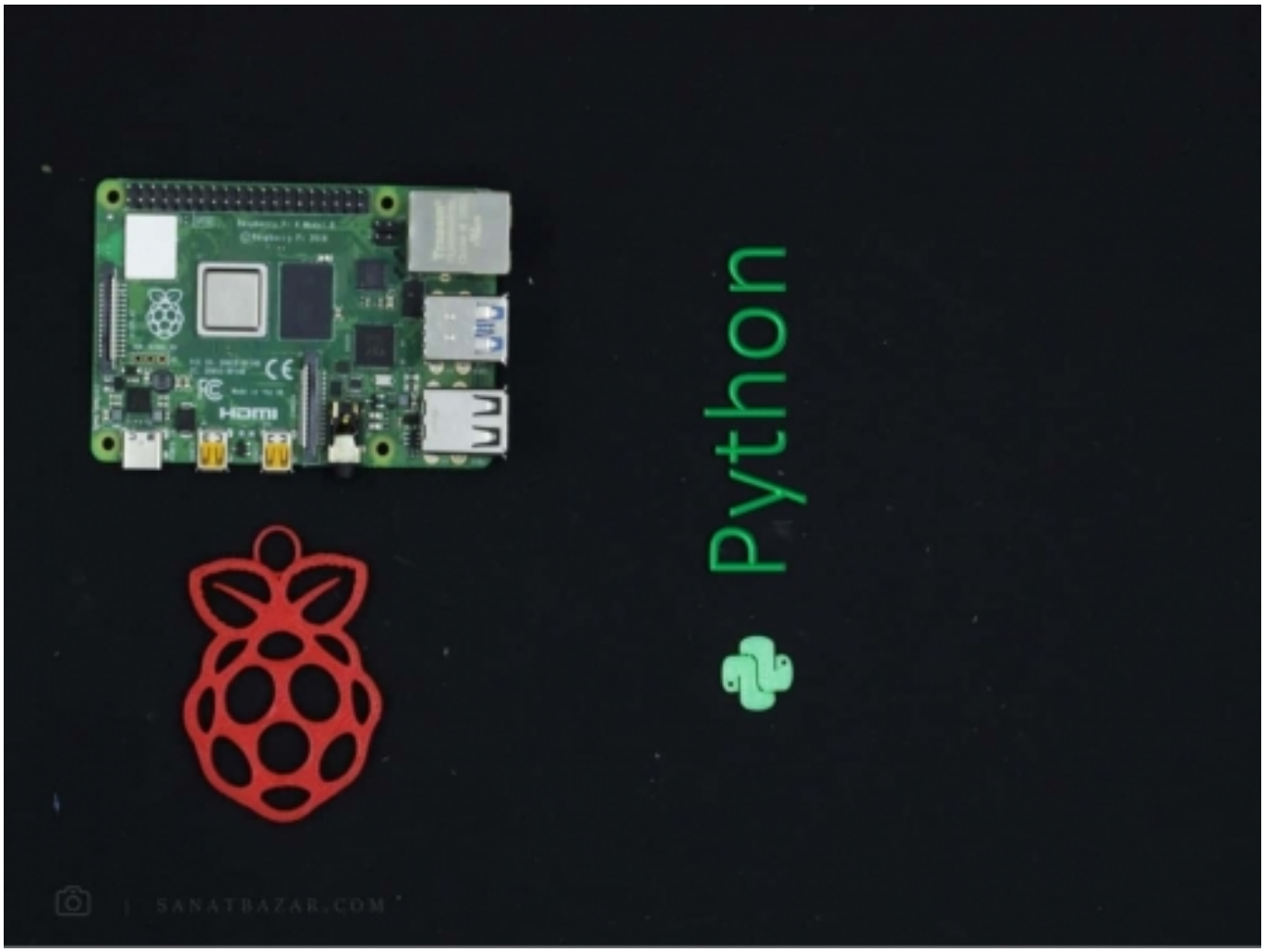
معرفی کامل دستورات کاربردی پایتون برای کار با رزبری پای



امروزه استفاده از واژه‌ی هوشمند برای دستگاه‌های شامل کامپیوترهای Embedded مثل موبایل هوشمند، تلوزیون هوشمند، یخچال هوشمند و ... رایج شده است. اما در حقیقت این کامپیوترها و دستگاه‌ها نیستند که هوشمندند، چرا که فقط دستورات برنامه‌نویس را مو یه مو اجرا می‌کنند. این برنامه‌نویس‌ها هستند که هوشمند شده‌اند!! در دنیایی که همه چیز رو به هوشمندی و کامپیوتری شدن می‌رود، بدون شک برنامه‌نویسان مهم‌ترین افراد یک جامعه خواهند شد. در این بین، یکی از زبان‌های بسیار کاربردی برای برنامه‌نویسی بردهای Embedded، وب و هوش مصنوعی، زبان پایتون (Python) است. بردهای رزبری‌پای هم در کنار زبان‌های برنامه‌نویسی مختلف مانند C++، جاوا، Scratch، از پایتون پشتیبانی می‌کنند. اصلاً واژه‌ی پای (Pi) برای سیستم‌های مبتنی بر پایتون استفاده می‌شود! پس اگر می‌خواهید در کنار یادگیری برنامه‌نویسی رزبری‌پای و انجام پروژه‌های عملی مختلف، از افراد مهم و تاثیر گذار آینده باشید، این مطلب می‌تواند نقطه‌ی شروعی برای شما باشد. اگر با برنامه‌نویسی آشنا نیستید، اصلاً نگران نباشید چون قصد داریم از پایه شروع کنیم. اگر هم بلدید که چه بهتر، می‌توانید یادگیری مباحث جدید را سریع‌تر پیش ببرید. پس با من همراه باشید که می‌خواهیم مباحث مقدماتی و کاربردی پایتون را به ترتیب زیر به زبان ساده و با مثال یاد بگیریم:

- مقدمه‌ای بر پایتون و ویژگی‌های آن از جمله شی‌گرایی
- آموزش نصب و راه‌اندازی پایتون
- آموزش دستورات کاربردی پایتون به زبان ساده و همراه با مثال

مقدمه: چرا پایتون؟

پایتون یک زبان برنامه‌نویسی سطح بالاست که به دلیل تمرکز بر سادگی، خوانایی دستورات، شی‌گرایی و همچنین قدرت بالای خود در زمینه‌های یادگیری ماشین و آنالیز داده، در بین برنامه‌نویسان بسیار محبوب شده است. همین قدرت و محبوبیت بالا کافیست تا شرکت‌های بزرگی مانند گوگل، اینستاگرام، آمازون و اسپاتیفای برای ارتقای سرویس‌های خود توسط آنالیز داده‌های کاربران، کم‌کم به این زبان برنامه‌نویسی کوچ کنند. طبق ادعای مجله‌ی معتبر The Economist، پایتون در حال تبدیل شدن به محبوب‌ترین زبان برنامه‌نویسی در میان مهندسان و برنامه‌نویسان است. به طور کلی از این زبان برای برنامه‌نویسی سیستم‌های اتوماسیون، هوش مصنوعی، نرم‌افزارها و

 | این محتوا توسط صنعت بازار تولید یا برگردان شده است - کپی رایت © 2019

وبسایت‌ها استفاده می‌شود. (پس اگه خوب پایتونو بلد باشی، بیکار نمی‌مونی!!!)

قبل از ورود به آموزش پایتون، اجازه بدبد با دو ویژگی مهم آن یعنی خوانایی و شی‌گرایی آشنا شویم:

سادگی و خوانایی (Readability): توسعه دهندگان پایتون همیشه سعی کرده‌اند تا شیوه‌ی کدنویسی و دستورات، به زبان انسان بسیار نزدیک باشد. این ویژگی، یادگیری و استفاده از این زبان برنامه‌نویسی را نسبت به سایر زبان‌ها ساده‌تر کرده است. البته این سادگی به معنای پیش پا افتاده بودن این زبان نیست. در واقع پایتون در عین قدرتمند بودن برای حرفه‌ای‌ها، سادگی را هم برای تازه‌کارها فراهم کرده است.

شی‌گرایی (Object Oriented Programming): تا قبل از ورود زبان‌های شی‌گرا، برنامه‌نوسی به صورت رَویه‌ای (Procedural) انجام می‌شد. در این روش _که هم‌اکنون برای انجام بسیاری از پروژه‌های درسی و ساده استفاده می‌شود_ برنامه‌ی کلی به چندین بخش تقسیم شده و با وارد کردن متغیرها به هر بخش، خروجی آن محاسبه و در صورت نیاز دوباره به قسمت دیگری از برنامه معرفی می‌شد. اما در صورتی که برنامه‌ی شما پیچیده و طولانی باشد، وابستگی و تعداد زیاد متغیرها و توابع می‌تواند دردسر ساز باشد. برای حل این مشکل، ایده‌ی برنامه‌نویسی شی‌گرا مطرح شد. به‌طوری که در این روش تعدادی از متغیرها و توابع در یک واحد مستقل به نام شی تعریف و در برنامه‌های مختلف، فراخوانی می‌شوند. به متغیرهای هر شی، Properties و به توابع آن Methods گفته می‌شود. پس تا اینجا گفتیم درOOP تعدادی از مغیرها و توابعی که به هم مربوطند، در یک شی جمع‌آوری می‌کنیم تا بعداً در جاهای مختلف از آن‌ها استفاده شود(Encapsulation). با این کار می‌توان از یک سری تابع و متغیر، در بخش‌های مختلف یک برنامه یا سایر برنامه‌ها، به صورت آماده استفاده کرد. همچنین برای ساده‌تر کردن رابط کاربری و استفاده از اشیا، برخی از Methodها و Propertyهایی که کاربر با آن‌ها کاری ندارد را پنهان می‌کنیم. مثلاً فرض کنید برای انجام یک عمل، چندین تابع تو در تو دارید. در این روش فقط تابع اول که لازم است فراخوانی شود، نمایش داده و سایر توابع درون آن به کاربر نشان داده نمی‌شوند. با این کار از نمایش کدهای اضافی و پیچیده جلوگیری می‌شود (Abstraction). علاوه‌بر این، با شی‌گرایی می‌توانید از دستورات یک شی در اشیای دیگر استفاده کنید. این کار نیز مانع از کدنویسی تکراری و طولانی می‌شود (Inheritance). همچنین در OOP این قابلیت را دارید که برای یک Method، با توجه به Property آن، رفتار متفاوتی تعریف کنید. بنابراین نیازی به نوشتن Switch-Case های مختلف و طولانی ندارید (Polymorphism). خب حالا که با مفاهیم بالا آشنا شدید،می‌خواهم شی‌گرایی را در یک جمله به شما معرفی کنم: به زبان برنامه‌نویسی که ویژگی‌های Inheritance، Abstraction، Encapsulation و Polymorphism را دارا باشد، شی‌گرا گفته می‌شود.

به طور کلی از ویژگی‌های پایتون می‌توان به موارد زیر اشاره کرد:

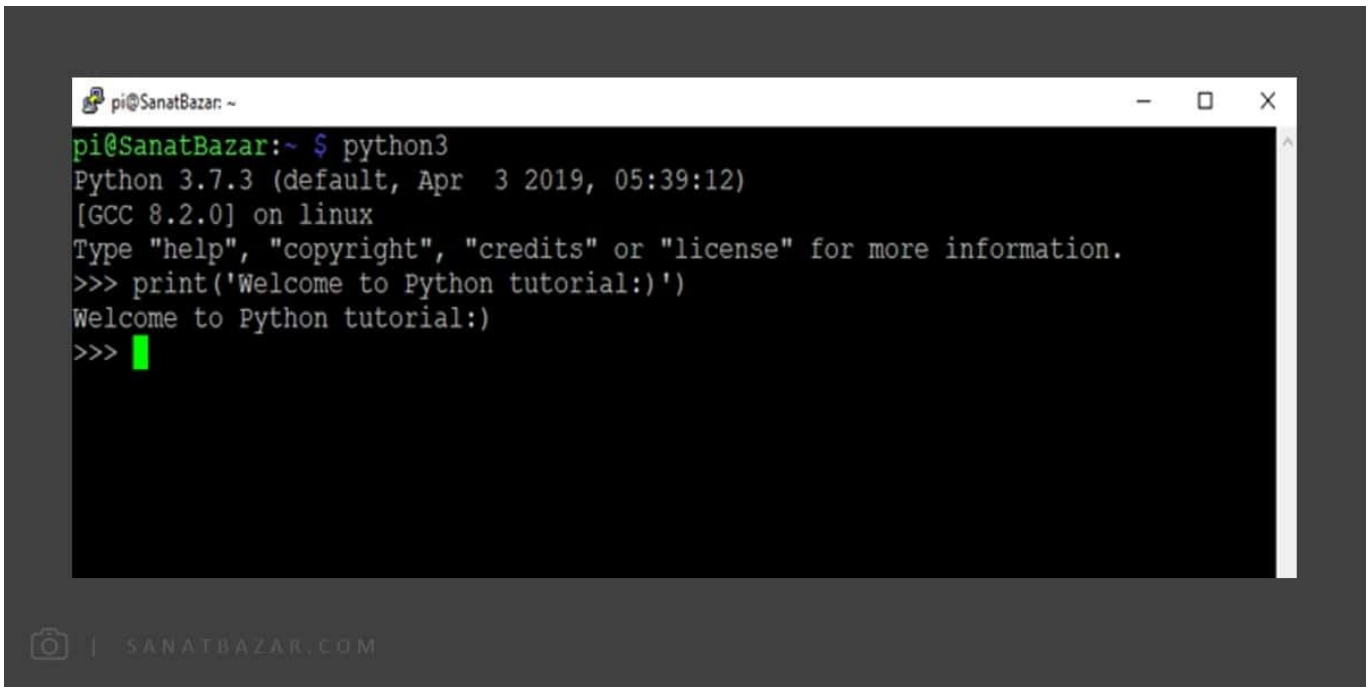
- سادگی و خوانایی دستورات
- شی‌گرایی
- وجود کتابخانه‌های کامل و به‌روز
- اجرای دستورات به صورت خط به خط و سادگی در شناسایی و رفع خطاهای برنامه‌نویسی

خب پس از این که با اهمیت و کاربردهای پایتون آشنا شدیم، آماده باشید که می‌خواهیم آموزش را شروع کنیم.

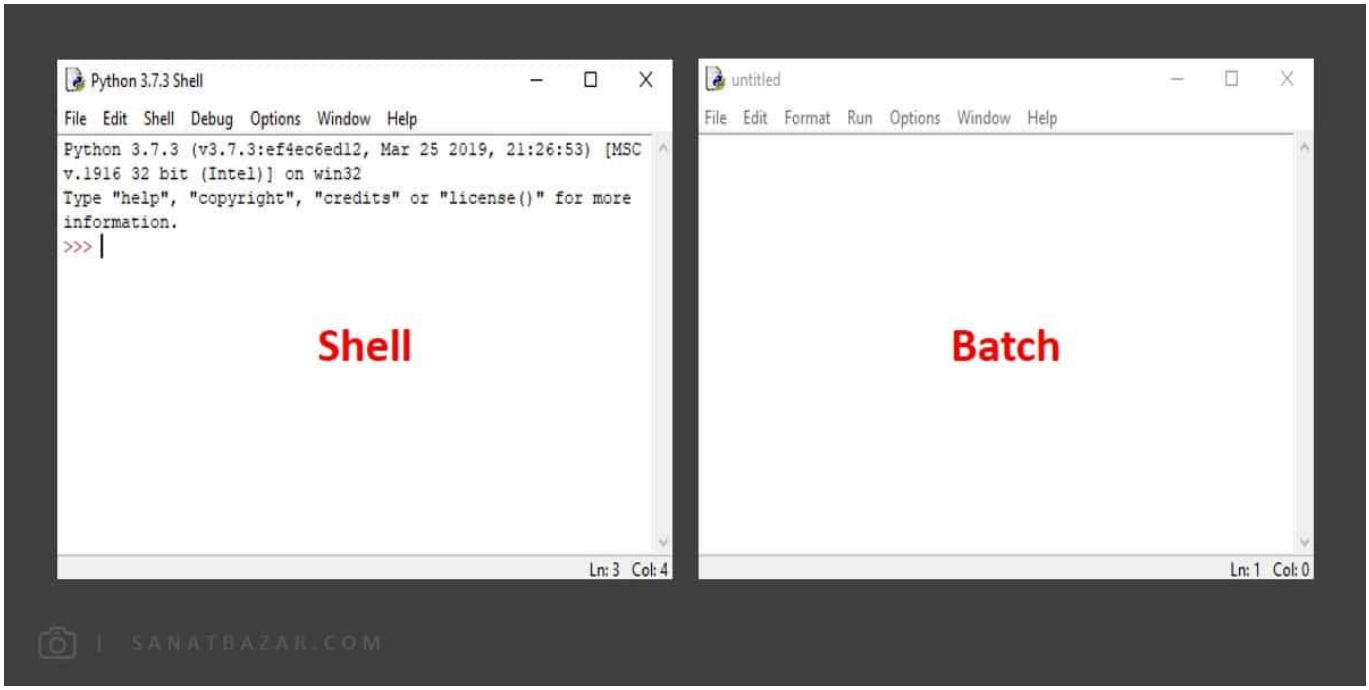
پیش‌نیاز: کدهای پایتون را کجا بنویسم؟ چطوری اجرا کنم؟

اگر رزبری‌پای ندارید، نگران نباشید. چرا که این آموزش فقط منحصر به این برد نبوده و هدف ما در اینجا صرفاً آموزش مقدماتی پایتون است. بنابراین برای یادگیری و اجرای دستورات پایتون روی کامپیوتر، می‌توانید آخرین نسخه‌ی آن را از سایت python.org دانلود کنید. در حال حاضر دو نسخه‌ی پایتون ۲ و پایتون ۳ منتشر شده که به جز اندکی تفاوت در Syntax دستورات، بسیار به هم شبیه می‌باشند. با توجه به این که بسیاری از وب‌سایت‌ها و برنامه‌ها در گذشته بر اساس پایتون ۲ نوشته شده و امکان انتقال این حجم عظیم کدها از نسخه‌ی ۲ به ۳ وجود ندارد، هنوز هم از نسخه‌ی دوم پشتیبانی می‌شود. با این وجود، ما در این بخش آموزشی با پایتون ۳ کار خواهیم کرد. اگر از رزبری‌پای استفاده می‌کنید، هر دو نسخه‌ی ۲ و ۳ به صورت پیش‌فرض روی سیستم‌عامل‌های این برد نصب شده است. برای کدنویسی، به طور کلی دو راه دارید: نوشتن کد به صورت Interactive در Shell یا نوشتن کد به صورت یکجا و Batch.

در روش اول دستورات یک به یک نوشته، اجرا شده و قابلیت ذخیره‌سازی ندارند. مسلماً این شیوه برای انجام پروژه‌های مختلف و پیچیده مناسب نیست. ما هم در اینجا فقط برای آشنایی و آموزش دستورات از این محیط استفاده خواهیم کرد. برای این کار در کامپیوتر می‌توانید از محیط Shell برنامه‌ی IDLE که دانلود کردید استفاده کنید. در رزبری‌پای نیز از طریق منوی Start، در بخش Programming می‌توانید Thonny Python IDE را انتخاب و از بخش Shell آن استفاده کنید. در محیط Command-Line نیز با اجرای python $ وارد Shell پایتون ۲ و با python3 $ وارد Shell پایتون ۳ خواهید شد. (همین الان هر روشی که براتون ممکنه اجرا کنید)

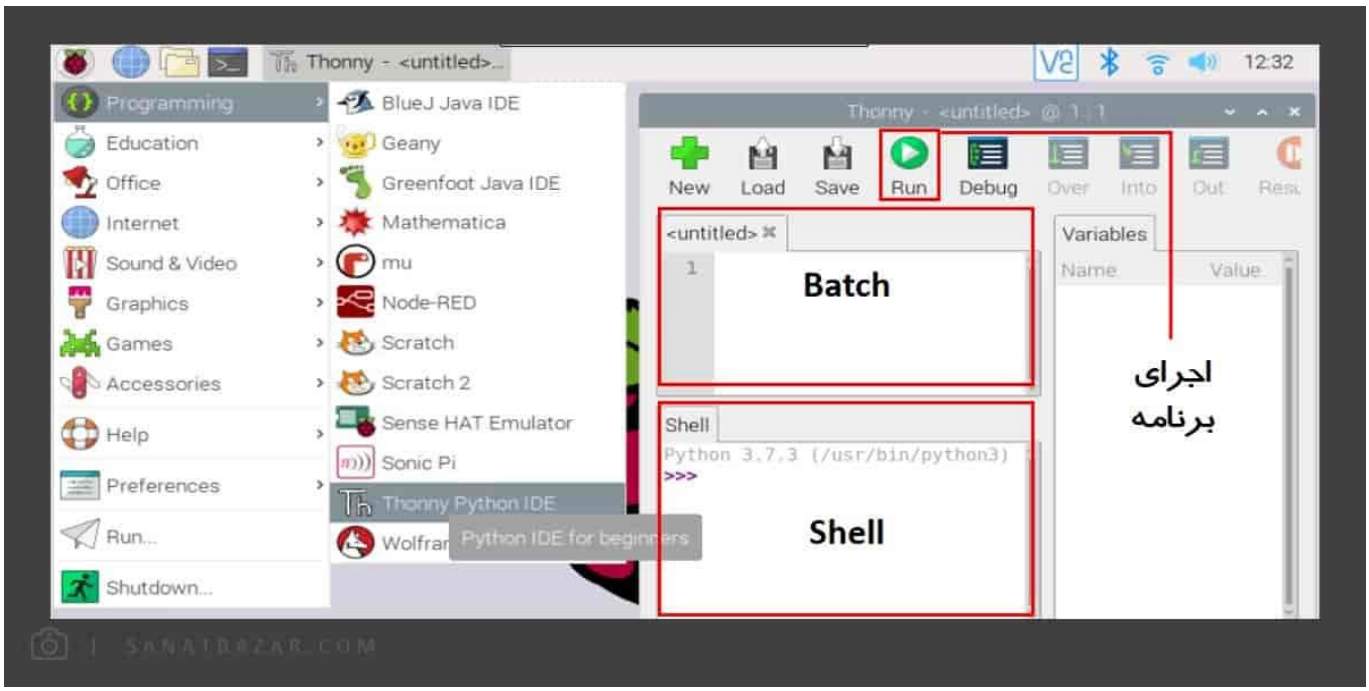


در روش دیگر، می‌توانید کدهای خود را ابتدا کامل نوشته و یکجا Run کنید. برای این کار در کامپیوتر از بخش File در برنامه‌ی IDLE، New file را انتخاب کرده و در بخش باز شده کدهای خود را بنویسید. در نهایت برای اجرا می‌توانید روی Run از نوار بالای صفحه، کلیک کنید.



علاوه‌بر این محیط که رسماً توسط پایتون ارائه شده، نرم‌افزارهای دیگری هم برای برنامه‌نویسی پایتون وجود دارد که در صورت نیاز می‌توانید از آن‌ها هم استفاده کنید. (به‌عنوان مثال، Pycharm یکی از بهترین محسط‌ها برای برنامه‌نویسی پایتون است)

در رزبین نیز می‌توانید در بخش بالایی صفحه (Batch) برنامه‌ی خود را نوشته و با زدن Run آن را اجرا کنید.



برای آشنایی با سایر برنامه‌های موجود در Programs می‌توانید به بخش راه‌اندازی رزبری پای Raspberry Pi 4 با نصب سیستم‌عامل رزبین (Raspbian Buster) مراجعه کنید.

اوضاع در Command-Line لینوکس از این هم ساده‌تر است. برای این کار کافیست کدهای خود را در یک فایل Text (مثلاً با استفاده از nano $) بنویسید و ذخیره کنید. بهتر است فایل با پسوند .py ذخیره شود (گذاشتن پسوند، اختیاری است). در نهایت با دستورات زیر به‌راحتی می‌توانید کد خود را اجرا کنید:

```
$ python
$ python3
```

نحوه‌ی کدنویسی به این روش، در ادامه به‌صورت تصویری نمایش داده شده است.

پس از این که با نحوه‌ی اجرای دستورات و کدنویسی آشنا شدید، بریم سراغ دستورات مهم و کاربردی.

برای مشاهده‌ی آموزش مقدماتی و کاربردی لینوکس می‌توانید به بخش معرفی دستورات کاربردی لینوکس برای کار با رزبری‌پای مراجعه کنید.

دستورات کاربردی و مهم پایتون

در این بخش قصد داریم دستورات مهم و کاربردی پایتون را با هم بررسی کنیم. برای یادگیری بهتر، دستورات را همزمان روی کامپیوتر یا رزبری‌پای خود اجرا کنید.

توجه: در این آموزش مقادیر خروجی هر دستور و برنامه، پس از >>> نمایش داده شده است.

print: اولین و پرکاربردترین دستوری است که معرفی می‌کنیم. از print برای نمایش خروجی‌ها استفاده می‌شود. برای مثال دستور زیر را در Shell اجرا کنید:

```
print ("Welcome to Python World!")
```

بر خلاف سایر زبان‌های برنامه‌نویسی، پایتون به Space حساس است. بنابراین دستورات خود را برای اجرا در اول هر سطر، بدون Space و Indent بنویسید. در غیر این صورت با خطا مواجه خواهید شد (به استثنای دستورات شرطی و حلقه‌ها که در ادامه به آن‌ها می‌پردازیم). همچنین در پایتون از ; در انتهای دستورات استفاده نمی‌شود. بنابراین وقتی در سطر بعدی دستوری را می‌نویسید، یعنی دستور سطر قبلی تمام شده است.

#: از این علامت برای Comment کردن دستورات استفاده می‌شود. پس با قرار دادن # قبل از دستورات، از اجرای آن‌ها جلوگیری می‌کنید. از Comment برای نوشتن توضیحات اضافه در مورد برنامه و دستورات استفاده می‌شود.

```
print ("Welcome to Python World!")     # This is our first command in this tutorial
```

برای Comment کردن چندین خط، به جای این که اول هر کدام از آن‌ها # قرار دهید، Comment ها را بین """ """بنویسید.

```
"""
Hello, ') print('
Print('Welcome')
Print('This is our first command in this tutorial')
"""
print ("Welcome to Python World!")
```

متغیرها: از متغیرها برای ذخیره و دریافت مقادیر استفاده می‌شود. نام‌گذاری آن‌ها لزوماً با حروف یا _ (Underline) شروع شده و تنها شامل حروف، اعداد و _ می‌باشد. توجه داشته باشید که در پایتون کلمات و دستورات کلیدی مانند for ،if و ... با رنگ نارنجی نمایش داده شده و برای نام‌گذاری متغیرها، مجاز به استفاده از آن‌ها نیستید. برای تخصیص یک مقدار به متغیر از = استفاده کنید.

```
a = 20
b2 = 11
my_variable = b2
```

بر خلاف زبان C++، نیازی نیست ابتدا نوع متغیر را تعریف کنید. خود Interpreter برنامه، نوع آن را از مقدار تخصیص داده شده، تشخیص می‌دهد.

مثال)

```
a = 20                    #است  Integer این متغیر از نوع
b = 20.5                #است  Float این متغیر از نوع
c = "Hello"          #است  String این متغیر از نوع
d = True               #است  Boolean این متغیر از نوع
```

Integer: اعداد صحیح مثبت و منفی

Float: اعداد اعشاری مثبت و منفی

String: حروف و کارکترها

Boolean: مقادیر منطقی صحیح و غلط

متغیرهای Boolean تنها شامل True و False هستند. این دو عبارت بدون quotation و دقیقاً به همین شکلی که می‌بینید باید نوشته شوند. در غیر این صورت شناخته نخواهند شد.

همچنین می‌توانید به چند متغیر در یک سطر و دستور مقدار مساوی تخصیص دهید:

مثال)

```
x = y = a = 2e3          #  e برای نمایش توانی از 10 استفاده می‌شود
print(x)
<<< 2000
```

برای تعریف متغیرهای String کافیست آن‌ها را بین ' ' یا " " قرار دهید.

پایتون Case Sensitive است. یعنی بین A و a تفاوت قایل می‌شود.

type: در زبان برنامه‌نویسی پایتون، نمی‌توانید دو متغیر از دو نوع مختلف را با هم جمع کنید. برای مثال، در برنامه‌ی زیر سعی شده دو متغیر string و int با هم جمع و جواب نهایی چاپ شود:

```
a = “How old are you? “
b = 14
print(a+b)
TypeError: can only concatenate str (not "int") to str   <<<
```

خب با توجه به این که نوع متغیرها را خود پایتون تشخیص می‌دهد، از کجا بفهمیم متغیر ما از چه جنسی است؟ برای این کار می‌توانیم از دستور type استفاده کنیم. به مثال زیر توجه کنید:

مثال)

```
a = 20
Type(a)
```

```
>>> <class 'int’>
```

```
 b = 1 + 2j          # j برای نمایش بخش موهومی یک عدد مختلط استفاده می‌شود
type(b)
```

```
>>> <class 'complex'>
```

همچنین با دستورات زیر می‌توانید کلاس متغیرها را تغییر دهید:

```
a = 2
b = int(a)                    # تبدیل به integer
b = float(a)              # تبدیل به float
b = str(a)                  # تبدیل به string
b = complex(a)     # تبدیل به complex
```

مثال)

```
int(2.8)
```

```
>>> 2
```

```
str(2.8)
```

```
>>> ‘2.8’
```

 | این محتوا توسط صنعت بازار تولید یا برگردان شده است - کپی رایت © 2019

len: از این تابع برای محاسبه‌ی طول (تعداد اعضا) یک آرایه استفاده می‌شود. به عنوان مثال می‌خواهیم تعداد کارکترهای (طول رشته) عبارت python را در مثال زیر بررسی کنیم:

```
a='Python'
print(len(a))
```

```
>>> 6
```

این دستور برای انواع داده‌ها از جمله Dictionary، Tuple، List و Set که در ادامه معرفی خواهیم کرد، قابل استفاده است.

اندیس‌ها: در پایتون دو شیوه برای اندیس‌گذاری آرایه‌ها وجود دارد. اندیس‌گذاری مثبت و اندیس‌گذاری منفی

در حالت مثبت، اندیس‌ها از سمت چپ و صفر شروع شده و تا len(متغیر)-1 ادامه دارد. (مثلاً برای آرایه‌ی تعریف شده در a، اندیس آخر len(a)-1 یعنی 5 است)

برای مثال قبلی یعنی کلمه‌ی پایتون داریم:

```
a[0]
```

```
>>> 'P'
```

```
a[2]
```

```
>>> 't'
```

اما در اندیس‌گذاری منفی، شرایط برعکس است. در واقع اندیس‌ها از سمت راست و از -1 شروع شده و به (متغیر -len(نیز ختم می‌شوند. بنابراین اندیس‌گذاری مثال قبلی به شکل زیر است:

```
a[-6]
```

```
>>> 'P'
```

```
a[-4]
```

```
>>> 't'
```

```
a[-1]
```

```
>>> 'n'
```

در حالت کلی، برای عبارت Python اندیس‌گذاری به شکل زیر است:

P	y	t	h	o	n
5	4	3	2	1	0

1- 2- 3- 4- 5- 6-

همچنین با [x : y] اعضای xام تا y-1 ام آرایه را انتخاب می‌کنیم:

```
a[1:4]    #3و 2،1 انتخاب اعضای
a[:3]       # از اندیس صفر تا دوم
a[3:]       #  از اندیس سوم تا آخر
```

in و not in: ازاین دو دستور برای بررسی وجود یا عدم وجود اعضا در یک آرایه استفاده می‌شود.

مثال)

```
txt = “Hello, how are you?”
a = “how” in txt
b = “you” not in txt
c = “good” in txt
print(a)
```

```
>>> True
```

```
Print(b)
```

```
>>> False
```

```
Print(c)
```

```
>>> False
```

variable.find و variable.index: برای پیدا کردن اندیس کارکترها از این دو دستور استفاده می‌شود. توجه داشته باشید که به جای variable باید نام متغیر خود را بنویسید.

مثال)

```
a = “Hello!”
a.find(‘e’)
```

```
>>>1
```

عملگرهای ریاضی:

مثال	کاربرد	عملگر
5 << 2+3	جمع	+
2 << 7-5	تفریق	-
8 << 2*4	ضرب	*
1.6666 << 5/3	تقسیم	/

27 << 3**3	توان	**
1 << 3 // 5	مقدار صحیح خارج قسمت (تقسیم صحیح)	//
2 << 5%3	باقیمانده	%

اولویت عملیات ریاضی مشابه زبان‌های دیگر به صورت زیر است:

۱- پرانتز

۲- توان

۳- ضرب و تقسیم

۴- جمع و تفریق

عملگرهای انتسابی: از این عملگرها برای ریختن مقادیر در متغیرها استفاده می‌شود.

مشابه	کاربرد	عملگر
x = 2	x = 2	=
x = x + 2	x += 2	=+
x = x □ 2	x -= 2	=-
x = x * 2	x *= 2	=*
x = x / 2	x /= 2	=/
x = x ** 2	x ** = 2	=**
x = x // 2	x // = 2	=//
x = x % 2	x % = 2	=%

عملگرهای مقایسه: این دسته از عملگرها در دستورات شرطی و حلقه‌ها بسیار کاربردی هستند.

مثال	کاربرد	عملگر
True << 2==2	تساوی	==
True << 5=!2	عدم تساوی	=!
True << 6>5	کوچکتر	<
True << 1<5	بزرگتر	>
True << 7=>5	کوچکتر مساوی	=>
False << 7=<5	بزرگتر مساوی	=<

به جای == از is و به جای != از is not نیز می‌توان استفاده کرد.

عملگر های منطقی: از این عملگرها نیز می‌توان در دستورات شرطی و حلقه‌ها استفاده کرد:

مثال	کاربرد	عملگر
and 5>1 << True 3>2	اگر همه‌ی شرط‌ها برقرار باشد، خروجی True است.	and
or 5>7 << True 3>2	اگر فقط یکی از شرط‌ها برقرار باشد، خروجی True است.	or
Not 5>3 >> False	نقیض: True را به False و False را به True تبدیل می‌کند.	not

List: لیست‌ها آرایه‌هایی تغییر پذیرند که می‌توان هر نوع داده‌ای را به آن‌ها اختصاص داد. این دسته از داده‌ها در پایتون با [ ] تعریف می‌شوند.

مثال)

```
list1 = ‘sib’ , ‘golabi’ , ‘angoor’]
list2 = [1 , 2 , 5 , 8 , 4]
list3 = ‘sib’ , 9 , 12 , True]
list4 = [1 , [1 , 2 , 3] , True , [‘apple’ , ‘banana’]]
```

اندیس‌گذاری لیست‌ها نیز مانند Stringها به دو صورت اندیس‌های مثبت و منفی انجام می‌شود. به مثال‌های زیر دقت کنید:

```
list1[1] >>> ‘golabi’
```

```
list2[1:3] >> [2 , 5]
list3[-2] >> 12
list4[1] >> [1 , 2 , 3]
```

سوال: در List4، چطوری داده‌ی 2 را انتخاب کنم؟ خیلی ساده اگر لیست‌های تو در تو داشتیم، مقدار اندیس‌ها را به ترتیب وارد می‌کنیم.

```
list4[1][1] >> 2
list4[1][-3:-1] >> [1 , 2]
```

برای این که ببینیم آیا یک داده در لیست ما وجود دارد یا نه می‌توانیم از دستورات in و not in استفاده کنیم.

مثال)

```
'apple' in list4 >> False
'apple' in list4[3] >> True
```

همچنین می‌توانیم با استفاده از = داده‌های جدیدی را به لیست اضافه و آن را به‌روزرسانی کنیم:

```
List1 += [2 , 4]
print(List1) >> ['sib' , 'golabi' , 'angoor' , 2 , 4]
```

نکته: در جمع لیست‌ها خاصیت جابه‌جایی وجود ندارد:

```
[2 , 3] + [4 , 5] >>> [2 , 3 , 4 , 5]
[4 , 5] + [2 , 3] >>> [4 , 5, 2 , 3]
```

برای انجام عملیات ریاضی باید با اندیس‌ها کار کنید!

مثال:

```
s=list2[1] + list2[2]       # 2+5=7
```

```
>>> 7
```

del: برای پاک کردن داده‌ها می‌توانید از دستور del استفاده کنید.

```
del list1[1:3]
print(list1) >> ['sib' , 'golabi' , 'angoor']
del(list2) #کلاً list2 را پاک می‌کند
```

(هر متغیر در پایتون، فضایی از حافظه‌ی شما را اشغال می‌کند. برای آزادسازی این فضا باید از del استفاده کنید.)

Method های پرکاربرد قابل استفاده برای داده‌های لیست:

کاربرد	Method
اضافه کردن داده‌ی x به انتهای لیست a	a.append(x)
حذف داده‌ی x از لیست a	a.remove(x)
مشاهده‌ی اندیس داده‌ی x از لیست a	a.index(x)

قرار دادن داده x در اندیس iام	(a.insert(i,x
اضافه کردن لیست b به انتهای لیست a	(a.extend(b
خروج و حذف داده‌ی اندیس i از لیستa. اگر i را وارد نکنید، آخرین داده را حذف می‌کند.	(a.pop(i
کپی کردن لیست a	a.copy

در اینجا Methodها روی متغیر فرضی a انجام شده و شما به جای a باید نام متغیر دلخواه خود را وارد کنید.

نکته خیلی مهم: در سایر زبان‌های برنامه‌نویسی برای کپی کردن مقدار یک متغیر در متغیر دیگر، از = استفاده می‌شود. اما این روش در پایتون می‌تواند کل برنامه‌ی شما را بهم بریزد. چرا؟ بیایید با هم اجرا کنیم. فرض کنید:

```
a1 = [ 1 , 2 , 3]
a2 = a1
a1.pop( )
```

```
print(a1) >>> [1 , 2]
print(a2) >>> [1 , 2]
```

همانطور که دیدید، دستور pop را فقط روی a1 اجرا کردیم، اما اثر آن را درa1 و a2 مشاهده کردیم. حالا دستور زیر را امتحان کنید:

```
a1 = [1 , 2 , 3]
a2 = a1.copy( )
a1.pop( )
```

```
print(a1) >> [1 , 2]
print(a2) >> [1 , 2 , 3]
```

این نکته برای سایر آرایه‌ها که در ادامه به آن‌ها می‌پردازیم نیز صادق است.

Tuple: این دسته از داده‌ها دقیقاً مشابه لیست‌ها هستند، با این تفاوت که نمی‌توانید اعضای آن‌ها به‌روزرسانی کنید. برای تعریف Tupleها می‌توانید از پرانتز استفاده کنید. (یا اصلاً از هیچی استفاده نکنید!)

مثال)

```
a = (1 , 2 , 3)
b = ‘Yellow’ , ‘Blue’ , Red’ , ‘Green’
c = 1 ,
```

اندیس‌گذاری و فراخوانی اعضای Tuple دقیقاً مشابه List است. بنابراین از گفتن مطالب تکراری خودداری می‌کنیم. همانطور که گفتیم، Tupleها قابل به‌روزرسانی نیستند. اما می‌توانید این ویژگی را به دو روش دور بزنید! در روش اول می‌توانید چند Tuple را با + کنار یکدیگر قرار دهید. با این روش فقط می‌توانید به Tuple، داده اضافه کنید و توانایی حذف نخواهید داشت.

مثال)

```
a = ‘a’ , ‘b’
a= (‘c’ , ‘d’ , ‘n’)
a = a + b
```

```
print(a) >>> ('a' , 'b' , 'c' , 'd' , 'n')
```

در روش دوم می‌توانید با دستور Tuple ،list را ابتدا به List تبدیل کرده و سپس تغییرات را اعمال کنید.

```
k = 1 , 2
type(k) >>> <class 'tuple'>
h = list(k)
type(h) >>> <class 'list'>
```

Set: مجموعه‌ها خیلی شبیه List و Tupleها هستند، با این تفاوت که اندیسی در آن‌ها تعریف نمی‌شود. بنابراین داده‌های مجموعه قابل فراخوانی نیستند. اما می‌توانید مانند List آن‌ها را حذف یا به‌روزرسانی کنید. Setها با { } تعریف شده و می‌توانند شامل هر نوع داده‌ای باشند. در جدول زیر تعدادی از Methodهای پرکاربرد برای کار با آن‌ها معرفی شده است.

کاربرد	Methood
اضافه کردن عضو جدید x به مجموعه a	a.add(x)
اضافه کردن اعضای مجموعه‌ی b به مجموعه‌ی a	a.update(b)
ایجاد کپی از مجموعه‌ی a	a.copy( )
حذف عضو x از مجموعه‌ی a	a.remove(x)
حذف عضو x از مجموعه‌ی a	a.discard(x)
حذف آخرین عضو مجموعه‌ی a	a.pop()
بررسی تساوی دو مجموعه	a.isdisjoint(b)
بررسی زیر مجموعه بودن a از b	a.issubset(b)
مجموعه‌ی اجتماع a و b	a.union(b)
مجموعه‌ی اشتراک a و b	a.intersection(b)
مجموعه‌ی تفاضل a و b	a.difference(b)

در جدول بالا برای a.discard و a.remove یک توضیح نوشته شده اما این دو Method یک تفاوت جزئی با هم دارند: در صورتی که داده‌ی x در مجموعه وجود نداشته باشد، با اجرای a.remove(x) با خطا مواجه می‌شوید. اما در a.dicard(x) خطایی دریافت نمی‌کنید.

همچنین برای تبدیل سایر کلاس‌ها (مانند List و Tuple) به مجموعه می‌توانید از دستور `set()` استفاده کنید.

Dictionary: خب تا حالا سه نوع آرایه به شما معرفی کردم. حالا می‌رسیم به آخرین و جالب‌ترین نوع کلاس داده!

می‌توان گفت دیکشنری آزادترین نوع داده است. چرا که علاوه‌بر اعضا، اندیس‌ها را هم می‌توانید آن طوری که می‌خواهید تعریف کنید!! این داده هم مثل setها با { } تعریف می‌شود. اجازه بدید دیکشنری را با یک مثال ببینیم:

```
my_dict={'Brand' : 'Peykan' , 'Model':Javanan' , 'Rang' : 'Gojeyi'}
```

در این مثال، 'Model'، 'Brand' و 'Rang' اندیس‌ها و 'Javanan'، 'Peykan' و 'Gojeyi' اعضای my_dict هستند.

```
my_dict('Model') >>> 'Javanan'
```

حالا بیایید یک عضو جدید هم اضافه کنیم:

```
my_dict['Year']='57'
```

در اینجا 'Year' اندیسی برای '57' تعریف می‌شود.

در این کلاس، اندیس‌ها و اعضا می‌توانند هر نوع داده ای از جمله int، str و ... باشند. در جدول زیر تعدادی از Methodهای پرکاربرد این داده‌ها را با هم ببینیم:

کاربرد	Method
نمایش اندیس‌های دیکشنری a	a.keys()
نمایش اعضای دیکشنری a	a.values()

ایجاد کپی از دیکشنری	a.copy()
عضو اندیس x را برمی‌گرداند.	a.get(x)
عضو اندیس x را پاک می‌کند.	a.pop(x)

برای تبدیل سایر کلاس‌ها به دیکشنری می‌توانید از دستور dict( ) استفاده کنید.

input: دریافت ورودی توسط کاربر

```
ورودی= input('جمله‌ی مورد نظر برای نمایش')
```

مثال) برنامه‌ای بنویسیدکه پس از دریافت نام کاربر، پیغام خوش آمدید را برای او نمایش دهد.

```
s=input('Enter your name: ')
print('Welcome, ' + s)
```

```
>>> Enter your name: Arvin
>>> Welcome, Arvin
```

این دستور تمامی ورودی‌ها را به‌صورت str در نظر می‌گیرد. بنابراین برای محاسبات، باید مقدار دریافت شده را به float، integer یا هر نوعی که نیاز دارید، تبدیل کنید.

خب تا اینجا دستورات مقدماتی، انواع متغیرها و کلاس‌های داده را دیدیم. از این به بعد می‌خواهیم کمی بیشتر وارد برنامه‌نویسی و اصول پشرفته‌ی آن بشویم. این کار را با دستورات شرطی و حلقه‌ها شروع می‌کنیم. (توصیه می‌کنم از این به بعد دستورات رو به‌صورت Batch اجرا کنید تا با اون محیط هم آشنا بشید)

if-elif-else: اولین دستور شرطی که بررسی می‌کنیم، دستور if است. همراه if همیشه از عملگرهای شرطی استفاده شده و در صورتی که شرط برقرار باشد، دستور مورد نظر اجرا خواهد شد. اجازه بدید برگردیم عقب‌تر! اگر یادتان باشد در ابتدای این آموزش گفتیم که پایتون بر خلاف سایر زبان‌های برنامه‌نویسی به فاصله و Space حساس است. این حساسیت اینجا مطرح می‌شود. در واقع در پایتون به جای { } در C++ یا end در MATLAB از فاصله و Space برای تعیین نقطه‌ی پایان if استفاده می‌شود. این یکی از ویژگی‌هایی است که این زبان را نسبت به بقیه ساده‌تر می‌کند. دستورات شرطی if به شکل زیر نوشته می‌شوند:

```
if    : شرط
دستور ۱
دستور ۲
            .
            .
            .
elif :   شرط
دستور ۱
دستور ۲
            .
            .
            .

else :
      دستور
```

روال کار به این صورت است که اولین شرط توسط if بررسی می‌شود. اگر شرایط if برقرار نبود، شرایط جدید elif، و اگر هیچ یک از شروط elif هم برآورده نشد، دستورات درون else اجرا می‌شوند.

اول یک مثال ساده ببینیم. برنامه‌ای بنویسید که دو عدد را دریافت و عدد بزرگتر را نشان دهد:

```
s1=int(input('Enter your first number: '))
s2=int(input('Enter your second number: '))
 if s1 >= s2 :
        print('s1 is greater')
else:
        print('s2 is greater')
```

در این مثال if s1 > s2 جمله‌ی شرطی ما است. پس بعد اتمام آن باید از : استفاده کنیم. سپس برای نوشتن دستورات مربوط به این شرط، در خط بعدی چند تا Space بزنید و دستور را تایپ کنید. (مهم نیست اولش چند تا Space بزنید، مهم اینه دستورات بعدی هم دقیقاً زیر اولین دستور بزنید.) هر کجا که دوباره از سر خط شروع به کدنویسی کنید، از if خارج شدید. این روند برای else و elif (که همون else if خودمونه) هم مشابه است.

در مثال بعدی می‌خواهیم if های تو در تو را بررسی کنیم:

مثال) برنامه‌ای بنویسید که ابتدا یک عدد را دریافت کند. اگر مثبت و زوج بود آن را دو برابر و اگر مثبت و فرد بود، آن را 3 برابر کند. در صورت منفی بودن نیز خود عدد را نمایش دهد.

```
x = int (input('Enter your number: '))
if x > 0 :
       if x % 2 == 0  :
                x = x * 2
                print ('x was positive & even, so result is: ', x)
       else:
                x *= 3
                print ('x was positive & odd, so result is:' , x)
elif x < 0  :
        print ('x was negative, so result is: ', x)
else :
         print('x is Zero')
```

این برنامه با توجه به این که دارای دستورات متعدد و شرطی می‌باشد، بهتر است به صورت batch نوشته شود. در این مثال به space ها توجه کنید! با توجه به فاصله‌های داده شده، پایتون if دوم را در صورت تایید if اول بررسی خواهد کرد. یعنی اگر عدد مثبت باشد، زوج بودن آن بررسی می‌شود. در صورت منفی بودن، زوج بودن آن بررسی نخواهد شد. اجازه بدید این برنامه را در رزبری‌پای اجرا کنیم:

ابتدا یک فایل text با دستور زیر ایجاد کرده و نام آن را test با پسوند.py در نظر می‌گیریم.

```
$ nano test.py
```

سپس کدهای مورد نظر را در این فایل می‌نویسیم.

```
pi@Sanatbazar: ~
  GNU nano 3.2                    test.py                    Modified

x = int (input('Enter your number: '))
if x > 0 :
     if x % 2 == 0  :
            x = x * 2
            print ('x was positive & even, so result is: ', x)
     else:
            x *= 3
            print ('x was positive & odd, so result is:' , x)
elif x < 0  :
      print ('x was negative, so result is: ', x)
else :
       print('x is Zero')

^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify
^X Exit     ^R Read File ^\ Replace  ^U Uncut Tex ^T To Spell
```

SANATBAZAR.COM

پس از ذخیره، برنامه را با python3 اجرا کنید:

```
pi@Sanatbazar: ~
pi@Sanatbazar:~ $ python3 test.py
Enter your number: 4
x was positive & even, so result is:  8
pi@Sanatbazar:~ $ python3 test.py
Enter your number: 5
x was positive & odd, so result is: 15
pi@Sanatbazar:~ $ python3 test.py
Enter your number: -1
x was negative, so result is:  -1
pi@Sanatbazar:~ $ python3 test.py
Enter your number: 0
x is Zero
pi@Sanatbazar:~ $
```

SANATBAZAR.COM

```
$ python3 test.py
```

while: تا زمانی که شرط برقرار است، دستورات حلقه را اجرا کن! طرز نوشتن آن به صورت زیر است:

```
while : شرط

دستورات
```

مثال) اعداد ۱ تا ۱۰ را نمایش دهید:

```
i=1
```

```
while i <= 10 :
        print(i)
        i += 1
```

توجه: در صورت نیاز متغیر شرط حلقه را به‌روزرسانی کنید (یعنی i = i + 1)، در غیر این صورت حلقه‌ی شما به صورت بی‌نهایت اجرا می‌شود.

range: تابع range(a,b,c) اعداد بین a و b را با گام‌های c تا c تا می‌شمارد. اگر c را وارد نکنید، گام‌ها به صورت پیش‌فرض ۱ در نظر گرفته می‌شوند. در صورتی که a را وارد نکنید، پایتون به‌صورت پیش‌فرض، اعداد را از صفر می‌شمارد. کاربرد اصلی این تابع در حلقه‌های for است.

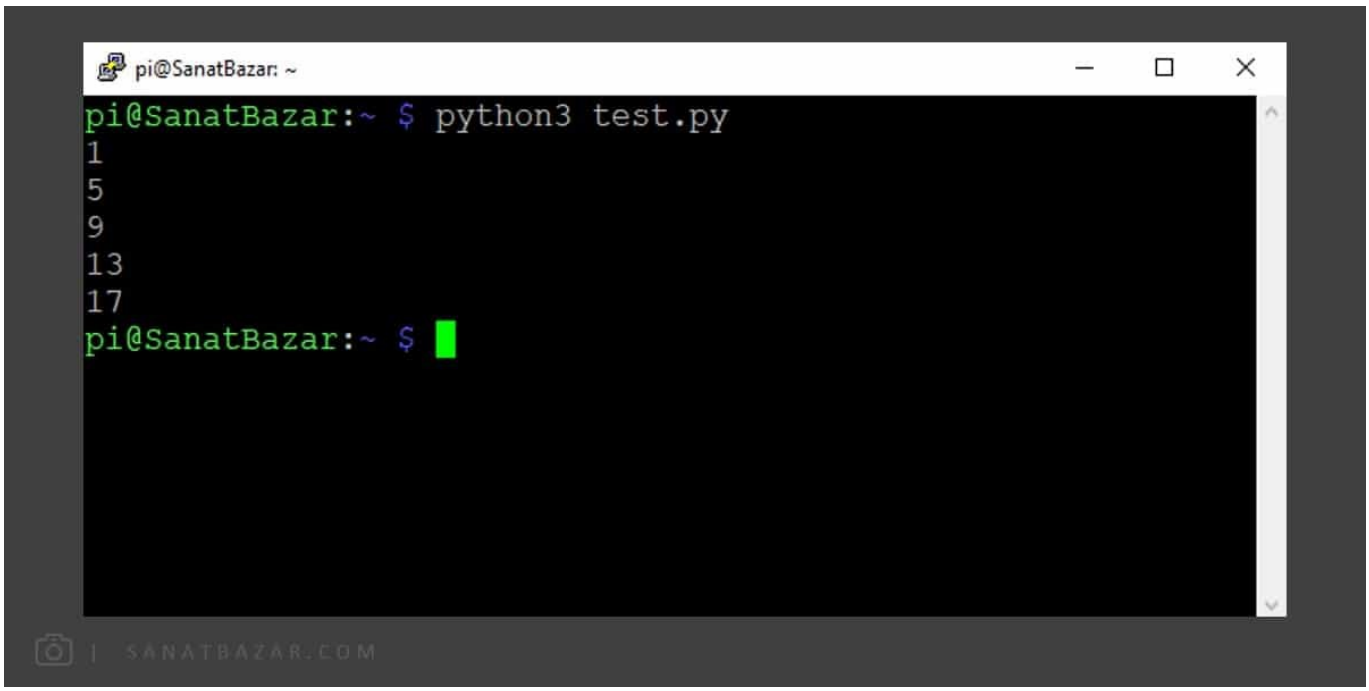
for-in: دستور for برای تکرار و شمردن یک آرایه استفاده می‌شود. این آرایه می‌تواند هر داده‌ای از جمله tuple، list یا dictionary باشد.

```
for متغیر حلقه in آرایه:
دستورات حلقه
```

این حلقه تا زمانی که تمامی اعضای موجود در آرایه توسط متغیر حلقه شمرده شوند، ادامه پیدا می‌کند.

مثال) نمایش اعداد ۱ تا ۲۰ با گام‌های ۴ تایی:

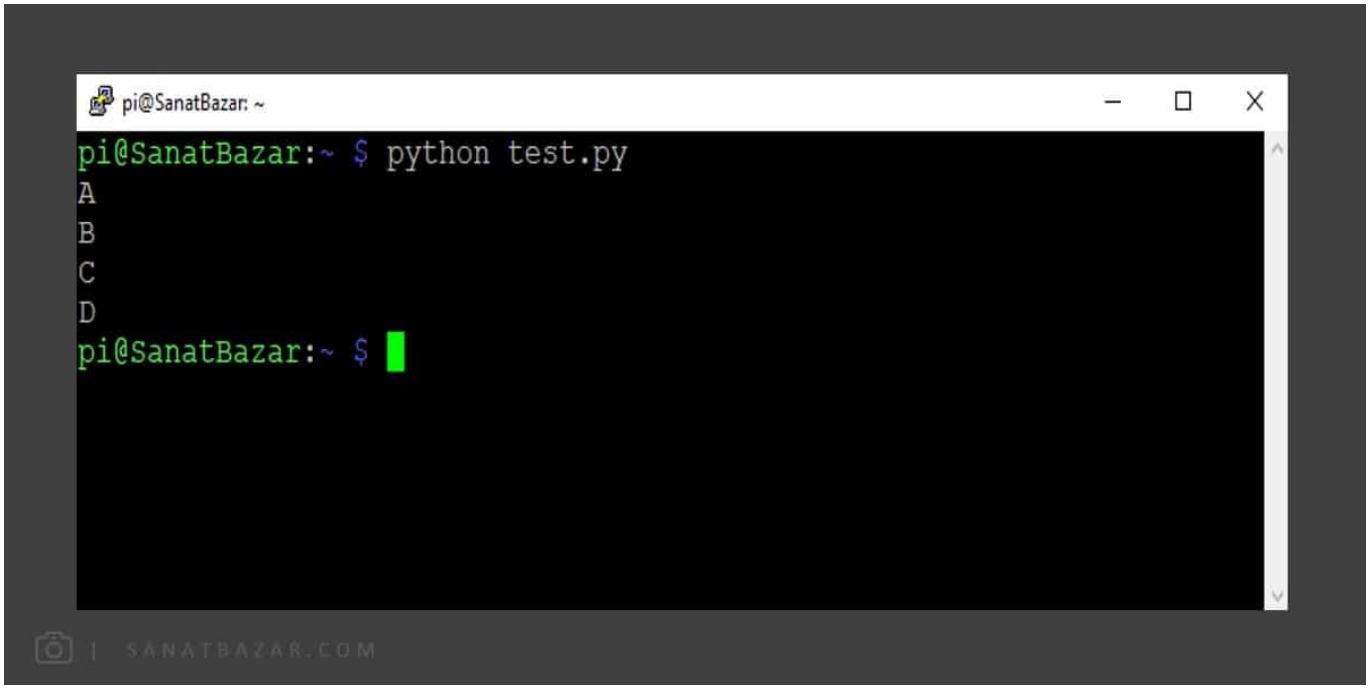
```
for i in range(1,20,4) :
        print(i)
```



شمارنده حتماً لازم نیست عدد باشد. می‌توانید for را روی هر داده‌ای از جمله String هم اجرا کنید.

مثال)

```
t = ['A' , 'B' , 'C' , 'D']
for i in t :
        print (i)
```



try-except: یکی از خصوصیت‌های فوق‌العاده‌ی پایتون، دستور try-except است.

```
try :
        دستورات
except :
```

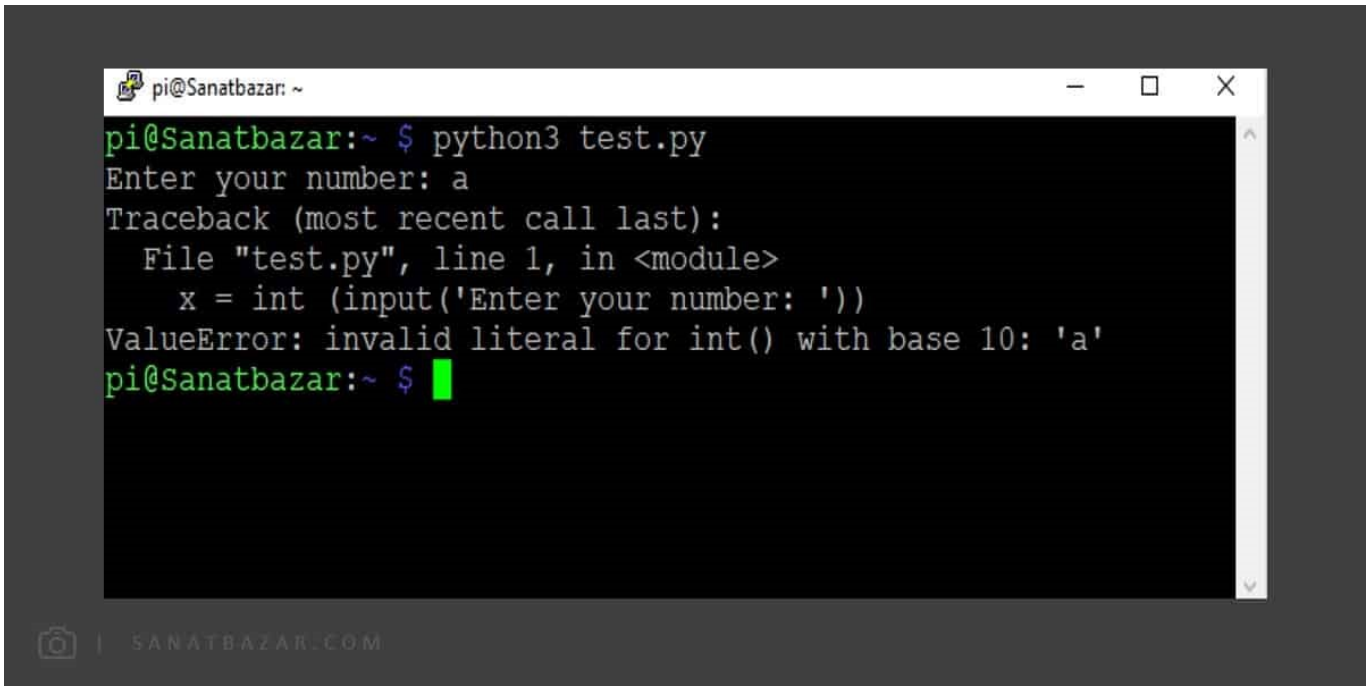
```
        دستورات
```

در این روش اگر برنامه با خطایی همراه نبود، دستورات try و در صورت بروز خطا، دستورات except اجرا می‌شود. بنابراین می‌توانید خطاهای احتمالی را پیش‌بینی کرده و عملکرد مناسب برنامه را برای آن تعریف کنید. حتی می‌توانید برنامه را طوری بنویسید که نسبت به خطاهای مختلف، عملکرد متفاوتی نیز داشته باشد.

مثال) برنامه‌ای بنویسید که یک عدد را از کاربر گرفته و قرینه‌ی آن را نمایش دهد:

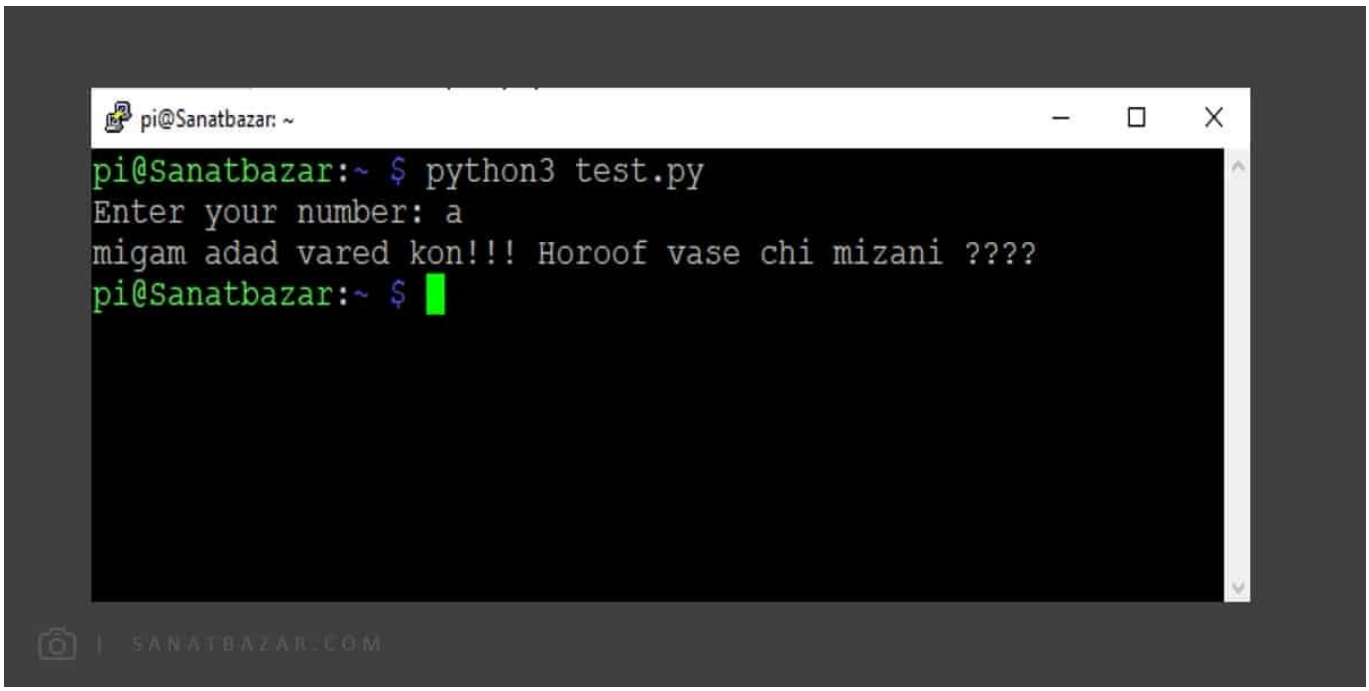
```
x =int(inpu('Enter your number'))
print(-x)
```

فکر می‌کنید اگر به جای وارد کردن عدد، یک کارکتر مثلاً a را وارد کنید، چه جوابی می‌گیرید؟



خب همانطور که انتظار داشتیم با خطا مواجه شدیم. حالا می‌توانیم این خطا را با قرار دادن دستوری مناسب در except برطرف کنیم:

```
try:
      x =int(inpu('Enter your number'))
      print(-x)
except:
      print('migam adad vared kon!!! Horoof vase chi mizani ????')
```



در کنار try و except، می‌توانید از دستور finally هم استفاده کنید. این دستور تحت هر شرایطی (چه خطا باشه چه نباشه) اجرا خواهد شد. پس اگر خطا نداشته باشیم، try و finally و اگر خطا داشتیم، except و finally اجرا می‌شوند.

KeyboardInterrupt: این دستور، بیانگر وقفه‌های Ctrl+C و Ctrl+Z می‌باشد که در بخش آموزش لینوکس گفتیم از آن‌ها برای خروج از برنامه‌ی در حال اجرا استفاده می‌شود. در پایتون با استفاده از KeyboardInterrupt و try-except می‌توانید رفتار برنامه را در صورت اعمال وقفه تعیین کنید.

مثال)

```
try:
دستورات
except KeyboardInterrupt :
print("Interrupt!")
```

برای مثال در برنامه‌ی بالا ، اگر وقفه اعمال شود، پیغام Interrupt! برای کاربر نمایش داده می‌شود.

function: توابع، مجموعه‌ای از دستورات هستند که در صورت فراخوانی اجرا خواهند شد. برای تعریف توابع از def استفاده می‌کنیم. فرمت نوشتن آن‌ها به صورت زیر است:

```
def : (ورودی‌ها) نام تابع
دستورات
        return خروجی
```

مثال) تابعی بنویسید که یک عدد را دریافت کرده سپس تعیین کند زوج است یا فرد؟

نام تابع را even در نظر می‌گیریم:

```
def even(x) :
    if x % 2 == 0 :
        return 'even'
    else:
```

```
            return 'odd'
```

```
print(even(2)) >>> even
print(even(3)) >>> odd
```

برای تعریف مقدار اولیه، می‌توانید آن را در تعریف تابع به ورودی اعمال کنید. مثلاً در مثال قبلی می‌خواهیم مقدار اولیه را صفر تعریف کنیم: def even(x=0):

در صورت نیاز می‌توانید چندین ورودی را به تابع اعمال کنید. برای این منظور می‌توانید یا از List استفاده کنید یا مشابه برنامه‌ی زیر، مقادیر را وارد کنید:

```
def name(first = 'arvin' ,  last = 'ghahremani') :
        print('my first name is ' + first + ' and my last name is ' + last)
```

در برنامه‌ی فوق، مقادیر پیش فرض arvin و ghahremani تعریف شده است.

متغیر Global: متغیرهایی که درون تابع مقداردهی و تعریف می‌شوند، تنها درون همان تابع قابل شناسایی اند. برای این که از این متغیرها بتوان در برنامه‌ی اصلی یا سایر توابع نیز استفاده کرد، باید درون هر تابع، آن‌ها را global تعریف کنیم. برای این کار کافیست پیش از مقداردهی، عبارت global را قبل از نام متغیر وارد کنید.

مثال)

```
def test( ) :
        global a
        a=2
        return a
```

خب تا اینجا توابعی که فراخواندیم در خود برنامه نوشته شده بود. حالا اگر تابع در جایی خارج از برنامه نوشته شده باشد، چه کار باید کرد؟

Module: برای این که تابع ما قابلیت اجرا شدن درون هر برنامه‌ای را داشته باشد، باید آن را جداگانه به‌صورت ماژول نوشته و در محلی ذخیره کنیم. پس در گام اول، محلی را برای ذخیره‌ی توابع خود ایجاد کنید. من برای این کار، یک دایرکتوری به نام python در ~ ساختم. در قدم بعدی، برای فراخوانی ماژول، باید آدرس این دایرکتوری را به پایتون معرفی کنم. آدرس فایل توابع خود را به شکل زیر در Command-Line وارد کنید:

```
$ export PYTHONPATH = 'آدرس دایرکتوری توابع'
```

سپس تابع مورد نظر را نوشته و آن را با فرمت .py ذخیره کنید. حالا در برنامه‌ی اصلی که می‌خواهید تابع مورد نظرتان فراخوانی شود، دستور زیر را وارد کنید:

خروجی= نام فایل تابعی که ذخیره کردید . نام تابع (ورودی)

برای مثال من تابع even را با نام iseven,py ذخیره کردم. حالا می‌خوام نتیجه تابع را برای ورودی ۳ در خروجی y بریزم. پس باید دستور زیر را وارد کنم:

```
y = iseven . even(3)
```

module در واقع مانند یک کتابخانه عمل می‌کند. علاوه‌بر توابع، می‌توانید متغیرهای module را هم فراخوانی کنید.

خروجی= نام فایل تابعی که ذخیره کردید . نام متغیر

در نسخه‌ی pc هم ابتدا مشابه لینوکس، یک Folder برای moduleهای خود بسازید، آدرس فایل را با دستور زیر به پایتون معرفی کرده و سپس با import آن را فراخوانی کنید:

```
import sys
sys.path.append('آدرس مورد نظر')
```

توجه داشته باشید که آدرس باید با slash جدا شود. مثال:

```
sys.path.append('C:/Users/Arvin-PC/Documents/Python/')
```

ایجاد Class و Object: تقریباً همه چیز در پایتون به صورت شی و Method نوشته شده است. اگر دقت کرده باشید، تا الان اکثر دستوراتی که بررسی کردیم، قالب object.method( ) داشت. برای ایجاد Objectها از دستور Class استفاده می‌کنیم:

مثال) کلاسی تعریف کنید که دارای دو ویژگی نام و سن شما باشد:

```
class Test() :
      name = 'arvin'
      age = '24'
```

برای مشاهده‌ی نتیجه، دستور زیر را اجرا می‌کنیم:

```
a = Test()
a.name >>> 'arvin'
a.age >>> '24'
```

اگر می‌خواهید یک کلاس کلی و بدون property داشته باشید، می‌توانید از دستور pass استفاده کنید. در این صورت یک class خام ایجاد کرده و می‌توانید هر ویژگی که خواستید به آن اضافه کنید:

```
class Test :
      pass

a = Test()
a.name = 'mohammad'
a.job = 'teacher'
```

به‌صورت استاندارد، در پایتون نام متغیرها و توابع با حروف کوچک و نام کلاس‌ها با حروف بزرگ شروع می‌شود. همچنین اگر نام دو بخشی باشد، بخش دوم در متغیرها و توابع با _ و در کلاس‌ها با حروف بزرگ از بخش اول جدا می‌شود. مثلاً متغیر یا تابع first_name و کلاس FirstName

تابع __init__( ): از این تابع برای تخصیص مقادیر به دستورات Class استفاده می‌شود. __init__ به صورت خودکار پس از فراخوانی Class اجرا می‌شود.

مثال) با اسفاده از Class، برنامه‌ای بنویسید که دو عدد را گرفته و حاصل ضرب آن‌ها را نمایش دهد:

```
class Mul:
       def __init__(self , x , y):
                      self.x=x
                      self.y=y
                      self.result=self.x*self.y
```

برای اجرای Class کافیست با دستور زیر، یک Object با نام دلخواه از آن ایجاد کنیم:

```
s = Mul (2 , 4)
print (s . result) >>> 8
```

می‌توانید دستور print را در کد class قرار دهید تا پس از اجرای آن، خروجی به صورت خودکار نمایش داده شود.

 | این محتوا توسط صنعت بازار تولید یا برگردان شده است - کپی رایت © 2019

در صورت نیاز می‌توانید مقادیر اولیه را در تابع init قرار دهید:

```
class Mul:
        def __init__(self , x = 3 , y = 5):
                        self.x=x
                        self.y=y
                        self.result=self.x*self.y
                        print(self . result)
```

در این صورت اگر مقادیر x وy را وارد نکنید، پاسخ 15 نمایش داده خواهد شد.

```
s = Mul ( )
print (s . result) >>> 15
print(s . x) >>> 3
print(s . y) >>> 5
```

احتمالاً می‌پرسید self این وسط چیه و چی کار می‌کنه؟ همانطور که می‌بینید برای استفاده از این کلاس، ابتدا s را تعریف کردیم. یا برگردیم عقب‌تر. اگر یادتان باشد در مثال نام و سن، برای اجرا از متغیر a استفاده کردیم. در تابع init اولین متغیری که تعریف می‌کنید، در واقع جای همین s و a می‌نشیند. مثلاً در اینجا گفتیم print (s . result)). اگر به کد درون Mul توجه کنید، به جای متغیر s، از self استفاده کردیم. (یعنی اونجا تعریف کردیم برای متغیر فرضی self که نمیدونیم چیه و کاربر قراره براش یه اسمی تعریف کنه، چه property هایی وجود داشته باشه!) به طور کلی از self برای دسترسی به متغیرهای درون شی استفاده می‌شود. شما می‌توانید به جای self از هر اسم دلخواهی استفاده کنید. اما به خاطر داشته باشید که اولین متغیر در پرانتز، نقش self را برای شما بازی می‌کند. (در برنامه‌نویسی استاندارد از نام self استفاده می‌شود)

Method: Method ها در واقع همان توابع درون Class ها هستند. در هر کلاس بسته به نیاز خود می‌توانید توابع مختلفی تعریف کنید. به مثال زیر توجه کنید:

مثال) با استفاده از کلاس، یک ماشین حساب ساده طراحی کنید:

```
class Calc:
    def __init__(self,x,y):
        self.x=x
        self.y=y

    def add(self):
        self.result=self.x+self.y
        print(self.result)

    def sub(self):
        self.result=self.x-self.y
        print(self.result)

    def mul(self):
        self.result=self.x*self.y
        print(self.result)
```

```
    def div(self):
        self.result=self.x/self.y
        print(self.result)
```

خب در این مثال هر کدام از اعمال ریاضی، یک Method برای Calc هستند.

Inheritance (وراثت):

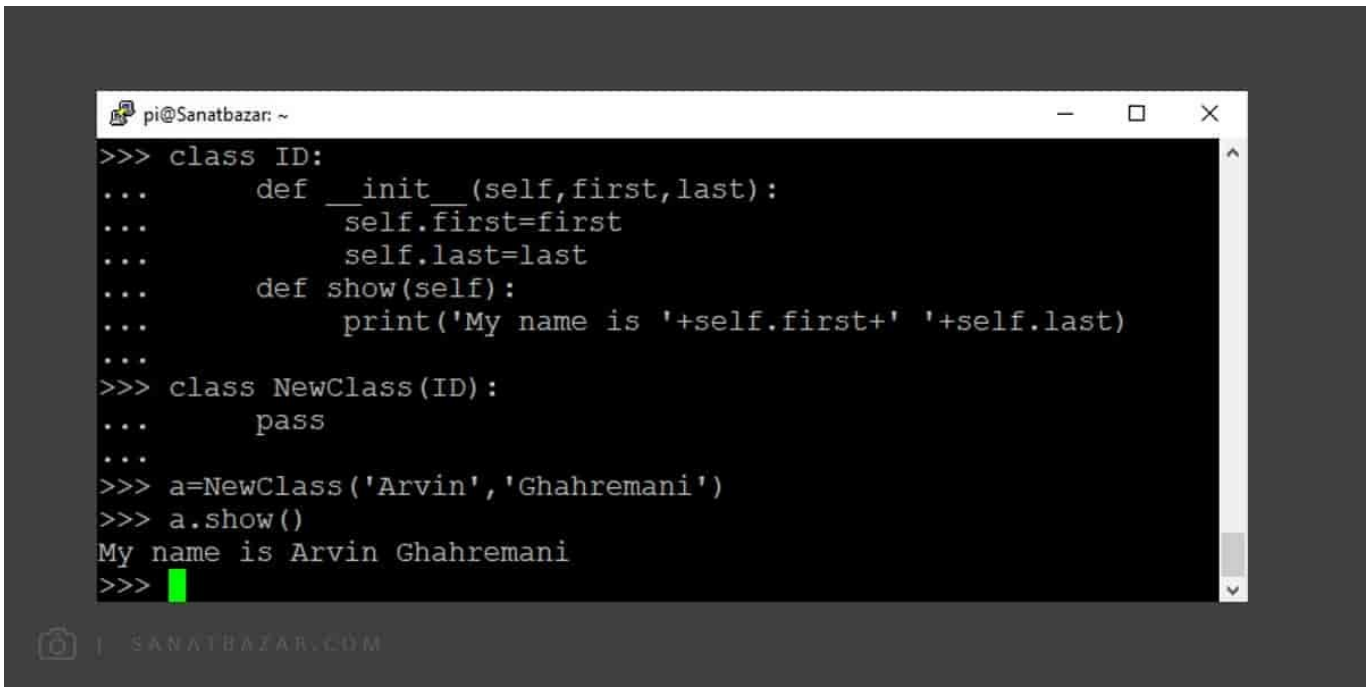
با استفاده از ویژگی وراثت پایتون، می‌توان Methodها و Propertyهای یک Class را به سادگی و با یک دستور به Class جدید انتقال داد. در واقع Class جدید، Methodها و Propertyهای کلاس والد را به ارث می‌برد. این کار از تلف شدن وقت و کدنویسی اضافه جلوگیری می‌کند. به Class اصلی، Parent Class و به Class مشتق شده، Child Class گفته می‌شود. برای این کار به مثال زیر توجه کنید:

```
class ID :
      def __init__(self , first , last) :
              self . firstname = first
              self . lastname = last

      def show(self) :
             print('My name is ' + self . firstname + ' ' + self . lastname)

class NewClass(ID):       # نام کلاس Parent در پرانتز کلاس Child نوشته می‌شود
        pass

x = NewClass('arvin' , 'ghahremani')
x . show()
```



همانطور که مشاهده می‌کنید، دستورات را در ID نوشتیم، اما با اجرای NewClass آن‌ها را اجرا کردیم.

Ctrl + F6: از این کلید برای پاک کردن مقدار تمام متغیرهای استفاده شده در Shell استفاده می‌شود.

کتابخانه‌های پایتون: یکی از ویژگی‌های مثبت پایتون که آن را از سایر زبان‌های برنامه‌نویسی متمایز می‌کند، کتابخانه‌های قدرتمند و به‌روز این زبان است. در این قسمت ابتدا می‌خواهیم شیوه‌ی فراخوانی و استفاده از آن‌ها را ببینیم، سپس کتابخانه‌های پرکاربرد math، time و RPi.GPIO را که برای برقراری ارتباط با پین‌های رزبری‌پای استفاده می‌شود، بررسی می‌کنیم.

import: این دستور کتابخانه‌های موجود در سیستم شما را فراخوانی می‌کند. بنابراین برای استفاده ار کتابخانه‌ها کافیست نام آن‌ها را import کنید.

```
import نام کتابخانه
```

علاوه‌بر این می‌توانید تنها تابع دلخواه خود را از کتابخانه فراخوانی کنید. فرمت فراخوانی با این روش، به صورت زیر است:

```
from نام کتابخانه import نام تابع مورد نظر
```

گاهی اوقات نام کتابخانه‌ها طولانی و سخت است و با توجه به فرمت استفاده از توابع، نوشتن نام آن‌ها سخت و وقت‌گیر خواهد بود. برای حل این مشکل می‌توانید کتابخانه‌ی مورد نظر خود را با نام دلخواهی که راحت‌ترید، فراخوانی کنید. برای این کار باید از فرمت دستوری زیر استفاده کنید:

```
import نام کتابخانه as نام جدید
```

کتابخانه‌ی time: همانطور که گفتیم برای استفاده از کتابخانه‌ها و Moduleها در پایتون، ابتدا باید آن‌ها را import کنیم. پس برای استفاده از توابع زمان، ابتدا دستور زیر را اجرا می‌کنیم:

```
import time
```

time.time( ): این تابع مدت زمان طی شده از ساعت 00:00 اول ژانویه سال 1970 (UNIX Epoch Time) را با استاندارد UTC نشان می‌دهد. شاید بپرسید خب این دستور به چه دردی می‌خوره؟ زمانی که اختلاف زمانی بین دو دستور مهم است می‌توانید این تابع را در دو جای مختلف اجرا و اختلاف آن‌ها را محاسبه کنید.

UTC یک معیار زمان برای تعیین ساعت‌ها و اختلافات زمانی در سراسر جهان است.

time.sleep( ): این دستور به اندازه‌ی مقدار ورودی، تاخیر ایجاد می‌کند.

مثال) تاخیر ۲ ثانیه:

```
time.sleep(2)
```

کتابخانه‌ی math: همانطور که از نام آن پیداست، از این کتابخانه برای انجام عملیات ریاضی استفاده می‌شود.

کاربرد	دستور
X به توان y	(math.pow(x,y
سینوس x	(math.sin(x
کسینوس x	(math.cos(x
لگاریتم x بر مبنای y	(math.logy(x
e به توان x	(math.exp(x
ریشه‌ی دوم x	(sqrt(x

math شامل توابع ریاضی فراوانی است که در صورت نیاز می‌توانید آن‌ها را در اینترنت جست‌وجو کنید.

اگر می‌خواهید هر بار برای استفاده از توابع این کتابخانه، math. را وارد نکنید، به جای from math import ،import math * را وارد کنید.

کتابخانه‌ی RPi.GPIO: از این کتابخانه برای تنظیمات و دسترسی به GPIO های رزبری‌پای استفاده می‌شود. برای سادگی در نوشتن دستورات، این ماژول را به صورت زیر فراخوانی می‌کنیم:

```
import RPi.GPIO as GPIO
```

GPIO.setmode( ): با این دستور، نحوه‌ی شماره‌گذاری پین‌های رزبری‌پای را تعیین می‌کنید. برای آشنایی با شماره‌گذاری، می‌توانید به بخش مشخصات و معرفی برد رزبری پای Raspberry Pi 4 مراجعه کنید.

```
GPIO.setmode(GPIO.BCM)              #  BCM شماره گذاری با استاندارد
GPIO.setmode(GPIO.BOARD)        #  BOARD شماره گذاری با استاندارد
GPIO.setup( ): با این دستور تعیین می‌کنید پین مورد نظر شما ورودی باشد یا خروجی.
GPIO.setup(18 , GPIO.OUT)  # انتخاب پین شماره‌ی 18 به عنوان خروجی
GPIO.setup(18 , GPIO.IN)       # انتخاب پین شماره‌ی 18 به عنوان ورودی
GPIO.output( ): تعیین مقدار خروجی پین مورد نظر
GPIO.output(18 , GPIO.HIGH)        #(یک منطقی)  HIGH  قرار دادن پین 18 روی حالت
GPIO.output(18 , GPIO.LOW)         #(صفر منطقی)  LOW  قرار دادن پین 18 روی حالت
```

GPIO.cleanup( ): پس از اجرا و اتمام یک برنامه، بهتر است تنظیمات GPIO را به حالت اولیه خود بازگردانید. برای این کار در انتهای برنامه می‌توانید از دستور GPIO.cleanup( ) استفاده کنید. این دستور پین‌ها را در حالت ورودی قرار می‌دهد. (برای جلوگیری از خطر اتصال کوتاه و آسیب دیدن برد). توجه داشته باشید که این دستور فقط روی پایه‌های استفاده شده در همان برنامه عمل می‌کند.

برای آشنایی بیشتر با GPIOهای رزبری‌پای و نحوه‌ی خواندن و نوشتن روی آن‌ها، می‌توانید به بخش آموزش راه‌اندازی GPIO و پین‌های رزبری‌پای مراجعه کنید.

pip: ار pip برای مدیریت پکیج‌ها و ماژول‌های (کتابخانه‌های) پایتون استفاده می‌شود. از python 3.4 به بعد، این دستور به‌صورت پیش‌فرض نصب شده است. اگر از نسخه‌های قدیمی‌تر استفاده می‌کنید، با دستور زیر می‌توانید pip را نصب نمایید.

```
$ sudo apt-get install python-pip          # python 2 برای
$ sudo apt-get install python3-pip      # python 3  برای
```

با استفاده از این دستور می‌توانید کتابخانه‌های نوشته شده توسط افراد دیگر را import کرده و از دستورات آماده‌ی آن‌ها استفاده کنید. در بخش‌های بعدی آموزش از این دستور پرکاربرد برای راه‌اندازی سنسور و ماژول‌های مختلف استفاده می‌کنیم. پس اگر برای برنامه‌ی شما کتابخانه‌ای در پایتون نیست، می‌توانید با این دستور آن را

دانلود و نصب کنید.

```
Pip install کتابخانه برای نصب #  نام ماژول
Pip install --upgradeکتابخانه # کردن کتابخانه برای upgrade نام ماژول
Pip uninstall کتابخانه برای حذف # نام ماژول
```

help( ): خب رسیدیم به آخرین دستور پایتون در این آموزش! برای مشاهده‌ی اطلاعات و help هر دستور پایتون، کافیست در Shell پایتون بنویسید help( ) و سپس نام دستور مورد نظر خود را وارد کنید.

نتیجه‌گیری

در این قسمت با دستورات مهم و کاربردی زبان برنامه‌نویسی پایتون آشنا شدید. خبر خوب این که از حالا به بعد، اطلاعات لازم برای انجام پروژه‌های عملی را دارید!!

می‌دانم که این قسمت خیلی طولانی شد و حتماً تا الان خیلی خسته شدید. اما باور کنید ارزش داشت. این نکات، نکات اساسی و پایه‌ای پایتون بودند که سعی شد تا حد امکان در عین کامل و روان بودن، به صورت خلاصه بیان شوند. با تسلط بر دستورات گفته شده، به راحتی می‌توانید این مسیر را ادامه داده و پایتون را تخصصی‌تر در زمینه‌ای که به آن نیاز دارید، یاد بگیرید. حالا که پایتون و لینوکس بلدید، کاملاً برای انجام پروژه‌های عملی و DIY آماده‌اید. ما هم قصد داریم از این قسمت به بعد، شما را با ماژول‌ها و پروژه‌های پرکاربرد رزبری‌پای آشنا کنیم. با من همراه باشید که در قسمت بعدی، اولین پروژه را با ماژول بسیار کاربردی دوربین رزبری‌پای شروع کنیم.

نظرات شما باعث بهبود محتوای آموزشی ما می‌شود. اگر این آموزش را دوست داشتید، همین‌طور اگر سوالی در مورد آن دارید، از شنیدن نظراتتان خوشحال خواهیم شد.