فقط در خود ژوپیتر کار میکند استفاده کنید. با استفاده از این تابع جادویی، میتوانید جزییات نمودار هایی که رسم میکنید را دقیق تر و بررسی کنید و امکانات جدیدی برایتان فراهم میشود.

```
%matplotlib notebook
```

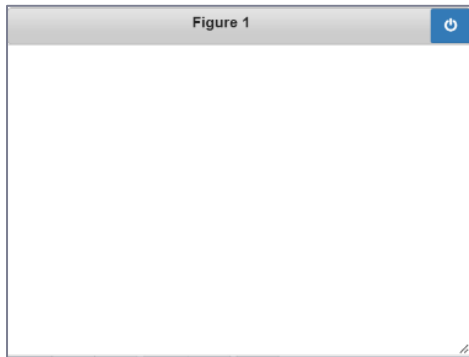
- حال میخواهیم یک خط صاف با معادله $y=x+10$ رسم کنیم که دامنه آن از ۰ تا ۹ است. برای این کار یک سری اعداد متوالی از ۰ تا ۹ درست میکنیم، با این حساب y هم از اعداد متوالی ۱۰ تا ۱۹ تشکیل شده است.

```
%matplotlib notebook
import matplotlib.pyplot as plt
import numpy as np
plt.figure()
plt.plot(np.arange(0,10) , np.arange(10,20))
plt.show()
```



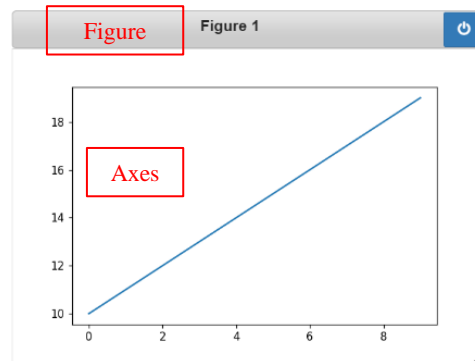
- به کدی که زده شده است توجه کنید. ابتدا با دستور `plt.figure()` یک فیگور کلی برای ما درست میشود که در اسلاید بعد بررسی میکنیم.

رسم نمودار



- حال که فیگور ایجاد شد، می‌خواهیم در این فیگور یک سری نمودار در یک Axes ترسیم شود، برای رسم یک خط از دستور `plt.plot` استفاده می‌کنیم. آرگومان‌های دیگر این تابع را می‌توانید در [دکویمنتیشن](#) این تابع بررسی کنید.
- در نهایت با دستور `plt.show()` نموداری که درست کردیم را به نمایش می‌گذاریم. برای ذخیره کردن نمودار نیز می‌توانید از دستور زیر استفاده کنید.

```
plt.savefig('name of the plot')
```

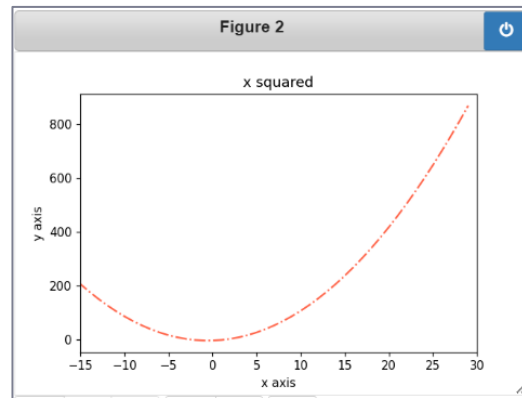


رسم نمودار

- به عنوان یک مثال دیگر و کامل تر، میخواهیم $y=(x-1)(x+2)$ را ترسیم کنیم.

```
plt.figure()
x=np.arange(-30 , 30)
y=(x-1)*(x+2)
plt.plot(x ,y , linestyle='-' , color='tomato')
plt.xlabel('x axis')
plt.ylabel('y axis')
plt.xlim([-15 , 30])
plt.title('x squared')
plt.show()
```

output



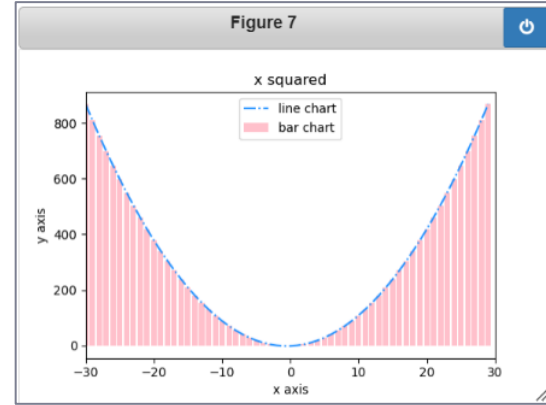
- ابتدا متغیر X را آرایه ای از اعداد -30 تا 30 قرار میدهم و متغیر Y را بر اساس آن تعریف میکنیم. (توجه کنید که چون جنس اعداد تولید شده آرایه میباشد، به همین دلیل عملیات جمع و ضرب به صورت جمع و ضرب مرتب اعضای داخل آرایه تعریف میشود. یعنی اگر X را ضرب در 5 کنیم، طول آن 5 برابر نمیشود (مثل لیست نیست) بلکه اعداد داخل آن 5 برابر میشود.
- در گام بعد موقع رسم X و Y ، طرح خط واصل را به نقطه خط تبدیل میکنیم و رنگش را به "گوجه ای" عوض میکنیم.
- سپس برای محور افقی و عمودی، اسم میگذاریم، برای کل نمودار تایتل قرار میدهم.
- نکته دیگری که باید به آن توجه شود این است که میتوانید به دلخواه بازه و دامنه نمایش داده شده در نمودار را تغییر دهید. در اینجا، این دامنه نمایش داده شده، به -15 تا 30 تغییر یافته است.

رسم نمودار

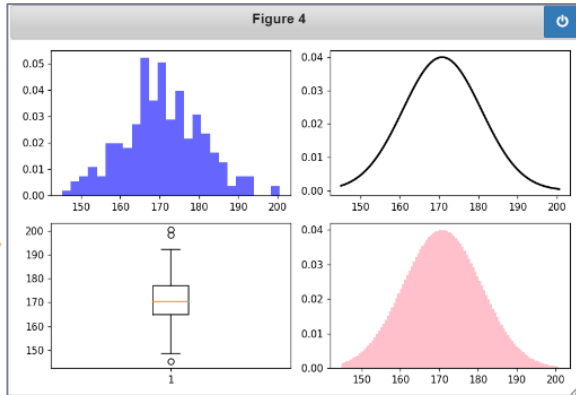
- همچنین میتوان چند نمودار را در یک فیگور و در یک Axes کشید. در مثال قبل، بارچارت مربوطه نیز اضافه شده است. با دستور `plt.legend` مشخص میشود که کدام رنگ مربوط به کدام نمودار است.

```
plt.figure()
x=np.arange(-30 , 30)
y=(x-1)*(x+2)
plt.plot(x,y , linestyle='-.', color='dodgerblue',label='line chart')
plt.bar(x , y , color='pink',label='bar chart')
plt.xlabel('x axis')
plt.ylabel('y axis')
plt.xlim([-30 , 30])
plt.title('x squared')
plt.legend()
plt.show()
```

output →



- حال فرض کنید که میخواهیم یک فیگور با ۴ تا Axes داشته باشیم، مثال روبرو را در نظر بگیرید:



رسم نمودار

```
import numpy as np
from scipy.stats import norm
import matplotlib.pyplot as plt 1

plt.figure(figsize=(8,5))

plt.subplot(2,2,1)
data = np.random.normal(loc=170, scale=10, size=250)
mu=np.mean(data)
std=np.std(data) 2
# Plot the histogram.
plt.hist(data, bins=25, density=True, alpha=0.6, color='b')

# Plot the PDF.
plt.subplot(222)
x = np.linspace(min(data), max(data) , 100)
pdf_x = norm.pdf(x, mu, std)
plt.plot(x, pdf_x, 'k', linewidth=2) 3

# Plot the boxplot.
plt.subplot(223)
plt.boxplot(data, showfliers=True) 4

# Plot the barchart.
plt.subplot(224)
plt.bar(x, pdf_x , color='pink') 5

plt.tight_layout()
plt.show() 6
```

- برای رسم چنین نموداری، کد زیر زده شده است که با هم هر بخش را بررسی میکنیم:
- **1:** اول از همه، کتابخانه های مورد نیاز را وارد میکنیم. توجه کنید که خط آخر بخش اول، در عبارت `figsize`، اندازه فیگوری که میخواهیم درست شود را مشخص میکنیم.
- **2:** اگر به نمودار اسلاید پیش توجه کنید، از ۴ Axes در یک `figure` تشکیل شده است. برای درست کردن یک فیگور با ۴ Axes خالی، از دستور `plt.subplot` استفاده میکنیم که پارامتر اول تعداد سطر و پارامتر دوم تعداد ستون های `figure` است. پارامتر سوم، Axes ای که میخواهیم در آن نمودار را ترسیم کنیم، مشخص میکند. در این بخش میخواهیم هیستوگرام یک متغیر تصادفی نرمال با میانگین ۱۷۰ و انحراف معیار ۱۰ با اندازه نمونه ۲۵۰ را، در گوشه بالا سمت چپ `figure` رسم کنیم. این ۲۵۰ نمونه در `data` ذخیره شده است. با دستور `plt.hist` این نمودار ترسیم میشود. پارامتر `density`، مشخص میکند که `pdf` را رسم کند یا خیر. `alpha` نیز شفافیت نمودار را عوض میکند.
- **3:** در بخش سوم، میخواهیم نمودار را در گوشه بالا سمت راست ترسیم کنیم که جایگاه دوم را داراست. (توجه کنید که `plt.subplot(222)` معنی یکسانی با `plt.subplot(2,2,2)` دارد) سپس فضای بین مینیمم و ماکسیمم متغیر تصادفی نرمالی که درست شده است را به ۱۰۰ بخش مساوی تقسیم میکنیم و اعداد جدا کننده بخش ها را در `X` میریزیم.

رسم نمودار

`np.linspace(3 , 9 , 3)` **output** `array([3., 6., 9.])`

```
import numpy as np
from scipy.stats import norm
import matplotlib.pyplot as plt

plt.figure(figsize=(8,5))

plt.subplot(2,2,1)
data = np.random.normal(loc=170, scale=10, size=250)
mu=np.mean(data)
std=np.std(data)
# Plot the histogram.
plt.hist(data, bins=25, density=True, alpha=0.6, color='b')

# Plot the PDF.
plt.subplot(222)
x = np.linspace(min(data), max(data) , 100)
pdf_x = norm.pdf(x, mu, std)
plt.plot(x, pdf_x, 'k', linewidth=2)

# Plot the boxplot.
plt.subplot(223)
plt.boxplot(data, showfliers=True)

# Plot the barchart.
plt.subplot(224)
plt.bar(x, pdf_x , color='pink')

plt.tight_layout()
plt.show()
```

• متغیر X یک دنباله حسابی با قدر نسبت $(\max - \min) / (\text{size} - 1)$ میباشد.

• سپس PDF نرمال این اعداد را با تابع `norm.pdf` (که در بخش اول `norm` را وارد کردیم) حساب میکنیم و PDF شان را با دستور `plt.plot` رسم میکنیم. ('k' رنگ مشکی است)

• 4: در این بخش باکس پلات متغیر تصادفی نرمال محاسبه شده را می یابیم. به این نحو که دوباره جایگاه نموداری که میخواهیم در آن ترسیم کنیم را مشخص کرده و سپس از دستور `plt.boxplot` شروع به ترسیم میکنیم. پارامتر دومی که در داخل دستور `boxplot` داده شده است، داده های پرت را مشخص میکند. (outliers)

• 5: بارچارت مربوط به داده های X و PDF متناظرشان را رسم میکنیم.

• 6: دستور `plt.tight_layout` باعث میشود که اگر اعدادی داخل فیگور، در هم رفته اند و یا فضا های اضافی داخل نمودار را از بین ببرد. استفاده از آن در تمامی رسم نمودار ها توصیه میشود. در نهایت فیگوری که رسم کردیم را با `plt.show` نمایش میدهیم.

راهنمای استفاده از نمودارها

- همانطور که گفته شد، نمودار های زیادی وجود دارد که کتابخانه های متفاوتی از آن ها پشتیبانی میکنند. نحوه تشخیص اینکه از چه نموداری، در چه زمانی و در کجا استفاده شود، این نمودار چه پارامتر هایی دارد و ... فقط و فقط با جست و جو در اینترنت بدست می آید.
- Matplotlib برای هر کدام از دستوراتی که برای رسم پلات های مختلف دارد، در داکيومنتیشن خود، مثال حل شده نیز دارد. سعی کنید برای نحوه استفاده از نمودارها، به سایت خودش مراجعه کنید.
- از دیگر سایت هایی که نمودار های مختلف و محل استفاده از نمودار های مختلف را به شما نمایش میدهد، این سایت است.
- توجه کنید که هر نمودار در این کتابخانه ممکن است بیش از ۲۰ پارامتر داشته باشد، نکته مهم این است که شما از این پارامتر ها، تنها آن هایی که به بهتر شدن کارتان کمک میکند استفاده کنید، و این را به خاطر داشته باشید، همیشه پیچیده بودن به معنای بهتر بودن نیست.
- تنها در صورتی پارامتر اضافه بدهید که به دانش خواننده چیزی اضافه کند، در غیر این صورت اضافی است و نباید آورده شود.

ورودی گرفتن از کاربر

- برای ورودی گرفتن از کاربر، از دستور `input` استفاده میکنیم. به این صورت که پیغامی که می‌خواهیم برای کاربر چاپ شود را داخل "نوشته، سپس از کاربر خواسته میشود تا جواب بدهد و چیزی که به ما برمیگردد، از جنس `string` خواهد بود.

```
height = input('enter your height: ')\nprint(type(height))\n\n#how to convert str to float\nheight=float(height)\nprint(type(height))
```

output

```
enter your height: 175.5\n<class 'str'>\n<class 'float'>
```

- در این مثال، از کاربر خواسته میشود تا قد خود را وارد کند، با اینکه ورودی ما عدد است، اما به صورت رشته ذخیره میشود، برای تبدیل آن به عدد از جنس اعشاری، باید دستور `float` را روی آن بزنیم تا این اتفاق بیفتد.

دیگر دیتاستراکچرها

• از دیگر دیتاستراکچرهای معروفی که (علاوه بر List) در پایتون وجود دارد و استفاده زیادی از آن ها میشود:

• Tuple • Dictionary • Set

• Tuples:

• تاپل ها کالکشنی مرتب از دیتاهای با ایندکس و مرتب هستند و اولین عضو آن با ایندکس [0] و دومین عضو [1] و ... است.

• تاپل ها میتوانند عضو تکراری ذخیره کنند.

• وقتی که عضوی به تاپل ها اختصاص داده شد، مقدار آن نمیتواند عوض شود. (immutable)

• میتوان دیتاتایپ های متفاوتی در تاپل ها ذخیره کرد (اعم از رشته، بولین، اعداد اعشاری و ...).

• نحوه تعریف آن ها به صورت زیر است که باید اعضای آن را بین دو پرانتز بنویسیم و آن ها را با , جدا کنیم:

```
0      1      2      3      4      5
tup1=(1 , 2 , 54.73333 , 'Stats' , 'DS' , True)
```

• نحوه دیدن اعضای آن و کار با ایندکس دقیقاً مانند لیست ها میباشد.

```
tup1[0:3] → output (1, 2, 54.73333)
```

دیگر دیتا استراکچرها

```
tup1[1]=12 #error
```

- همانطور که گفته شد اعضای آن را نمیتوان تغییر داد، پس با اجرا کردن کد زیر به ارور مواجه میشویم:

```
tup1.index('Stats') → output 3
```

- پیدا کردن مکان یک عضو خاص:

```
tup2=(1 , 2 ,3,3,3,3, 54.73333 , 'Stats' , 'DS' , True) → output 4  
tup2.count(3)
```

- تعداد تکرار یک عضو خاص:

- Dictionary:

- کالکشنی از دیتا که از کلید و مقدار متناظر با آن کلید تشکیل شده است (key-value pair).

- قابل تغییر هستند و هم میتوان اعضای آن را عوض کرد، هم به آن key-value جدید اضافه کرد.

- نحوه ی تعریف دیکشنری:

```
dic1={1:'gum' , 2:'money' , 3:'key' , 4:'wallet'}
```

↑ key ↑ value

- به طور مثال اگر عدد یک را به عنوان کلید فراخوانی کنیم، مقدار متناظر با آن 'gum' برگردانده میشود.

```
dic1[1] → output 'gum'
```

دیگر دیتاستراکچرها

- محدودیتی در تعریف key نداریم و میتوانیم از هر نوع تاییپی که میخواهیم آن را تعریف کنیم.

```
dic1={1:'gum' , 2:'money' , 'hello':'key' , 4:'wallet'}  
dic1['hello']
```

output → 'key'

- عوض کردن value متناظر با یک key:

```
dic1['hello']='salam'  
dic1
```

output → {1: 'gum', 2: 'money', 'hello': 'salam', 4: 'wallet'}

- دسترسی پیدا کردن به تمام key ها:

```
d2={1:'one' , 2:'two' , 3:'three' , 4:'four'}  
list(d2.keys())
```

output → [1, 2, 3, 4]

- دسترسی پیدا کردن به تمام value ها:

```
list(d2.values())
```

output → ['one', 'two', 'three', 'four']

- دسترسی پیدا کردن به جفت کلید و مقدار:

```
list(d2.items())
```

output → [(1, 'one'), (2, 'two'), (3, 'three'), (4, 'four')]

دیگر دیتاستراکچرها

- Set:

- همان مجموعه ای است که در ریاضیات میخوانیم.

- کالکشنی از یک سری دیتا میباشد که حتما غیر تکراری هستند.

- اعضا ترتیب ندارند و نمیتوان با استفاده از ایندکس آن ها را فراخوانی کرد.

- نحوه تعریف:

```
s={1,2,2,2,2,2,2,2,4,5,'hello'}  
s
```

output → {1, 2, 4, 5, 'hello'}

- میبینیم که اعضا تکراری نیستند و هر عضو یک بار تکرار شده است.

- اجتماع دو مجموعه:

```
s1={1,2,3,4}  
s2={3,4,4,4,4,4,4,5,6,7,8}  
s1.union(s2)
```

output → {1, 2, 3, 4, 5, 6, 7, 8}

- اشتراک دو مجموعه:

```
s1.intersection(s2)
```

output → {3, 4}

- بررسی اینکه آیا یک مجموعه زیر مجموعه ی مجموعه دیگر است یا نه:

```
s1.issubset(s2)
```

output → False

- اعضای در یک مجموعه که در مجموعه دیگر نیست:

```
s1.difference(s2)
```

output → {1, 2}

مثال ۱: میانگین متحرک

- فرض کنید مقدار تقاضای یک کالا برای ۱۰ دوره ی گذشته به صورت زیر است:

دوره	۱	۲	۳	۴	۵	۶	۷	۸	۹	۱۰
تقاضا	۵.۸	۶	۶.۱	۶.۳	۶.۴	۶.۴	۶.۵	۶.۷	۶.۹	۷

خواسته شده است که مقدار پیشبینی دوره های ۵ تا ۱۱ را با استفاده از روش میانگین متحرک ساده با $k=4$ بدست آورید.

پاسخ:

- ابتدا مقادیر تقاضا را وارد یک لیست میکنیم، سپس از ۴ تا تقاضای آخر در هر کدام از دوره ها که هستیم، میانگین میگیریم، به طور مثال، مقدار پیشبینی در دوره ۶، میانگین مقادیر تقاضا در دوره های ۲، ۳، ۴ و ۵ میباشد.
- چه تعداد پیشبینی خواهیم داشت؟ پیشبینی برای دوره ۵، ۶، ...، ۱۱، پس ۷ مقدار داریم. لیست خالی ای به اسم Forecast میسازیم و مقادیر پیشبینی را در آن ذخیره میکنیم.

مثال ۱: میانگین متحرک

```
demands = [5.8 , 6 , 6.1 , 6.3 , 6.4 , 6.4 , 6.5 , 6.7 , 6.9 , 7]
forecast = []
k=4

for i in range(7):
    forecast.append(np.mean(demands[i:i+k]))
print(forecast)
```

output

```
[6.05, 6.199999999999999, 6.299999999999999, 6.4, 6.5, 6.625, 6.775]
```

	demands	forecast
1	5.8	NaN
2	6.0	NaN
3	6.1	NaN
4	6.3	NaN
5	6.4	6.050
6	6.4	6.200
7	6.5	6.300
8	6.7	6.400
9	6.9	6.500
10	7.0	6.625
11	NaN	6.775

هرچند میتوان مقادیر تقاضا و پیشبینی را به صورت منظم تر در یک دیتافریم وارد کرد نتیجه را بتوان راحت تر مقایسه کرد:

مثال ۱: میانگین متحرک

حال فرض کنید مقدار واقعی تقاضا در دوره ۱۱ برابر با ۷.۲ است. خواسته شده است که مقدار متریک MSE را برای مقادیر پیشبینی که انجام داده اید حساب کنید:

در حله اول، مقدار ۷.۲ را به لیست تقاضا اضافه میکنیم:

```
demands.append(7.2)
```

روش اول) استفاده از for loop:

```
MSE=0
for i in range(7):
    SSE += (demands[i+k]-forecast[i])**2
MSE=SSE/len(forecast)
MSE
```

output

```
0.11053571428571447
```

روش دوم) استفاده از خاصیت برداری آرایه های numpy:

```
demands = np.array(demands)
forecast = np.array(forecast)
MSE=0
MSE = np.mean((demands[k:]-forecast)**2)
MSE
```

output

```
0.11053571428571447
```

مثال ۱: میانگین متحرک

- متریک میانگین خطای نسبی را بدست آورید.

روش اول) استفاده از for loop:

```
PE=0
for i in range(7):
    PE+=(demands[i+k]-forecast[i])/demands[i+k]
MPE=PE/len(forecast)
MPE
```

output

0.04743615300202517

روش دوم) استفاده از خاصیت برداری آرایه های numpy:

```
MPE=np.mean((demands[k:]-forecast)/demands[k:])
MPE
```

output

0.04743615300202517

مثال ۲: هموار سازی نمایی ساده

- فرض کنید مقدار تقاضای یک کالا برای ۱۰ دوره ی گذشته به صورت زیر است:

دوره	۱	۲	۳	۴	۵	۶	۷	۸	۹	۱۰
تقاضا	۵.۸	۶	۶.۱	۶.۳	۶.۴	۶.۴	۶.۵	۶.۷	۶.۹	۷

مقادیر پیشبینی را با استفاده از روش هموار سازی نمایی ساده با آلفای ۰.۲ برای دوره های ۲ تا ۱۱ محاسبه کنید.

پاسخ:

- توجه داشته باشید که شاید نوشتن تابع چه در این سوال چه در سوال قبل، رویکرد بهتری باشد تا اینکه بخواهیم برای هر دیتاست، از اول کد را بنویسیم. اما ورودی های تابع مد نظر برای هموار سازی نمایی ساده چه خواهد بود؟
- این تابع به آلفا، مقدار پیشبینی دوره قبل و مقدار تقاضای دوره قبل احتیاج دارد تا مقدار پیشبینی دوره فعلی را به ما به عنوان خروجی بدهد.

مثال ۲: هموار سازی نمایی ساده

```
def simple_exp_forecast(alpha , forecast_prev , demand):  
    forecast = forecast_prev + alpha*(demand - forecast_prev)  
    return forecast
```

- تابع مذکور:

- حال نوبت به محاسبه مقادیر پیشبینی با کمک تابع نوشته شده میباشد، توجه کنید که برای محاسبه پیشبینی دوره دو، چون که مقدار پیشبینی در دوره اول نداریم، همان مقدار تقاضای دوره اول را به جای پیشبینی دوره اول نیز قرار میدهیم.

```
demands = [5.8 , 6 , 6.1 , 6.3 , 6.4 , 6.4 , 6.5 , 6.7 , 6.9 , 7]  
forecast=[]  
alpha=0.2  
  
for i in range(len(demands)):  
    if i==0:  
        prev_forecast = demands[i]  
        forecast.append(simple_exp_forecast(alpha , prev_forecast , demands[i]))  
    else:  
        prev_forecast=forecast[i-1]  
        forecast.append(simple_exp_forecast(alpha , prev_forecast , demands[i]))
```

output
→

```
[5.8,  
5.84,  
5.8919999999999995,  
5.973599999999999,  
6.058879999999999,  
6.127103999999999,  
6.2016832,  
6.30134656,  
6.421077248,  
6.5368617983999995]
```

مثال ۲: هموار سازی نمایی ساده

- همچنین بهتر بود اگر از دیتافریم استفاده میکردیم تا راحت تر متوجه محاسبات بشویم و خروجی مثل زیر میشد:

	demands	forecast
1	5.8	NaN
2	6.0	5.800000
3	6.1	5.840000
4	6.3	5.892000
5	6.4	5.973600
6	6.4	6.058880
7	6.5	6.127104
8	6.7	6.201683
9	6.9	6.301347
10	7.0	6.421077
11	NaN	6.536862

رگرسیون

- توجه داشته باشید که کتابخانه های زیادی برای فیت کردن مدل های رگرسیون و ... است، از پرکاربردترین آن ها که برای کارهای یادگیری ماشین معمولاً استفاده میشود، استفاده میکنیم.
- کتابخانه scikit-learn از کتابخانه های پیشرو در حوزه ماشین لرنینگ و دیتاساینس میباشد به طوری که مدل های بسیار زیادی را در خود جای میدهد.
- مدل های رگرسیونی نیز از جمله مدل هایی است که بسیار زیاد مورد استفاده کاربران قرار میگیرد که قرار است صرفاً با این بخش از کتابخانه کار کنیم و به بخش های دیگر کتابخانه نخواهیم پرداخت.
- در اسلاید بعد، گام های طی شده برای پیاده سازی یک مدل رگرسیون را با هم بررسی خواهیم کرد.



رگرسیون

```
from sklearn.linear_model import LinearRegression

demands = np.array([5.8 , 6 , 6.1 , 6.3 , 6.4 , 6.4 , 6.5 , 6.7 , 6.9 , 7])
periods = np.arange(1,11)

linreg = LinearRegression()
linreg = linreg.fit(periods.reshape(-1, 1) , demands.reshape(-1, 1))
linreg.predict(np.array([11,12]).reshape(-1, 1))
```

output
→

```
array([[7.1      ],
       [7.22545455]])
```

- مدل رگرسیون خطی را از کتابخانه وارد میکنیم، سپس آجکتی از آن در محیط پایتون میسازیم. قبل از ساختن آجکت، عملاً شما رگرسیون خطی را وارد کرده اید، ولی استفاده ای از آن جایی نشده است. حال با در نظر گرفتن مقادیر periods به عنوان X و مقادیر demands به عنوان y، میخواهیم خط رگرسیونی به آن ها برازش یا اصطلاحاً فیت کنیم.
- پس آجکتی که ساختیم را فراخوانی میکنیم و از متد فیت استفاده میکنیم و مقادیر X و y را به ترتیب به آن میدهیم. توجه کنید که دلیل استفاده از متد `reshape(-1,1)` به علت این است که این کتابخانه برای انجام محاسبات مورد نیاز، ورودیش را به صورت ماتریسی دریافت میکند، این درحالیست که demands و periods هر دو یک بردار هستند (1-D array) و ماتریس (2-D array) نمیباشند.

رگرسیون

- حال که مدل رگرسیونی را ساختیم و پارامترها محاسبه شده است، نوبت به پیشبینی برای مقادیر دلخواه دوره میباشد، که با استفاده از متد predict انجام میشود.
- با به طور مثال میتوانیم مقادیر پیشبینی را برای دوره های ۱ تا ۱۰ ببینیم و با مقادیر واقعی مقایسه کنیم:

```
pd.DataFrame({'period':periods,'demand':demands , 'forecast':linreg.predict(periods.reshape(-1,1)).flatten()})
```

output

	period	demand	forecast
0	1	5.8	5.845455
1	2	6.0	5.970909
2	3	6.1	6.096364
3	4	6.3	6.221818
4	5	6.4	6.347273
5	6	6.4	6.472727
6	7	6.5	6.598182
7	8	6.7	6.723636
8	9	6.9	6.849091
9	10	7.0	6.974545

رگرسیون

- مقدار متریک R^2 : برای محاسبه این مقدار، صرفاً X و y را به عنوان ورودی به متد score میدهم:

```
linreg.score(epochs.reshape(-1, 1), demands.reshape(-1, 1))
```

output → 0.9770162117791915

- با در نظر گرفتن این که مدل رگرسیونی به شکل $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots$ باشد، به β_1, β_2, \dots coefficient و به β_0 intercept میگویند، با این حساب داریم:

```
linreg.intercept_
```

output → array([5.72])

```
linreg.coef_
```

output → array([[0.12545455]])

- به طور مثال اگر بخواهیم مقدار پیشبینی را در دوره ۱۱ حساب کنیم:

```
linreg.intercept_[0]+linreg.coef_[0][0]*11
```

output → 7.1

- که برابر با مقدار پیشبینی ۲ اسلاید قبل است.