

پایتون مقدماتی



فهرست مطالب

6	آشنایی با پایتون
10	قابلیت های زبان برنامه نویسی پایتون
11	دانلود و نصب پایتون
15	برنامه یا PROGRAM چیست؟
17	فرآیند DEBUGGING چیست
18	متغیر ها (VARIABLES)
19	قواعد نام گذاری متغیر ها
21	استفاده از SCRIPT MODE
25	کامنت گذاری در SCRIPT
26	انواع ERROR ها در پایتون
27	انواع DATA در پایتون
28	اعداد در پایتون
28	انواع اعداد در پایتون

29	تبدیل نوع داده های عددی به یک دیگر
30	عملگر ها در پایتون
30	عملگر های ریاضی (ARITHMETIC OPERATORS)
32	ترتیب عملگر های ریاضی در محاسبات
33	عملگر های مقایسه ای (RELATION OPERATORS)
35	عملگر های تخصیص یا انتساب (ASSIGNMENT OPERATORS)
36	عملگر های منطقی (LOGICAL OPERATORS)
38	عملگر های عضویت (MEMBERSHIP OPERATORS)
39	عملگر های هویت (IDENTITY OPERATORS)
40	رشته ها (STRINGS) در پایتون
41	SLICING در پایتون
42	به هم چسباندن رشته ها (STRING CONCATENATION)
42	تکرار رشته ها (REPETITION)
43	جستجوی کارکتر خاص در STRING
44	کاراکتر های کنترلی (ESCAPE CHARACTERS) در پایتون
45	رشته های خام (ROW STRING)
45	استفاده از متغیر ها در STRING :
46	F STRING در پایتون
47	آشنایی به STRING METHODS

50	لیست (LIST) در پایتون
51	دسترسی به عناصر داخل لیست
52	تغییر عناصر داخل لیست
52	LIST METHODS
55	تاپل ها (TUPLES)
56	تبدیل تاپل به لیست
57	دیکشنری (DICTIONARY)
57	دسترسی به عناصر دیکشنری
58	تغییر یا اضافه کردن عناصر دیکشنری
58	حذف عناصر دیکشنری
59	ساخت دیکشنری تو در تو
60	ست (SET)
61	دستورات شرطی در پایتون
61	دستور IF
62	دستور ELSE
63	دستور ELIF
64	حلقه ها در پایتون

64	حلقه FOR
64	استفاده از حلقه FOR در LIST
64	استفاده از حلقه FOR در STRING
65	استفاده از تابع RANGE در حلقه FOR
65	ایجاد لیستی از آدرس های IP با استفاده از حلقه FOR و تابع RANGE
65	استفاده از ELSE در حلقه FOR
66	حلقه WHILE
67	استفاده از ELSE در حلقه WHILE
68	حلقه های تو در تو (NESTED LOOP)
68	NESTED IF LOOP
68	NESTED FOR LOOP
69	NESTED WHILE LOOP
69	استفاده ترکیبی از حلقه های تو در تو
70	BREAK و CONTINUE در حلقه ها
71	استفاده از PASS
72	مدیریت خطا در پایتون
73	استفاده از ELSE به همراه TRY
73	استفاده از FINALLY
74	آشنایی با توابع

75	ایجاد FUNCTION با پارامتر ورودی
75	دستور RETURN
76	اعمال مقادیر پیشفرض برای تابع
76	NAME SPACES

آشنایی با پایتون

پایتون (Python) یک زبان برنامه‌نویسی قدرتمند، سطح بالا، شی گرا و حرفه ایست که روز به روز در حال گسترش بوده و برنامه نویسان بیشتری را جذب خود می کند

یادگیری کم دردرس پایتون به برنامه نویسان اجازه داده است با صرف کمی وقت و تلاش، اصول اولیه این زبان را یاد گرفته و اولین برنامه کاربردی خود را با این زبان طراحی کنند، تعداد کلمات کلیدی در پایتون کم، ساده و کاملاً قابل درک است این موضوع فهم و یادگیری این زبان را برای کاربران تازه کار بسیار ساده کرده است.

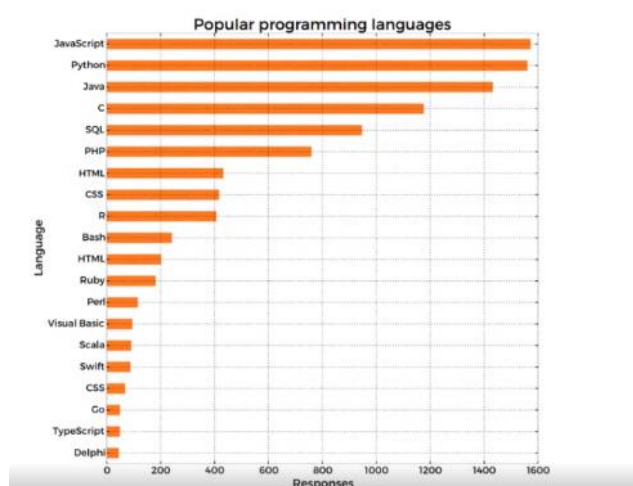
در پایتون، برنامه نویسان بدون مشکل خاصی قادرند منطق دستورات را به شکل صریح و روشنی درک کنند و برای همکاران خود شرح دهند، البته که این سادگی باعث نشده پایتون قابلیت های کمتری نسبت به سایر زبان های برنامه‌نویسی داشته باشد، از این زبان می توان برای ساخت برنامه های کاملاً حرفه ای با کیفیت بالا و ساخت بازی های رایانه ای استفاده کرد.

زبان برنامه‌نویسی پایتون در سال ۱۹۹۱ میلادی توسط یک برنامه نویس هلندی به نام [خیدو فان روسوم](#) ایجاد شد، این زبان با قابلیت های فراوان و شگفت انگیزی که دارد تحولی در دنیای برنامه‌نویسی به وجود آورده است، از توسعه ی برنامه های تحت وب گرفته تا ایجاد بازی های رایانه ای!

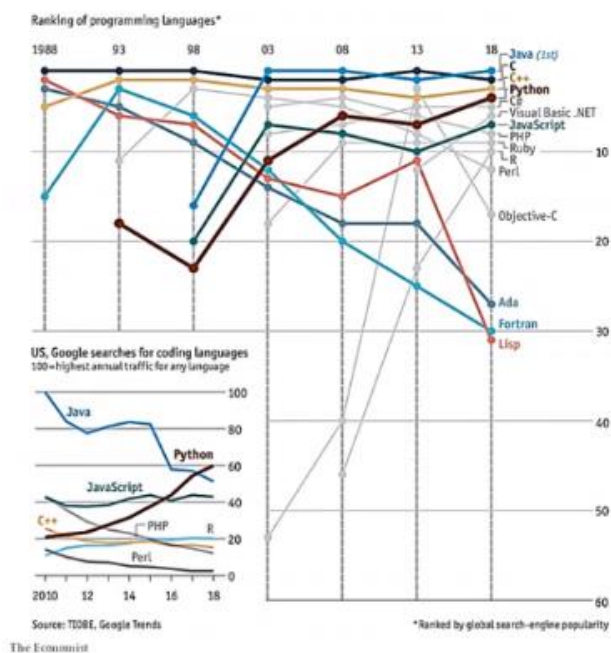
زبان برنامه نویسی پایتون یکی از قدرتمندترین و محبوب ترین زبان های برنامه نویسی می باشد و طبق آمار IEEE در سال ۲۰۱۸ همانطوری که در تصویر زیر مشاهده می کنید پایتون در بین زبان های برنامه نویسی در بالاترین رتبه قرار دارد:



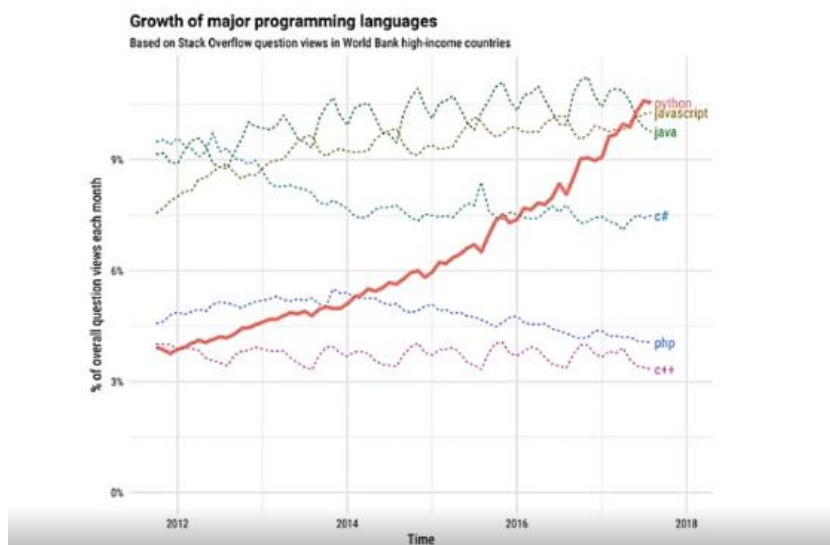
از طرف دیگر طبق گزارش Packet در سال ۲۰۱۸ نیز می توان نتیجه گرفت که زبان برنامه نویسی پایتون در بین زبان های برنامه نویسی دیگر رتبه خوبی را کسب کرده است:



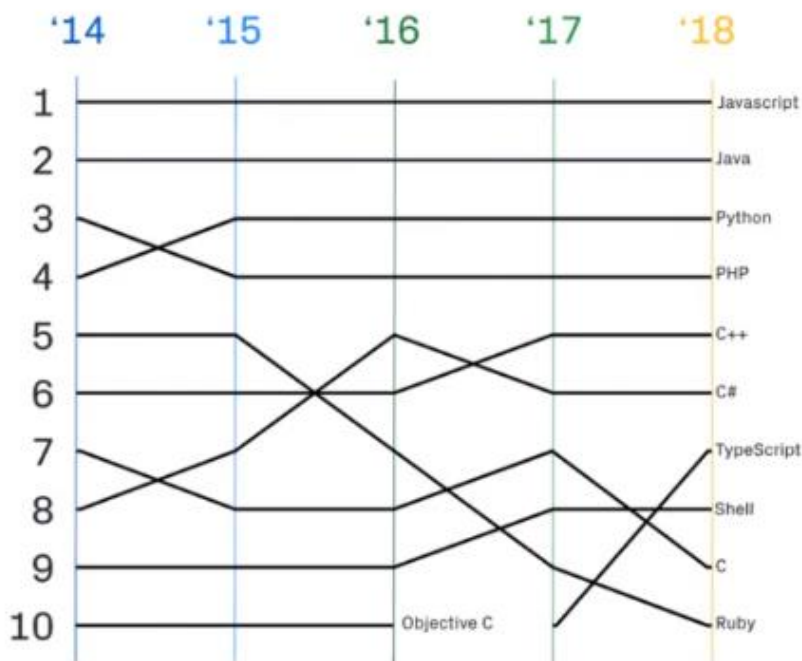
سایت GitHub نیز در گزارشی رتبه زبان های برنامه نویسی را به صورت زیر اعلام کرد:



یکی دیگر از گزارش هایی که می توان در زمینه محبوبیت زبان برنامه نویسی پایتون بررسی کرد گزارش سایت Stack Over Flow می باشد که یک سایت بزرگ در زمینه پرسش و پاسخ می باشد:



همین طور شما می توانید گزارشی را که Google از میزان جستجو هایی که در Google برای زبان های برنامه نویسی شده است را هم مشاهده کنید:



بر طبق گزارش های ارائه شده می توان نتیجه گرفت که زبان برنامه نویسی پایتون جز چند زبان برنامه نویسی اول دنیا می باشد که روند رو به رشد دارد.

برای اینکه بیشتر به بازار زبان برنامه نویسی پایتون پی ببرید می توانید جمله ایی را که یکی از مدیران گوگل در مورد این زبان گفته را ببینید و همین طور شرکت های زیادی که از این زبان برنامه نویسی استفاده می کنند.

- Google
 - Python has been an important part of Google since the beginning, and remains so as the system grows and evolves. Today dozens of Google engineers use Python, and we're looking for more people with skills in this language." said Peter Norvig, director of search quality at Google, Inc.
- Industrial Light & Magic, Journyx, IronPort, EVE Online, HomeGain, ...

قابلیت های زبان برنامه نویسی پایتون

- Easy to Learn and Use
- Expressive Language
- Interpreted Language
- Cross-Platform Language
- Free and Open Source
- Object-Oriented Language
- Extensible
- Large Standard Library
- GUI Programming Support
- Integrated

دانلود و نصب پایتون

برای نصب پایتون در ویندوز ابتدا نیاز به دانلود آن از وب سایت رسمی پایتون دارید:

<https://www.python.org/downloads>

Python Releases for Windows

- [Latest Python 3 Release - Python 3.9.0](#)
- [Latest Python 2 Release - Python 2.7.18](#)

Stable Releases

- [Python 3.9.0 - Oct. 5, 2020](#)
Note that Python 3.9.0 cannot be used on Windows 7 or earlier.
 - [Download Windows help file](#)
 - [Download Windows x86-64 embeddable zip file](#)
 - [Download Windows x86-64 executable installer](#)
 - [Download Windows x86-64 web-based installer](#)
 - [Download Windows x86 embeddable zip file](#)
 - [Download Windows x86 executable installer](#)
 - [Download Windows x86 web-based installer](#)
- [Python 3.8.6 - Sept. 24, 2020](#)

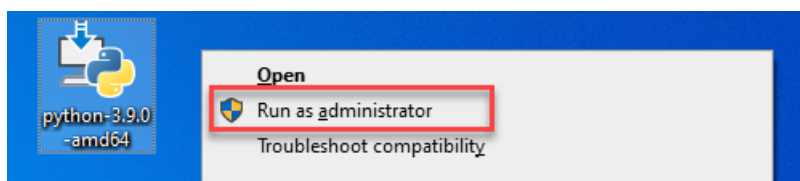


Pre-releases

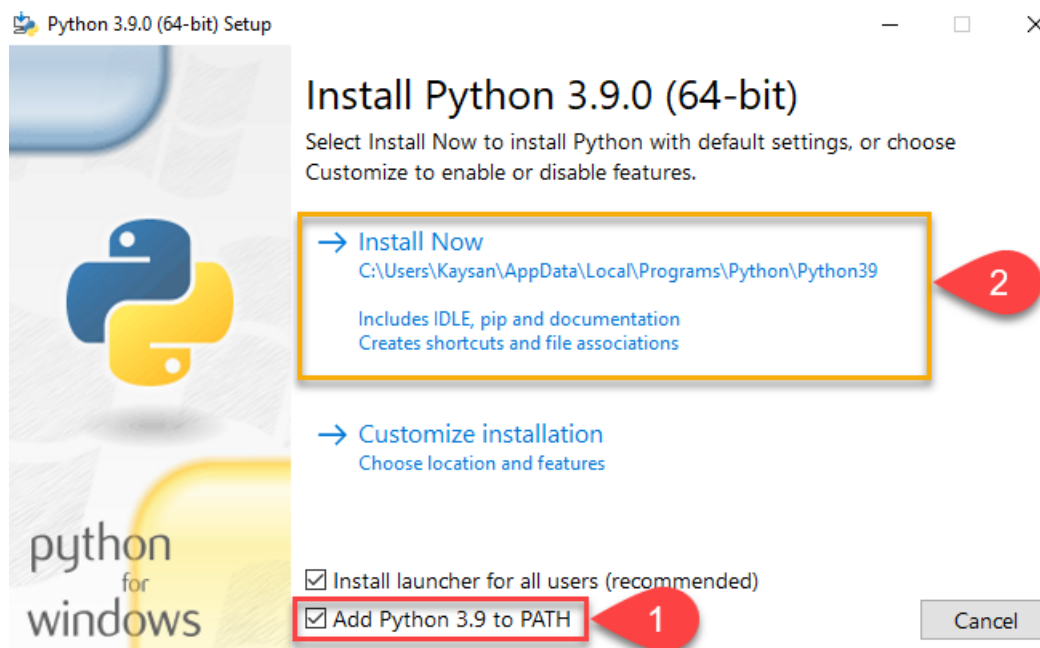
- [Python 3.10.0a1 - Oct. 5, 2020](#)
 - [Download Windows help file](#)
 - [Download Windows x86-64 embeddable zip file](#)
 - [Download Windows x86-64 executable installer](#)
 - [Download Windows x86-64 web-based installer](#)
 - [Download Windows x86 embeddable zip file](#)
 - [Download Windows x86 executable installer](#)
 - [Download Windows x86 web-based installer](#)
- [Python 3.9.0rc2 - Sept. 17, 2020](#)
 - [Download Windows help file](#)

نیاز دارید تا آخرین نسخه از پایتون ۳ را دانلود کنید! و به این نکته توجه کنید که نسخه ۳.۹ پایتون بر روی ویندوز ۷. نسخه های قبل از آن قابل نصب نیست، اما شما میتوانید از نسخه های قبلی ورژن ۳ پایتون استفاده کنید.

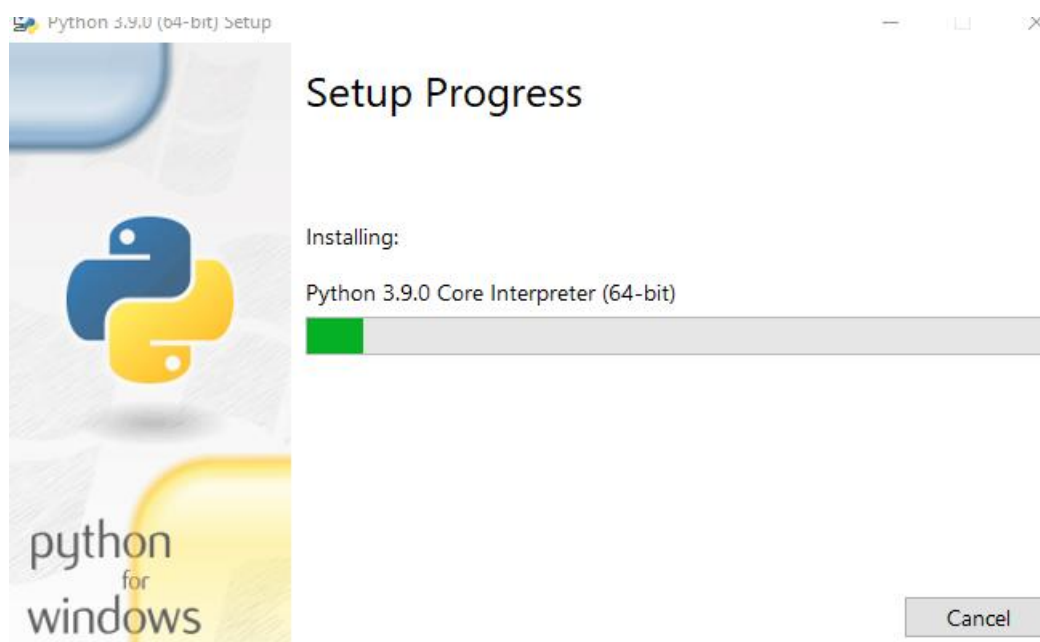
بعد از دانلود فایل exe. باید آن را با دسترسی Administrator اجرا کنید:



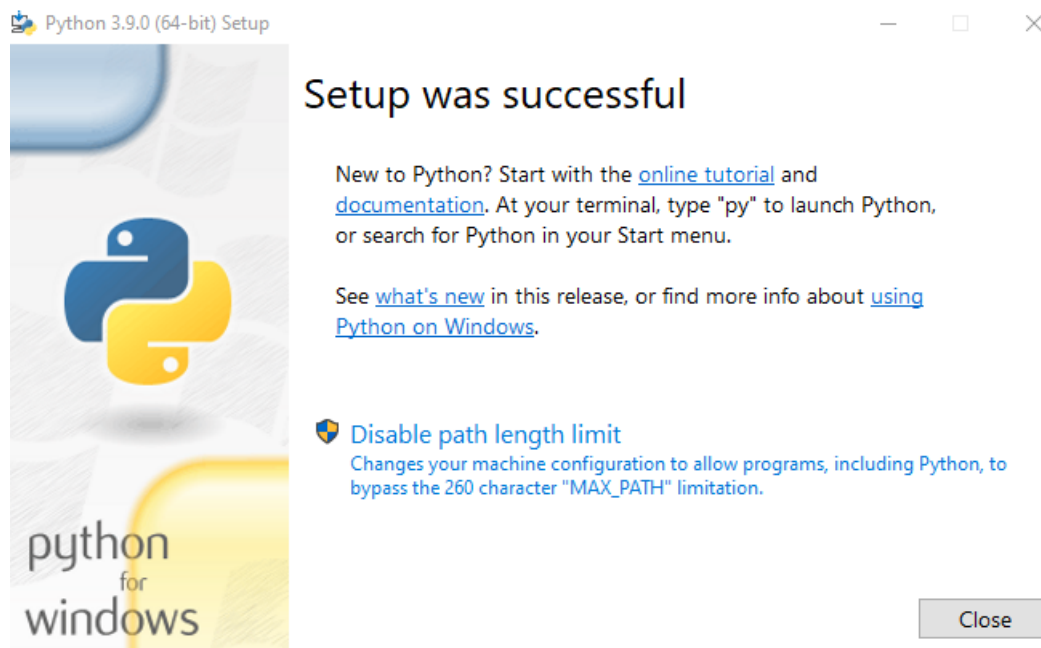
بعد از اجرای فایل وارد مراحل نصب پایتون می شوید برای ادامه حتما در پایین پنجره تیک گزینه Add Python To patch را بزنید! و در نهایت Install Now را انتخاب کنید:



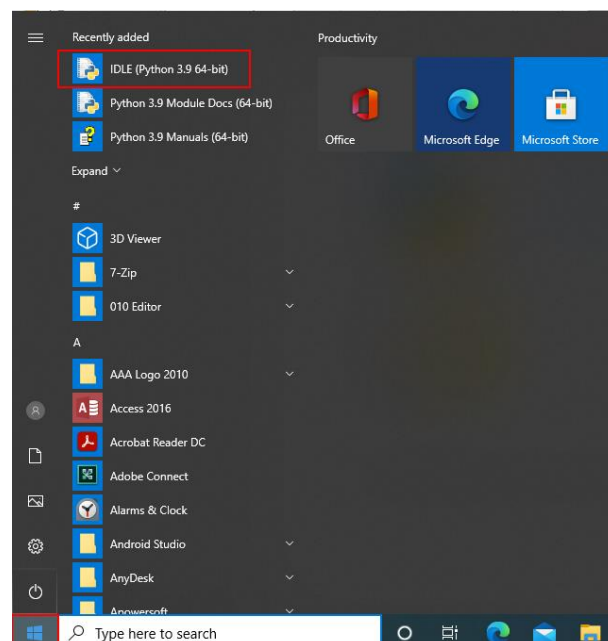
بعد از این مرحله نصب پایتون شروع می شود:



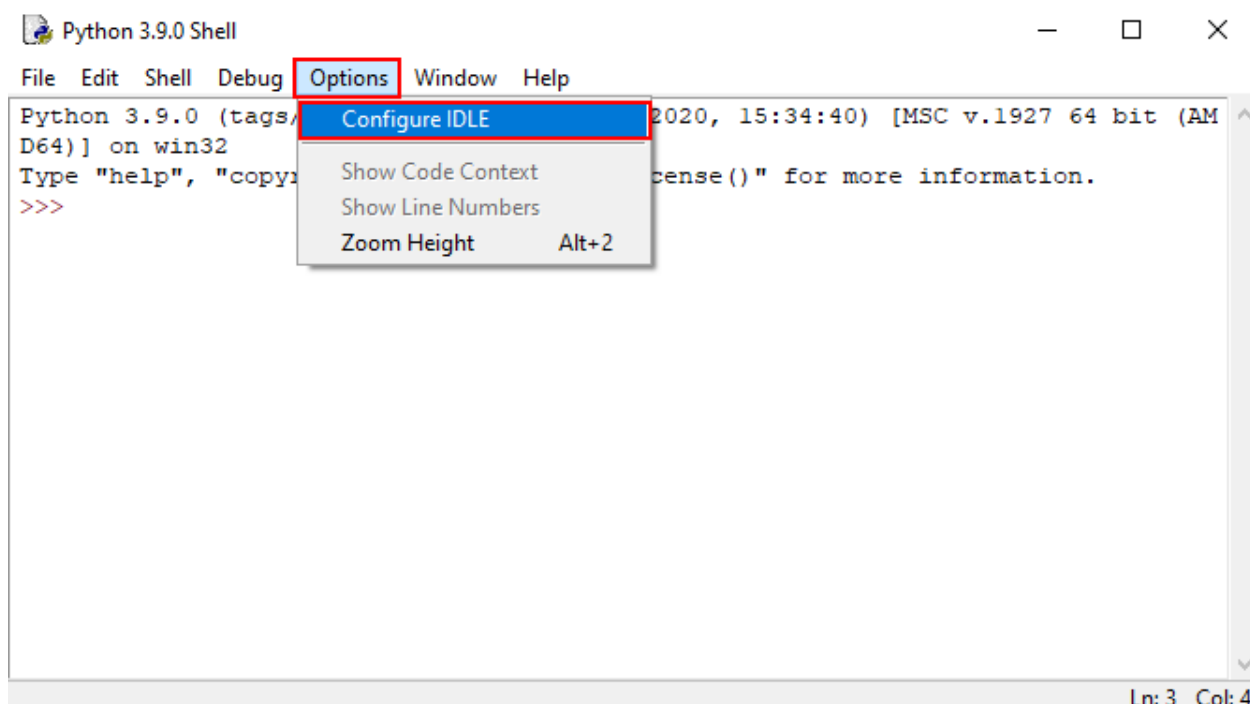
بعد از اتمام نصب در پنجره جدید پیغام نصب موفقیت آمیز پایتون نمایش داده می شود:



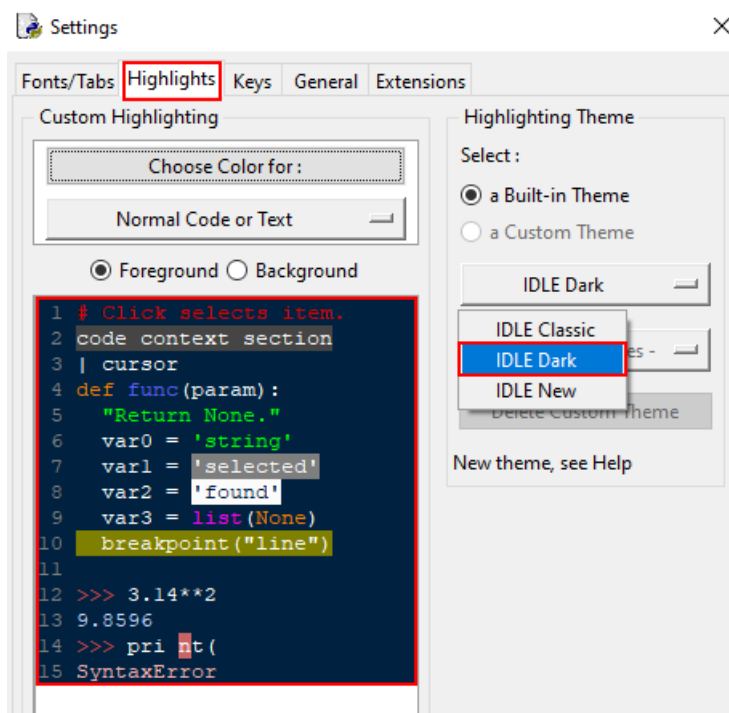
بعد از نصب پایتون شما می توانید Interface IDLE را برای شروع به کار پایتون انتخاب کنید. IDLE مخفف Integrated Development and Learning Environment می باشد و یک Python Shell یا Interactive Interpreter می باشد که شما در این محیط می توانید دستورات زبان پایتون را تست و اجرا کنید:



بعد از انتخاب IDLE همانطوری که می بینید صفحه Python Shell باز می شود که به صورت پیش فرض به رنگ سفید می باشد که می توان رنگ و فونت آن را تغییر داد، برای تغییر رنگ صفحه Python Shell بایستی از برروی منوی Option کلیک کنید و سپس گزینه Configure IDLE را انتخاب کنید:



در صفحه باز شده برروی گزینه Highlight کلیک کرده و سپس گزینه IDLE Dark را انتخاب کنید تا رنگ صفحه IDLE تغییر کند سپس برروی دکمه Apply کلیک کنید:



برنامه یا program چیست؟

یک برنامه در واقع یک توالی از دستورات می باشد که نحوه انجام شدن یک محاسبه را انجام می دهد.

هر زبان برنامه نویسی شامل یکسری دستورالعمل پایه می باشد که شامل موارد زیر می باشند:

- ورودی یا Input
- خروجی یا Output
- محاسبات یا Math
- اجرای شرطی یا Conditional Execution
- تکرار یا Repetition

برای برنامه نویسی پایتون شما می توانید از IDE های مختلفی استفاده کنید یک IDE محیطی است که شما می توانید کد خود را در داخل آن بنویسید IDE مخفف Integrated Development Environment می باشد که به صورت پیش فرض بعد از نصب پایتون شما یک IDLE خواهید داشت که یک IDE ساده برای پایتون می باشد.

شما می توانید از انواع IDE های زیر نیز استفاده کنید:

- Spider
- ATOM
- PyCharm
- VSCODE

برای شروع کار و برای آشنایی با دستورات بایستی از محیط IDLE استفاده کنید.

یکی از مزایای IDLE این است که شما می توانید دستورات خود را در این محیط وارد کنید و نتیجه آن دستور را نیز در همان محیط مشاهده کنید.

مثال ۱: محاسبه جمع دو عدد.

```
Python 3.9.0 Shell
File Edit Shell Debug Options Window Help
Python 3.9.0 (tags/v3.9.0:9cf6752, Oct 5 2020, 15:34:40) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> 7+5
12
>>> |
```

مثال ۲: محاسبه تفریق دو عدد.

```
>>> 5-2
3
```

مثال ۳: محاسبه ضرب دو عدد.

```
>>> 5*2
10
```

مثال ۴: محاسبه تقسیم دو عدد.

```
>>> 4/2
2.0
```

مثال ۵: توان رساندن یک عدد به دیگری.

```
>>> 2**4  
16
```

مثال ۶: چاپ یک رشته.

```
>>> print('Hello,World')  
Hello,World  
>>> |
```

نکته:

یک رشته در زبان برنامه نویسی پایتون بایستی میان دو ' ' یا " " قرار بگیرد به عبارت دیگر هر عدد یا کارکتری که مابین علامت های ' ' یا " " قرار بگیرد از جنس String خواهد بود.

فرآیند Debugging چیست

به فرآیندی گفته می شود که برنامه نویس بدنبال پیدا کردن خطاهای برنامه می باشد به خطاهای برنامه در اصطلاح Bug نیز گفته می شود.

برای مثال به Bug و روش رفع Bug دقت کنید:

```
>>>  
>>>  
>>> print 'Hello,World'  
SyntaxError: Missing parentheses in call to 'print'. Did you mean print('Hello,World')?  
>>>  
>>> print('Hello,World')  
Hello,World
```

Bug Detection

Fixed Bug

به صورت کلی پیدا کردن Bug و رفع آن کار بسیار سختی می باشد ولی پیدا کردن Bug باعث بالا رفتن مهارت شما در برنامه نویسی خواهد شد.

متغیرها (Variables)

یک Variable یک Location رزو شده در Memory می باشد که به این Location در اصطلاح Variable گفته می شود که در داخل یک Variable می توانید Value ذخیره کنید. یک Value می تواند String یا Number و یا هر چیز دیگری باشد، و به این نکته توجه کنید که بهتر است اسامی متغیرها را با مفهوم انتخاب کنید.

هر متغیر شامل دو بخش است:

۱- نام متغیر (Name)

۲- مقدار (Value)

به عنوان مثال:

```
1. >>> number = 10
```

در مثال بالا number نام متغیر و ۱۰ مقدار آن می باشد، مقدارهایی که در متغیرها ذخیره می شوند میتواند از انواع داده هایی مثل رشته ای، اعداد، منطقی و باشد، که در ادامه با هر کدام آشنا خواهید شد.

خب حالا که با متغیرها آشنا شدید میتوانید آن ها را در پایتون تعریف کرده و در خروجی مشاهده کنید:

```
1. >>> number = 10
2. >>> name = 'mobin'
3. >>> print(number)
4. 10      #خروجی
5. >>> print(name)
6. mobin   #خروجی
```

همانطور که در بالا مشاهده می کنید، دو متغیر را تعریف کردیم یکی با مقدار عددی و دیگری با مقدار رشته ای، برای تعریف مقدار رشته ای باید آن را بین دو " " یا ' ' قرار دهید، در ادامه با استفاده از دستور `print` دو متغیر را در خروجی نمایش می دهیم، توجه کنید به خاطر اینکه قصد داریم یک متغیر را در خروجی نمایش دهیم نباید از " " در دستور `print` استفاده کنید، و همانطور که مشاهده می کنید مقدار دو متغیر را در خروجی نمایش خواهد داد.

می توان یک مقدار را به صورت همزمان به چند متغیر اختصاص داد:

```
1. >>> a = b = c = 1
```

در خط کد بالا مقدار ۱ به صورت همزمان به چند متغیر `a` , `b` , `c` اختصاص داده میشود، و هر کدام را که در خروجی نمایش دهید مقدار ۱ را در خروجی باز می گرداند.

همچنین در پایتون می توان در یک خط چند متغیر را تعریف کنید و به هر کدام مقداری را اختصاص داد:

```
1. >>> a, b, c = 1 , 2 , 'mobin'
```

در خط کد بالا سه متغیر `a`, `b`, `c` تعریف شده و به ترتیب به هر کدام مقداری داده شد، مقدار ۱ به متغیر `a` ، مقدار ۲ به متغیر `b` ، و مقدار رشته ای `mobin` به متغیر `c` داده شد.

قواعد نام گذاری متغیر ها

برای نام گذاری متغیر ها در پایتون قواعد و قرارداد هایی وجود دارد:

برای نام گذاری متغیر ها از نگارش شتری یا (`camelCase`) استفاده می شود، در این روش نام با حرف کوچک شروع می شود و کلمه دوم با حرف بزرگ شروع می شود و بقیه حروف کوچک هستند:

```
1. myName  
2. myAge  
3. myAddress
```

این ها نمونه هایی از نام گذاری camelCase هستند.

نکاتی که بایستی در زمان تعریف متغیر بایستی رعایت کنید:

- همیشه نام متغیر را با یک حرف شروع کنید.
- یک متغیر نمی تواند با یک عدد شروع شود.
- از اسامی رزو شده در پایتون استفاده نکنید.
- سعی کنید از اسامی قابل فهم در تعریف متغیرها استفاده کنید.
- شما می توانید از کارکتر _ در ابتدا و یا در هر جایی از اسم متغیر استفاده کنید.

استفاده از Script Mode

ابزار IDLE دو مد زیر را دارد:

• Interactive Mode

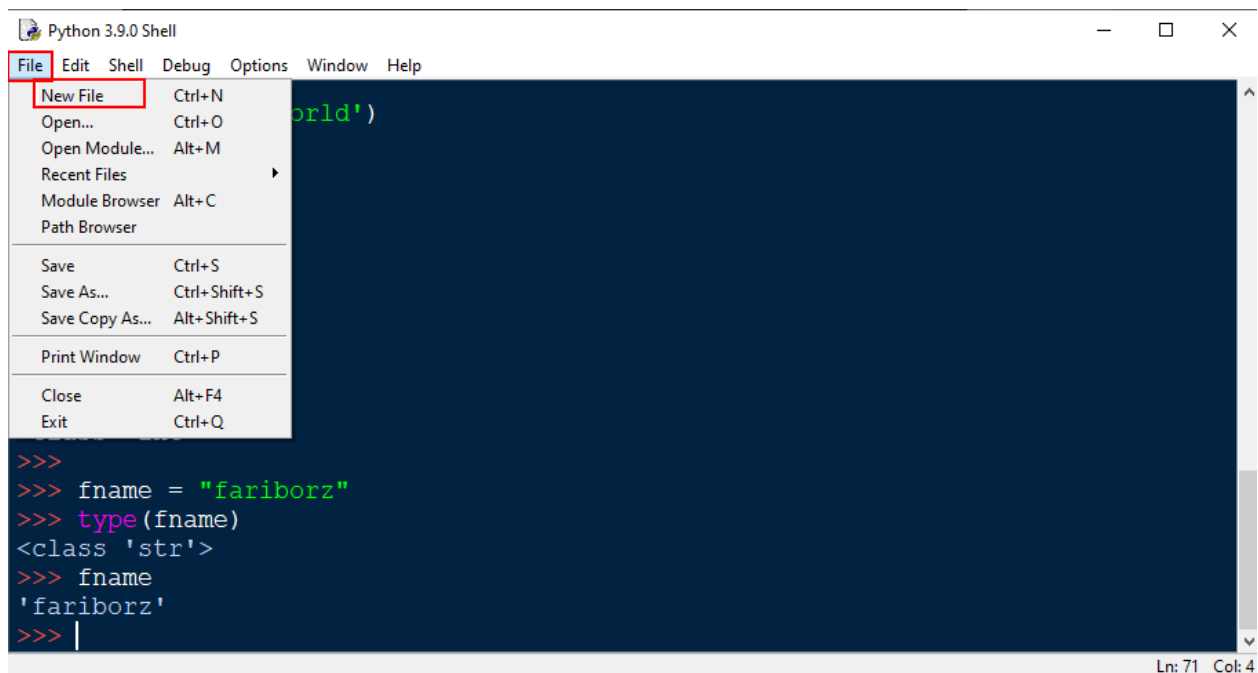
• Script Mode

در محیط Interactive Mode دستورات خط به خط تست و اجرا می شوند.

اما میتوان کد های پایتون را در یک فایل با پسوند `.py` ذخیره و اجرا کرد که به آن Script Mode گفته میشود، میتوان از IDLE برای این کار استفاده کرد:

مرحله 1:

در محیط IDLE از منوی File گزینه New File را انتخاب کنید:

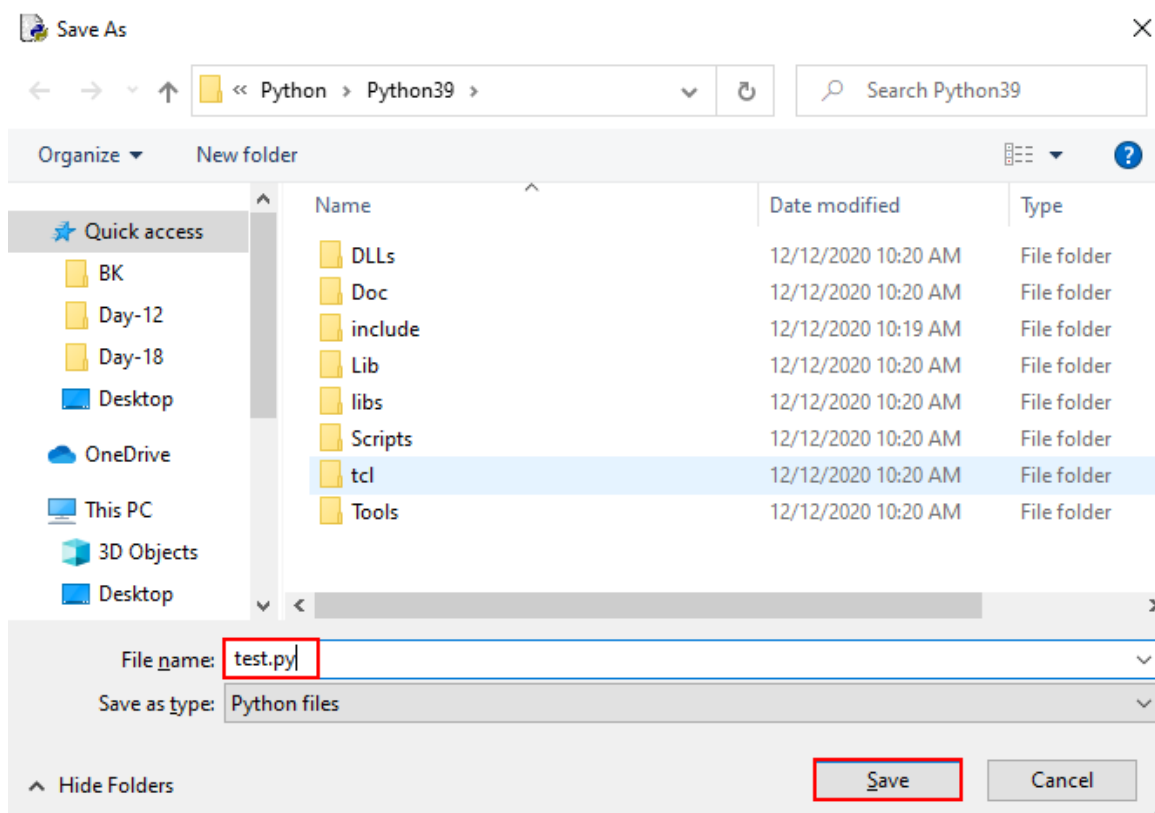


مرحله ۲:

در این مرحله بایستی از صفحه باز شده از منوی File گزینه Save را انتخاب کنید:

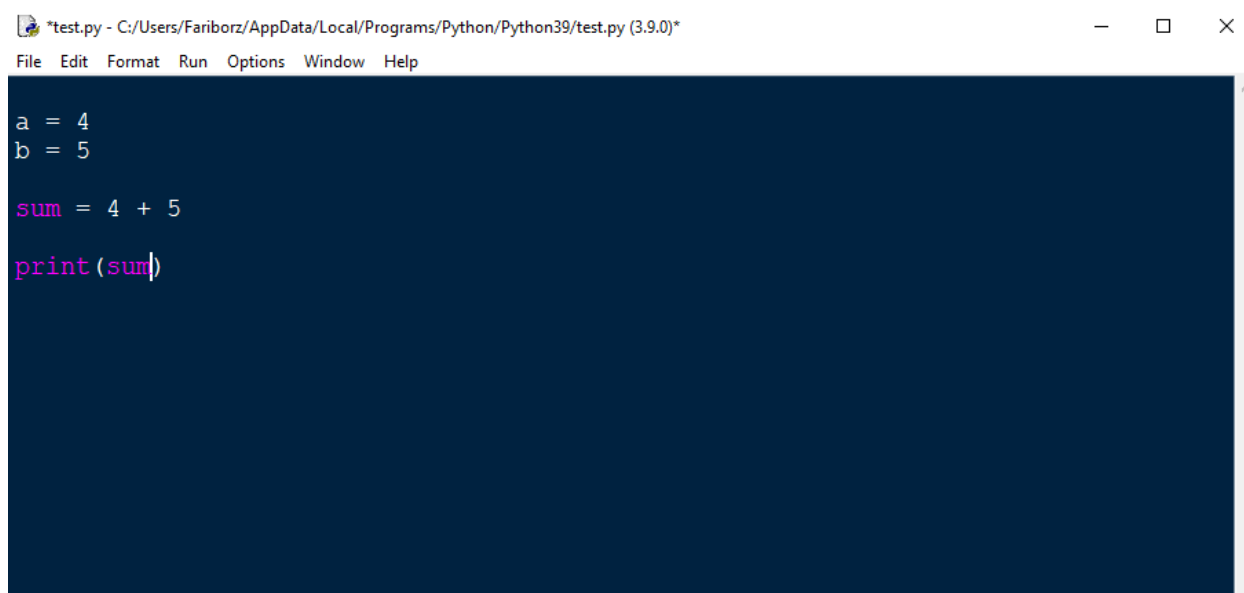


مرحله ۳:



مرحله ۴:

در فایل اسکریپت خود کد زیر را وارد کنید و سپس با کلید CTRL+S تغییرات را در داخل فایل ذخیره کنید:



```
*test.py - C:/Users/Fariborz/AppData/Local/Programs/Python/Python39/test.py (3.9.0)*
File Edit Format Run Options Window Help

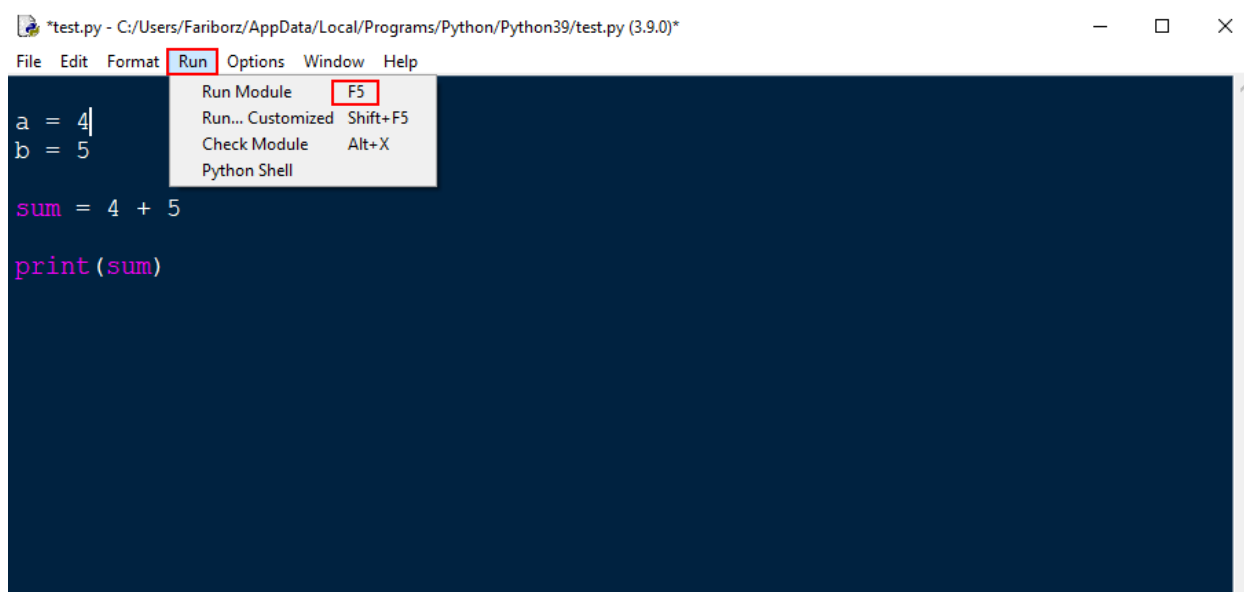
a = 4
b = 5

sum = 4 + 5

print(sum)
```

مرحله ۵:

برای اجرای فایل اسکریپت می توانید از منوی Run گزینه Run Module را انتخاب کنید و یا با زدن کلید F5 اسکریپت را اجرا کنید.



```
*test.py - C:/Users/Fariborz/AppData/Local/Programs/Python/Python39/test.py (3.9.0)*
File Edit Format Run Options Window Help

Run Module F5
Run... Customized Shift+F5
Check Module Alt+X
Python Shell

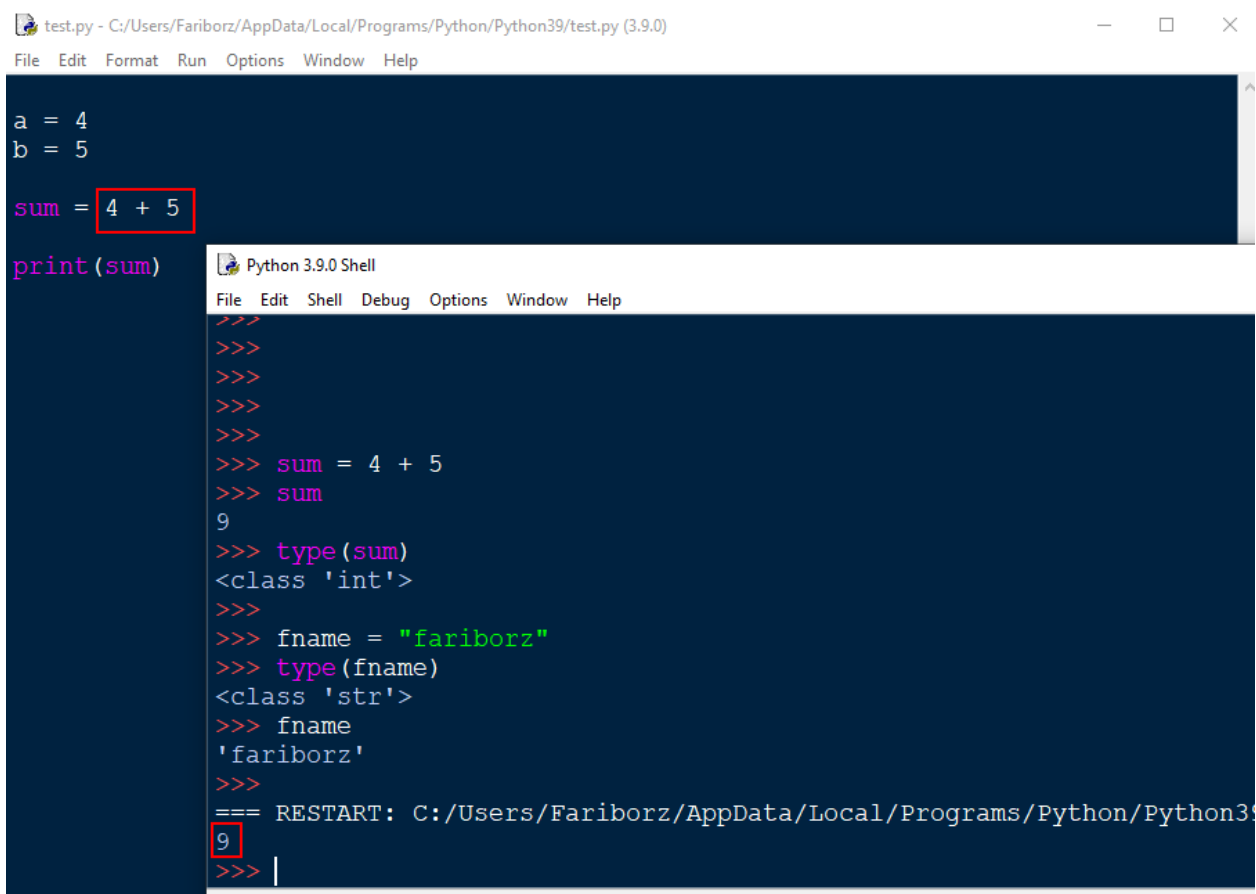
a = 4
b = 5

sum = 4 + 5

print(sum)
```


مرحله ۶:

در نهایت شما نتیجه اسکریپت را خواهید دید:



The image shows a Python IDE window titled 'test.py - C:/Users/Fariborz/AppData/Local/Programs/Python/Python39/test.py (3.9.0)'. The code in the editor is:

```
a = 4
b = 5
sum = 4 + 5
print(sum)
```

The 'sum = 4 + 5' line is highlighted with a red box. Below the editor is a 'Python 3.9.0 Shell' window showing the execution of the script:

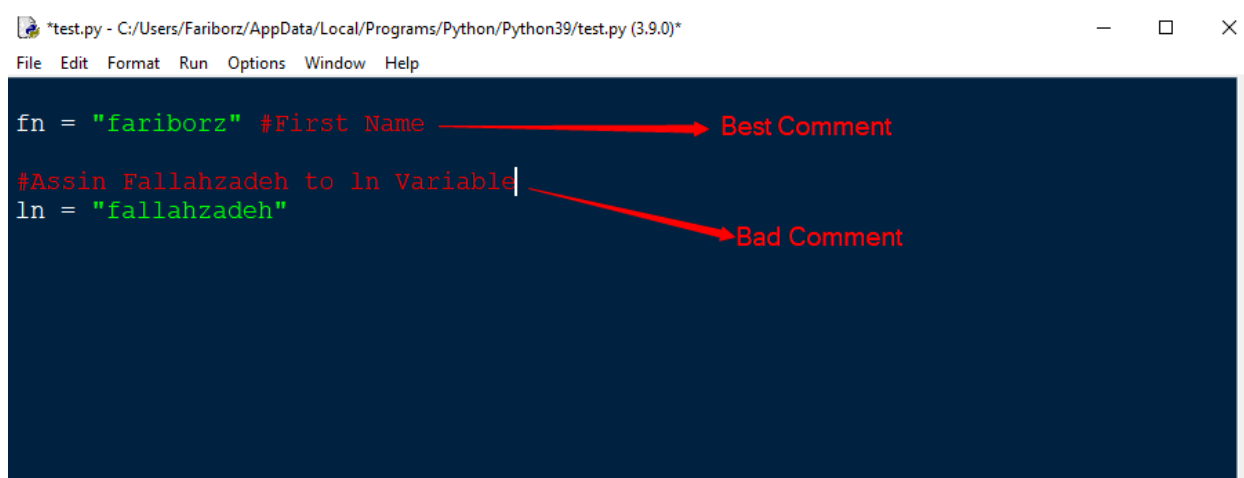
```
>>>
>>>
>>>
>>>
>>> sum = 4 + 5
>>> sum
9
>>> type(sum)
<class 'int'>
>>>
>>> fname = "fariborz"
>>> type(fname)
<class 'str'>
>>> fname
'fariborz'
>>>
=== RESTART: C:/Users/Fariborz/AppData/Local/Programs/Python/Python39
9
>>> |
```

The output '9' is highlighted with a red box.

کامنت گذاری در Script

برای مشخص کردن دلیل دستورات هر قسمت از کد می توان از توضیحاتی استفاده کرد که کد را قابل فهم برای هر برنامه نویسی بکند این توضیحات در حین اجرای کد به عنوان دستور اجرا نمی شوند و هیچ تاثیری بر روی خروجی برنامه ندارند.

در تصویر زیر می توانید نحوه درست Comment گذاری را مشاهده کنید:



```
*test.py - C:/Users/Fariborz/AppData/Local/Programs/Python/Python39/test.py (3.9.0)*
File Edit Format Run Options Window Help

fn = "fariborz" #First Name
#Assin Fallahzadeh to ln Variable
ln = "fallahzadeh"
```

انواع Error ها در پایتون

به صورت کلی ما ۳ نوع Error در زبان برنامه نویسی پایتون داریم که شامل موارد زیر می باشد:

- Syntax Error

این Error مربوط به اشتباهات Syntax در دستورات پایتون می باشد .

مثال:

```
>>> print "Hello"  
SyntaxError: Missing parentheses in call to 'print'. Did you mean print("Hello")  
?
```

- Runtime Error

این Error در زمان اجرای برنامه نمایان می شود.

- Semantic Error

این نوع از خطاها به خطاهای معنایی معروف هستند .

به صورت کلی دو نوع Data Type داریم آن دسته از Value هایی که بعد از ایجاد می توانیم آنها را تغییر دهیم و آن دسته از Value هایی که بعد از ایجاد نمی توانیم تغییر دهیم برای Value هایی که بعد از ایجاد می توانیم تغییر دهیم می تواند List و یا Set و یا Dictionary باشد در حالی که Value هایی مانند String و Number و یا Tuples و یا Booleans بعد از ایجاد قابل تغییر نیستند .

بنابراین می توان گفت که در زبان برنامه نویسی Python از Data Type های زیر استفاده می شود:

- String
- Number
- List
- Tuples
- Dictionary
- Set
- None
- Booleans

شما می توانید با استفاده از دستور Type نوع دیتا را شناسایی کنید.

اعداد در پایتون

هر مقدار (Value) در پایتون دارای یک نوع است. با توجه به اینکه در زبان برنامه‌نویسی پایتون همه چیز شی محسوب می‌شود، انواع داده در واقع کلاس هستند و متغیرها نمونه‌های (شی‌های) این کلاس محسوب می‌شوند. انواع داده‌های مختلفی در پایتون وجود دارند که یکی از آن‌ها اعداد هستند که در ادامه به آن‌ها خواهیم پرداخت.

انواع اعداد در پایتون

۱- اعداد صحیح (Integer):

```
1. >>> a = 5
2. >>> a = 123456789
```

اعداد صحیح می‌توانند طول‌های مختلفی داشته باشند، که این مورد به حافظه موجود محدود می‌شود.

۲- اعداد اعشاری (Float):

```
1. >>> a = 3.14
2. >>> a = 0.123456478912345678
```

یک عدد اعشاری در پایتون ۳ می‌تواند تا ۱۷ رقم اعشار امتداد داشته باشد و بخش صحیح و اعشاری آن با یک . از هم جدا می‌شوند.

با متد `type()` می توانید نوع داده داخل یک متغیر را در خروجی مشاهده کنید:

```
1. >>> number = 10
2. >>> name = "mobin"
3. >>> print (type(number))
4. <class 'int'>      #خروجی
5. >>> print (type(name))
6. <class 'str'>     #خروجی
```

همانطور که در خروجی می بینید نوع مقدار داخل متغیر `number` عددی و `name` از نوع رشته ای می باشد، میتوان انواع داده های عددی را هم با این متد بررسی کرد:

```
1. >>> num1 = 10
2. >>> num2 = 3.14
3. >>> num3 = 1+2j
4. >>> print (type(num1))
5. <class 'int'>      #خروجی
6. >>> print (type(num2))
7. <class 'float'>   #خروجی
```

تبدیل نوع داده های عددی به یک دیگر

میتوان با استفاده از متد های `float()` و `int()` هر کدام از داده های عددی را به یک دیگر تبدیل کرد:

```
1. >>> num1 = 2
2. >>> x = float(num1)
3. >>> print (x)
4. 2.0      #خروجی
```

در خروجی عدد ۲ که یک داده عددی از نوع `int` است را به صورت اعشار یا `float` نمایش می دهد.

عملگر ها در پایتون

عملگر ها یا operators سمبل های خاصی هستند که پردازش های حسابی و منطقی را انجام می دهند به زبان ساده تر عملگر ها در واقع علائمی هستند که کاری را در برنامه نویسی انجام می دهند، که در زبان پایتون از انواع مختلف آنها استفاده می شود:

عملگر های ریاضی (Arithmetic operators)

عملگر های ریاضی یا عملگر های حسابی، برای انجام پردازش های ریاضی همانند جمع، تفریق، ضرب و ... استفاده می شوند در جدول زیر به معرفی هر کدام از آنها می پردازیم:

عملگر	شرح عملگر	مثال
+	جمع کردن دو عمل وند	$a + b$
-	تفریق عمل وند	$a - b$
*	ضرب دو عمل وند	$a * b$
/	تقسیم کردن دو عمل وند	a / b
%	محاسبه باقی مانده تقسیم عمل وند	$a \% b$
//	محاسبه خارج قسمت	$a // b$
**	a رساندن متغیر b به توان	$a ** b$

```
>>> a = 20
>>> b = 10
>>> print(a+b)
30 # خروجی
>>> print(a-b)
10 # خروجی
>>> print(a*b)
200 # خروجی
>>> print(a/b)
2.0 # خروجی
>>> print(a%b)
0 # خروجی
>>> print(a//b)
2 # خروجی
>>> print(a**b)
10240000000000 # خروجی
```


ترتیب عملگرهای ریاضی در محاسبات

زمانی که در یک عملیات ریاضی چندین عملگر وجود داشته باشد ترتیب اجرای عملگرها به صورت PEMDAS می باشد P(Parentheses) به نشان دهنده پرانتز و E(Exponentiation) به معنی توان و M(Multiplication) به معنی ضرب و D(Division) به معنی تقسیم و A(Addition) به معنی جمع و S(Subtraction) می باشد به عبارت دیگر ترتیب اجرای این عملگرها به صورت زیر می باشند:

- پرانتز ()
- توان **
- ضرب و تقسیم / *
- جمع و تفریق + -

مثال ۱:

همانطوری که در مثال زیر مشاهده می کنید اول عملیات داخل پرانتز انجام می شود و بعد به توان خواهد رسید:

```
>>> (2+2) ** (4-2)
16
```

مثال ۲:

در مثال زیر اول عملیات توان انجام می شود و سپس عملیات جمع انجام می شود:

```
>>> 1 + 2**3
9
```

مثال ۳:

همانطوری که در تصویر زیر مشاهده می کنید ترتیب اجرا عملگرها از چپ به راست می باشد:

```
>>> 4*2/1*2
16.0
```

عملگر های مقایسه ای (Relation operators)

عملگر های مقایسه ای برای انجام مقایسه بین مقادیر استفاده می شوند و متناسب با شرط، خروجی مقایسه برابر یا True یا False خواهد بود:

مثال	شرح عملگر	عملگر
$a > b$	علامت بزرگ تر	$<$
$a < b$	علامت کوچک تر	$>$
$a == b$	برابر است با	$==$
$a != b$	نامساوی	$!=$
$a >= b$	بزرگ تر یا مساوی	$=<$
$a <= b$	کوچک تر یا مساوی	$=>$

```
>>> a = 20
>>> b = 25
>>> a > b
False #خروجی
>>> a < b
True  #خروجی
>>> a == b
False #خروجی
>>> a != b
True  #خروجی
>>> a >= b
False #خروجی
>>> a <= b
True  #خروجی
```

عملگر های تخصیص یا انتساب (Assignment operators)

عملگر های تخصیص در پایتون برای تخصیص یک مقدار به یک متغیر مورد استفاده قرار می گیرند، مثلاً $a = 5$ یک عملگر تخصیص ساده است که مقدار 5 را در متغیر a تخصیص می دهد عملگر های ترکیبی هم در پایتون وجود دارند مثل $a += 5$ این عملگر اول 5 را به مقدار داخل متغیر a اضافه می کند و در آخر حاصل را در متغیر a تخصیص می دهد. برای آشنایی بیشتر به جدول زیر دقت کنید:

عملگر	مثال	برابر است با
=	$x = 5$	$x = 5$
=+	$x += 5$	$x = x + 5$
=-	$x -= 5$	$x = x - 5$
=*	$x *= 5$	$x = x * 5$
=/	$x /= 5$	$x = x / 5$
=%	$x \% = 5$	$x = x \% 5$
=//	$x //= 5$	$x = x // 5$
=**	$x ** = 5$	$x = x ** 5$

عملگر های منطقی (Logical operators)

عملگر های منطقی در واقع and، or و not هستند:

مثال	شرح عملگر	عملگر
x and y	در صورتی که هر دو عمل وند درست (True) باشند، درست (True) است.	and
x or y	در صورتی درست (True) است که یکی از عمل وند ها درست (True) باشد.	or
not x	در صورتی درست (True) است که عمل وند غلط (False) باشد.	not

مثال:

عملگر and :

در عملگر and باید مقدار همه عمل وند ها True باشد تا نتیجه آن True شود، اما اگر فقط یکی از آنها False باشد نتیجه False می شود:

A	B	A && B
True	True	True
True	False	False
False	True	False
False	False	False

عملگر or :

این عملگر فقط زمانی False می شود که همه عمل وند ها False باشند در غیر این صورت True می شود. برای درک بهتر به جدول زیر دقت کنید:

A	B	A B
True	True	True
True	False	True
False	True	True
False	False	False

عملگر not :

اگر مقدار عمل وند برابر True باشد نتیجه False و اگر مقدار عمل وند باشد نتیجه True را بر می گرداند. عملگر not برخلاف سایر عملگر ها، فقط بر روی یک عمل وند کار می کند:

A	!A
True	False
False	True

عملگر های عضویت (Membership operators)

`in` و `not in` عملگر های عضویت در پایتون هستند. این عملگر ها برای بررسی اینکه یک مقدار یا متغیر در یک توالی پیدا می شود یا نه مورد استفاده قرار می گیرند (رشته، لیست، مجموعه و دیکشنری).

و به این نکته توجه داشته باشید که در یک دیکشنری فقط می توان وجود یا عدم وجود یک کلید را بررسی کرد، نه یک مقدار را برای درک بهتر عملگر های عضویت به جدول و مثال های زیر دقت کنید:

عملگر	شرح عملگر	مثال
<code>in</code>	در صورتی صحیح است که مقدار/متغیر در توالی پیدا شود.	<code>5 in x</code>
<code>not in</code>	در صورتی صحیح است که مقدار/متغیر در توالی پیدا نشود.	<code>5 not in x</code>

مثال:

```
1.>>> list = [1,2,3,4,5,6]
2.>>> print(1 in list)
3.True    #خروجی
4.>>> print(10 in list)
5.False   #خروجی
6.>>> print(100 not in list)

True     #خروجی
```

چون مقدار 1 در لیست وجود دارد عملگر `in` در نتیجه `True` را نمایش می دهد و چون مقدار 100 در لیست وجود ندارد `False` را در خروجی نمایش می دهد.

در عملگر not in چون مقدار 100 در لیست وجود ندارد در نتیجه True را در خروجی نمایش می دهد.

عملگر های هویت (Identity operators)

از این نوع عملگر ها برای مقایسه اشیا استفاده می شود. در واقع اشیا باید یکی باشند و در یک مکان از حافظه باشند (نه لزوما برابر) تا عملگر True برگرداند:

مثال	شرح عملگر	عملگر
X is y	اگر هر دو متغیر یک شی باشند، true بر می گرداند	is
X is not y	اگر هر دو متغیر یک شی نباشند، true بر می گرداند	is not

رشته ها (Strings) در پایتون

در زبان برنامه نویسی پایتون، مانند بسیاری دیگر از زبان ها، یک رشته ی متنی با علامت نقل قول انگلیسی (کوته شدن) دوتایی یا تکی مشخص می شود:

```
1. "This is a String"
2. 'This is another string'
3. "این یک رشته در پایتون است"
```

رشته ها در پایتون مانند لیست ها دارای اندیس هستند که شمارش آنها از ۰ شروع می شود و برای شمارش از آخر با ۱- شروع می شود، برای درک بهتر به مثال زیر توجه کنید:

T	e	c	h	O	n	e	2	4
۰	۱	۲	۳	۴	۵	۶	۷	۸
-۹	-۸	-۷	-۶	-۵	-۴	-۳	-۲	-۱

برای چاپ یک رشته در پایتون از دستور `print()` استفاده می شود. این دستور ورودی رشته ای را گرفته و در خروجی چاپ می کند:

```
1.>>> print("Hello World!!")
2.Hello World!!      #خروجی
```

همچنین می توان یک مقدار رشته ای را در یک متغیر ذخیره کرده و با دستور `print()` آن را در خروجی چاپ کرد:

```
1.>>> name = "Mobin"
2.>>> print(name)
3.Mobin      #خروجی
```

slicing در پایتون

در پایتون می توان با استفاده از متد Slice برای قطعه قطعه کردن استرینگ ها استفاده کرد و تغییراتی را در آنها ایجاد کرد برای استفاده از این متد در پایتون میتوان از علامت [] استفاده کرد، برای درک مطلب به مثال های زیر دقت کنید:

```
1. >>> site = "Techone24.com"
2. >>> print(site[0])
3. خروجی: T
```

در مثال بالا متغیری را تعریف کردیم و به آن مقدار رشته ای تخصیص دادیم، با دستور print() و نوشتن نام متغیر داخل آن اول از پایتون خواستیم تا مقدار داخل متغیر را به صورت کامل نمایش دهد و در قدم بعدی با استفاده از [0] پایتون مقدار اولین index رشته قرار داده شده در متغیر را در خروجی چاپ کرد.

به دلیل اینکه شمارش اندیس ها در پایتون از ۰ شروع می شوند برای چاپ کردن اولین عنصر مقدار ۰ را در [] قرار دادیم و اگر بخواهیم رنج خاصی از اندیس های داخل یک رشته را در خروجی ببینیم باید رنج مورد نظر را داخل [] قرار دهیم و مابین شماره اولین اندیس و آخرین اندیس علامت : را قرار دهید برای درک مطلب به مثال زیر توجه کنید:

```
1. # print([Start:End])
2. >>> site = "Techone24.com"
3. >>> print(site[0:3])
4. خروجی: Tec
```

همچنین میتوان با استفاده از ۱- آخرین اندیس داخل یک رشته یا لیست را در خروجی چاپ کرد (شمارش اندیس ها از آخر به اول در پایتون با ۱- شروع می شوند) برای درک بهتر به مثال زیر توجه کنید :

```
1. >>> site = "Techone24.com"
```

```
2. >>> print(site[-1])
3. m # خروجی
```

به هم چسباندن رشته ها (String Concatenation)

در پایتون میتوانیم استرینگ ها را به هم بچسبانیم و یا در نحوه نمایش آنها تغییراتی را ایجاد کنیم:

```
1. >>> firstName = "Mobin"
2. >>> lastName = "Ardakani"
3. >>> print(firstName + lastName)
4. MobinArdakani # خروجی
```

تکرار رشته ها (Repetition)

در پایتون میتوان چاپ یک رشته را چندین بار تکرار کرد:

```
1. >>> site = "TechOne24.com"
2. >>> print(site * 2)
3. TechOne24.comTechOne24.com # خروجی
4. >>> print(site * 5)
5. TechOne24.comTechOne24.comTechOne24.comTechOne24.comTechOn
   e24.com #
```

جستجوی کاراکتر خاص در String

می‌توانیم از `in` و `not in` برای جستجوی یک کاراکتر در `string` استفاده کنیم، `in` در صورتی که کاراکتر دلخواه در یک رشته وجود داشته باشد در نتیجه مقدار `True` و اگر نباشد مقدار `False` را نمایش می‌دهد اما `not in` در صورت وجود نداشتن کاراکتر در رشته در خروجی مقدار `True` و در صورت وجود داشتن مقدار `False` را نمایش می‌دهد برای درک بهتر به مثال زیر دقت کنید:

```
1.>>> site = "TechOne24.com"
2.>>> print('T' in site)
3.True    #خروجی
4.>>> print('S' in site)
5.False   #خروجی
```

در مثال پایین با استفاده از `not in` دو کاراکتر متفاوت را که یکی در متغیر وجود داشت و دیگری وجود نداشت جستجو کردیم همانطور که در خروجی‌ها می‌بینید در جستجوی کاراکتری که وجود داشت مقدار `True` و کاراکتری که وجود نداشت `False` را در خروجی چاپ کرد:

```
1.>>> site = "TechOne24.com"
2.>>> print('O' not in site)
3.False   #خروجی
4.>>> print('W' not in site)
5.True    #خروجی
```

کاراکترهای کنترلی (Escape Characters) در پایتون

کاراکترهای کنترلی در واقع کاراکترهای ترکیبی هستند که با یک \ شروع می شوند و به دنبال آنها یک حرف یا عدد می آید و یک رشته را با فرمت های خاص نمایش می دهند، جدول زیر لیست کاراکترهای کنترلی در پایتون را نمایش می دهد:

کاراکتر کنترلی	شرح عملکرد
\'	چاپ سینگل کوتیشن
\"	چاپ دابل کوتیشن
\\	چاپ بک اسلش
\0	چاپ فضای خالی
\n	خط جدید(رفتن به خط بعدی)
\t	Tab

مثال :

```
1. >>> print("this is a '\")
2. this is a '
3. >>> print("this is a '\"")
4. this is a "
5. >>> print("this is a \\")
6. this is a \
7. >>> print("this is \0 python")
8. this is  python
9. >>> print("Hello \n World")
10. Hello
11.  World
12. >>> print("Hello \t World")
```

```
13.      Hello      World
```

رشته های خام (Row string)

با کاراکترهای کنترلی آشنا شدید و مثال هایی هم در مورد آنها ذکر کردیم، اما مواقعی هستند که بخواهید از \ داخل یک رشته استفاده کنید و آن را در خروجی چاپ کنید برای اینکه پایتون آن را کاراکتر کنترلی به حساب نیاورد و آن را به همان شکل در خروجی چاپ کند باید از رشته خام یا row string استفاده کنید:

```
1.>>> dir = r'C:\python\scripts'
2.>>> print(dir)
3.C:\python\scripts #خروجی
```

در مثال بالا می خواهیم یک آدرس را داخل متغیر قرار دهیم در آدرس دهی باید از \ استفاده کنید اما پایتون آنها را به عنوان کاراکتر کنترلی می شناسد پس باید قبل از ' ' یک کاراکتر r قرار دهید تا پایتون آن را به عنوان r string بشناسد.

استفاده از متغیرها در String:

در پایتون میتوانیم از متغیرها داخل string ها استفاده کنید و آنها را در خروجی چاپ کنید، برای اینکار باید از % استفاده کنیم و با توجه به نوع داده داخل متغیر کاراکترهایی جلوی آن قرار بدهیم تا پایتون متغیر را در خروجی چاپ کند برای درک بهتر به مثال های زیر توجه کنید:

```
1. # مقدار داده رشته ای
2.>>> string = "techone24.com"
3.>>> print("Site is %s" %string) # %s
4.Site is techone24.com #خروجی
5. # مقدار داده دسیمال
6.>>> num = 20
7.>>> print("Decimal Number is %d" %num) # %d
8.Decimal Number is 20 #خروجی
9. # مقدار داده عدد صحیح
10.>>> intnum = 2020
11.>>> print("Year is %i" %intnum) #i
12.integer number is 2020 #خروجی
13. # مقدار داده عدد اعشاری
14.>>> floatnum = 3.8
15.>>> print("Python version is %f" %floatnum) # %f
```

F String در پایتون

نوعی ساده دیگری از string ها که در پایتون ۳.۶ به بعد قابل استفاده هستند و در ورژن های قبلی پایتون وجود ندارند و کار با آن بسیار ساده است در مثال های زیر با کاربرد های آن آشنا می شوید:

۱- در مثال اول قصد داریم نام و نام خانوادگی را در دو متغیر firstName و lastName ذخیره کنیم و با استفاده از f string آن ها را با یک رشته در خروجی چاپ کنیم:

```
1.>>> firstName = "Ali"
2.>>> lastName = "Alavi"
3.>>> example = f'My Name is {firstName} {lastName}'
4.>>> print(example)
5.My Name is Ali Alavi #خروجی
```

همانطور که می بینید برای تعریف f string باید کاراکتر f را قبل از علامت کوتیشن قرار دهید و برای استفاده از متغیر ها داخل رشته، کافی است تا نام متغیر را میان دو علامت { } قرار دهید.

۲- در مثال دوم میخواهیم حاصل یک معادله ریاضی را داخل یک رشته در خروجی چاپ کنیم:

```
1.>>> example = f'Multiply of 4 and 5 = {4*5}'
2.>>> print(example)
3.Multiply of 4 and 5 = 20 #خروجی
```

۳- فرض کنید که میخواهیم عدد Pi (۳.۱۴۱۵۹۲۶۵۳۵۹) را در خروجی چاپ کنیم، اما میخواهیم آن را فقط با دو رقم اعشار نمایش دهیم :

```
1.>>> Pi = 3.14159265359
2.>>> example = f'Pi = {Pi:.2f}'
3.>>> print(example)
4.Pi = 3.14 #خروجی
```

در قدم اول مقدار عددی را داخل متغیر Pi ذخیره کردیم و با قرار دادن f۲: جلوی نام متغیر به پایتون گفتیم که عدد داخل متغیر Pi را فقط با ۲ رقم اعشار نمایش دهد.

آشنایی به String Methods

در پایتون متد هایی وجود دارند که با استفاده از آنها می توانیم تغییراتی را در string ها ایجاد کنیم درواقع می توان گفت هر کدام از این متد ها کار خاصی را بر روی string ها انجام می دهند، در ادامه توضیحات و مثال هایی را در مورد متد های پر کاربرد بیان می کنیم.

• متد capitalize()

متد capitalize() حرف اول string را به صورت حروف بزرگ نمایش می دهد:

```
1. >>> example = "this is a string"
2. >>> print(example.capitalize())
3. This is a string      #خروجی
```

• متد count()

با استفاده از متد count() پایتون در خروجی تعداد تکرار یک کاراکتر را چاپ میکند:

```
1. >>> example = "this is a string"
2. >>> print(example.count('i'))
3. 3      #خروجی
```

• متد isalpha()

متد isalpha() در صورتی True را در خروجی چاپ می کند که داخل رشته فقط حروف وجود داشته باشد:

```
1. >>> alpha = "example"
2. >>> print(alpha.isalpha())
3. True      #خروجی
```


- متد isdigit()

متد isdigit() در صورتی True را در خروجی چاپ می کند که داخل رشته فقط اعداد وجود داشته باشد:

```
1. >>> num = "123456"
2. >>> print(num.isdigit())
3. True      #خروجی
```

- متد isalnum()

متد isalnum() در صورتی True را در خروجی چاپ می کند که رشته مخلوطی از حروف و اعداد باشد:

```
1. >>> year = "thisis2020"
2. >>> print(year.isalnum())
3. True      #خروجی
```

- متد isspace()

متد isspace() در صورتی True را در خروجی چاپ می کند که رشته فقط شامل فضای خالی (whitespace) باشد:

```
1. >>> space = ' '
2. >>> print(space.isspace())
3. True      #خروجی
```

- متد lower()

متد lower() تمام حروف بزرگ داخل یک رشته را به حروف کوچک تبدیل می کند:

```
1. >>> up = "I LOVE PYTHON"
2. >>> print(up.lower())
3. i love python #خروجی
```

- متد upper()

متد upper() تمام حروف کوچک داخل رشته را به حروف بزرگ تبدیل می کند:

```
1. >>> low = "this is python 3"
2. >>> print(low.upper())
3. THIS IS PYTHON 3      خروجی
```

• متد split()

فرض کنید که یک رشته دارید و قصد دارید حروفی را که در آن با space از یکدیگر جدا شده اند را به لیست تبدیل کنید، با متد split() میتوانید این کار را انجام دهید:

```
1. >>> sp = "192.168.1.100 192.168.1.120 192.168.1.130"
2. >>> print(sp.split(' '))
3. ['192.168.1.100', '192.168.1.120', '192.168.1.130']      خروجی
```

به این نکته توجه داشته باشید ما space را مثال زدیم کاراکتر دلخواه میتواند هر چیز دیگری باشد اما این کاراکتر باید مابین دو پرانتز متد split() قرار بگیرد.

• متد replace()

متد replace() کاراکترهای داخل رشته را با کاراکترهای مورد نظر جایگزین می کند:

```
1. >>> str = "this is java"
2. >>> newstr = str.replace('java', 'python')
3. >>> print(newstr)
4. this is python      خروجی
```

در مثال بالا قصد داریم کلمه java داخل رشته str را به پایتون تغییر دهیم، متغیری با نام newstr تعریف کردیم و با متد replace() کلمه python را جایگزین کردیم، به این نکته توجه داشته باشید که کلمه ای را که قصد حذف آن را داریم باید قبل از کلمه جایگزین داخل متد replace() قرار داده شود.

لیست (List) در پایتون

لیست ها یکی دیگر از (Data Type) های موجود در پایتون هستند که یکی از پر کاربرد ترین آن ها می باشند.

داده های یک لیست داخل علامت [] قرار می گیرند و با استفاده از ویرگول , از یکدیگر جدا می شوند، هر لیست می تواند دارای چندین عنصر باشد و این عناصر می توانند شامل انواع داده های متفاوتی مثل اعداد صحیح (integer)، ممیز شناور (float) و رشته ها (string) باشند. به مثال زیر توجه کنید:

```
1.>>> list1 = [] #لیست خالی
2.>>> list2 = [1,1.5,2,"Hello"]
3.>>> print(list1)
4.[1, 1.5, 2, 'Hello'] #خروجی
```

همانطور که مشاهده می کنید در خط اول یک لیست خالی تعریف کردیم و پایتون اجازه این کار را به ما داده است و در خط دوم یک لیست شامل داده های مختلف ایجاد کردیم و در خط آخر آن را با دستور print() در خروجی چاپ کردیم.

همچنین یک لیست می تواند شامل یک یا چند لیست دیگر باشد که به آنها لیست تو در تو (nested list) گفته می شود:

```
1.>>> nested = ["hello",[1,2,4],['a','b','c']]
2.>>> print(nested)
3.['hello', [1, 2, 4], ['a', 'b', 'c']] #خروجی
```

در لیست nested رشته "hello" وجود دارد همچنین ['a','b','c'] و [1,2,4] هر دو یک لیست مجزا هستند که در آن قرار گرفتند درواقع لیستی ایجاد شده است که خود شامل دو لیست دیگر است.

دست‌رسی به عناصر داخل لیست

برای دست‌رسی به عناصر داخل یک لیست می‌توانیم از عملگر `[]` استفاده کنیم، و باید به این نکته توجه داشت که اندیس لیست‌ها در پایتون از عدد ۰ شروع می‌شود، و برای مثال اگر لیستی دارای ۶ عنصر باشد می‌توانیم از اعداد ۰ تا ۵ برای شمارش عناصر داخل لیست استفاده کنید و این نکته هم مهم است که این اعداد باید از نوع (integer) باشند و نمی‌توان برای شمارش از اعداد ممیز شناور (float) استفاده کرد:

```
1.>>> list1 = ['I', 'R', 'A', 'N']
2.>>> list1[0]
3.'I' #خروجی
4.>>> list1[2]
5.'A' #خروجی
6.
7.>>> list2 = ['IRAN', [1, 2, 3, 4, 5]]
8.>>> list2[0]
9.'IRAN' #خروجی
10.>>> list2[0][2]
11.'A' #خروجی
12.>>> list2[1][2]
13.3 #خروجی
```

همانطور که در مثال بالا می‌بینید ما دو لیست با نام‌های `list1` و `list2` تعریف کردیم، `list1` دارای ۴ عنصر از نوع رشته‌ای می‌باشد که با عملگر `[]` توانستیم هر کدام را در خروجی چاپ کنیم. اما `list2` یک لیست تو در تو می‌باشد که عنصر ۰ آن یک رشته و عنصر ۱ آن لیستی است که خود شامل عناصری از نوع اعداد صحیح می‌باشد، برا اینکه بتوانیم به هر کدام از عناصر های این لیست دست‌رسی داشته باشیم در اولین عملگر `[]` شماره اندیس داخل `list2` را صدا می‌زنیم و در عملگر بعدی شماره اندیس دلخواه در آن را وارد می‌کنیم که به عنوان مثال رشته `IRAN` اولین اندیس از `list2` می‌باشد که شماره آن ۰ است آن را در عملگر `[]` قرار دادیم و در عملگر دوم اندیس شماره ۲ را صدا زدیم که پایتون `A` را در خروجی چاپ کرد.

تغییر عناصر داخل لیست

میتوان با انتخاب شماره اندیس یک عنصر از لیست، آن را تغییر داد :

```
1.>>> myList = ['apple', 'banana', 'cherry']
2.>>> myList[1] = 'kiwy'
3.>>> print(myList)
4. ['apple', 'kiwy', 'cherry']      خروجی#
```

List Methods

• متد append() و extend()

برای اضافه کردن یک عنصر به لیست میتوان از متد append() استفاده کرد:

```
1.>>> myList = [1,2,3,4]
2.>>> myList.append(5)
3.>>> print(myList)
4. [1, 2, 3, 4, 5]      خروجی#
```

و برای اضافه کردن چند عنصر به یک لیست میتوان از متد extend() استفاده کرد:

```
1.>>> myList = [1,2,3]
2.>>> myList.extend([4,5,6])
3.>>> print(myList)
4. [1, 2, 3, 4, 5, 6]      خروجی#
```

همانطور که در مثال بالا مشاهده می کنید در مرحله اول یک لیست با چند عنصر تعریف کردیم و در ادامه با استفاده از متد extend() لیست دیگری از عناصر را به آن اضافه کردیم.

- متد insert()

میتوان با استفاده از متد insert() یک عنصر را در موقعیت مشخصی داخل لیست قرار داد:

```
1.>>> myList = ['apple', 'banana', 'cherry']
2.>>> myList.insert(1, 'kiwy')
3.>>> print(myList)
4. ['apple', 'kiwy', 'banana', 'cherry'] #خروجی
```

- متد clear()

با استفاده از متد clear() میتوان تمام عناصر را از لیست حذف کرد:

```
1.>>> myList = ['apple', 'banana', 'cherry']
2.>>> myList.clear()
3.>>> print(myList)
4. [] #خروجی
```

- متد pop()

این متد یک عنصر را از اندیس مورد نظر حذف میکند و مقدار آن را باز می گرداند:

```
1.>>> myList = ['apple', 'kiwy', 'banana', 'cherry']
2.>>> myList.pop(1)
3. 'kiwy' #خروجی
4.>>> print(myList)
5. ['apple', 'banana', 'cherry']
```

- متد remove()

این متد یک عنصر دلخواه را از لیست حذف میکند:

```
1.>>> myList = ['C++', 'Python', 'Ruby', 'Perl']
2.>>> myList.remove('Ruby')
3.>>> print(myList)
4. ['C++', 'Python', 'Perl'] #خروجی
```

- متد sort()

این متد عناصر موجود در لیست را به ترتیب صعودی مرتب میکند:

```
1.>>> numbers = [3,5,8,4,9,7,1,6,2]
2.>>> numbers.sort()
3.>>> print(numbers)
4.[1, 2, 3, 4, 5, 6, 7, 8, 9] # خروجی
```

- متد reverse()

این متد ترتیب عناصر در لیست را معکوس میکند:

```
1.>>> myList = ['C++', 'Python', 'Ruby', 'Perl']
2.>>> myList.reverse()
3.>>> print(myList)
4.['Perl', 'Ruby', 'Python', 'C++'] # خروجی
```

تاپل ها (Tuples)

تاپل ها نوعی دیگر داده ها در پایتون هستند که میتوان گفت درست شبیه لیست ها می باشند با تفاوت اینکه مقادیر داخل آنها غیر قابل ویرایش است؛ همچنین عناصر یک تاپل داخل () تعریف می شود و برای جدا کردن آنها از , استفاده می شود:

```
1.>>> tup1 = ('mobin', 1, 3.14, "x")
2.>>> tup2 = () # تاپل خالی
```

همانطور که در مثال بالا مشاهده میکنید حتی اگر تاپل دارای یک عنصر باشد باید بعد از آن از , استفاده شود.

میتوان از متد `len()` برای شمارش تعداد عناصر یک تاپل استفاده کرد:

```
1.>>> tup1 = ('cisco', 2800, 12.4)
2.>>> len(tup1)
3.3
```

و همچنین با استفاده از متد `del()` میتوان یک تاپل را حذف کرد:

```
1.>>> tup1 = ('cisco', 2800, 12.4)
2.>>> del(tup1)
3.>>> print(tup1)
4.Traceback (most recent call last):
5. File "<stdin>", line 1, in <module>
6.NameError: name 'tup1' is not defined
```

```
1. ('python', 1998) # خروجی
```


تبدیل تاپل به لیست

میتوان با استفاده از متد `tuple()` یک لیست را به تاپل تبدیل کرد:

```
1.>>> list1 = ['python', 1998]
2.>>> tup1 = tuple(list1)
3.>>> print(tup1)
4.('python', 1998) #خروجی
```

دیکشنری (Dictionary)

دیکشنری ها یکی دیگر از انواع داده های موجود در پایتون هستند که بر خلاف لیست ها و تاپل ها فاقد ترتیب از عناصر هستند و در دیکشنری یک جفت (کلید : مقدار) وجود دارد و با توجه به این نکته که مقادیر میتوانند از انواع مختلفی از داده ها باشند و غیر قابل تکرار، همچنین مقادیر دیکشنری داخل {} قرار میگیرند.

به مثال های زیر توجه کنید:

```
1.>>> dict1 = {} # دیکشنری خالی
2.>>> dict2 = {1: 'mobin', 2: '1998'}
3.>>> dict3 = {'name': 'Mobin', 'number': [1, 2, 3, 4, 5]}
```

دسترسی به عناصر دیکشنری

در بقیه داده ها مثل لیست و تاپل برای دسترسی به عناصر از اندیس دهی استفاده میکردیم اما در دیکشنری از کلید ها استفاده میشود، برای اینکار میتوان نام کلید را در [] قرار داد یا با متد `get()` آن را فراخوانی کرد و باید به این نکته توجه کنید: اگر از متد `get()` استفاده می کنید در صورتی که کلید مورد نظر در دیکشنری وجود نداشته باشد به جای خطای `KeyError` مقدار `None` باز خواهد گشت:

```
1.>>> dict1 = {'name': 'Mobin', 'number': [1, 2, 3, 4, 5]}
2.>>> print(dict1['name']) # استفاده از []
3.Mobin # خروجی
4.>>> print(dict1.get('number')) # get() از متد
5.[1, 2, 3, 4, 5] # خروجی
```

تغییر یا اضافه کردن عناصر دیکشنری

عناصر دیکشنری قابل تغییر هستند و همچنین میتوان یک عنصر جدید را به دیکشنری افزود:

```
1.>>> dict1 = {'name': 'John', 'age': 23}
2.>>> dict1['age'] = 25 # age مقدار کلید تغییر
3.>>> print(dict1)
4.{'name': 'John', 'age': 25} # خروجی
5.
6.>>> dict1['number'] = 12282 # number ایجاد کلید
7.>>> print(dict1)
8.{'name': 'John', 'age': 25, 'number': 12282} # خروجی
```

حذف عناصر دیکشنری

- متد `pop()`:

میتوان با استفاده از متد `pop()` یک عنصر خاص را بر اساس نام کلید حذف کرد و این متد مقدار آن را در خروجی چاپ می کند:

```
1.>>> dict1 = {'name': 'John', 'age': 25, 'number': 12282}
2.>>> print(dict1.pop('age'))
3.25 # خروجی
```

- متد `clear()`:

میتوان با استفاده از متد `clear()` برای حذف تمام عناصر یک دیکشنری استفاده کرد:

```
1.>>> dict1 = {'name': 'John', 'age': 25, 'number': 12282}
2.>>> dict1.clear()
```

- متد `del()`:

با استفاده از متد `del()` میتوان یک کلید خاص را از دیکشنری حذف کرد و همچنین میتوان یک دیکشنری را به صورت کامل حذف کرد:

```
1.>>> dict1 = {'name': 'John', 'age': 25, 'number': 12282}
```

```
2.>>> del dict1['age'] # حذف یک کلید خاص
3.>>> del dict1        # حذف
```

ساخت دیکشنری تو در تو

در قسمت قبل با Dictionary ها آشنا شدید و میدانید که هر دیکشنری تشکیل شده از یک یا چند زوج (key : value)، میتوانیم value را به صورت یک دیکشنری تعریف کنیم:

```
1.>>> tobuy = {'fruit': {'banana':2, 'apple':3, 'cherry':5},
              'office': {'pen':2, 'notebook':6}} # دیکشنری تو در تو
2.>>> print(tobuy['fruit']) # fruit مقدار کلید
3.{'cherry', 'apple', 'banan'}
4.>>> print(tobuy['fruit']['banana'])
5. خروجی #2
```

ست (Set)

set یکی دیگر از انواع داده ها در پایتون می باشد که برای نگه داری تعدادی عنصر به کار می رود ولی عناصر داخل set بدون ترتیب هستند یعنی بر خلاف لیست ها هر عنصر یک شماره index را شامل نمی شود، و نکته مهم تر اینکه نمیتوان مقادیر داخل یک set را تغییر داد همچنین عناصر داخل set نمی توانند تکراری باشند.

عناصر یک لیست داخل {} قرار می گیرند:

```
1.>>> set1 = {1,2,3}      # set تعریف
2.>>> print(set1)
3.{1, 2, 3}              #خروجی
```

میتوان با استفاده از متد add() یک عنصر را به set اضافه کرد:

```
1.>>> set1 = {1,2,3}      # set تعریف
2.>>> set1.add(4)         # اضافه کردن یک عنصر
3.>>> print(set1)
4.{1, 2, 3, 4}           #خروجی
```

میتوان با استفاده از متد update() یک لیست را به set اضافه کرد:

```
1.>>> set1 = {1,2,3}
2.>>> set1.update([4,5,6]) #set اضافه کردن یک لیست به
3.>>> print(set1)
4.{1, 2, 3, 4, 5, 6}     #خروجی
5.
```

همچنین برای حذف یک عنصر از set از متد discard() استفاده کرد:

```
1.>>> set1 = {1,2,3,4,5,6} # set تعریف
2.>>> set1.discard(6)      #set حذف یک عنصر از
3.>>> print(set1)
4.{1, 2, 3, 4, 5}        #خروجی
```

دستورات شرطی در پایتون

دستور if

برای ایجاد شرط در پایتون شما می توانید از `if elif else` استفاده کنید که در این ساختار از یک عبارت شرطی مقابل `if` استفاده می شود که اگر این عبارت `True` باشد `Action` مورد نظر را بعد از `if` انجام می دهد در غیر این صورت از `elif` و `else` استفاده می کند استفاده از `elif` و `else` به صورت `Optional` می باشد.

نکته:

در دستورات `if` و یا حلقه ها بایستی حتما بلاک ها توسط `Wide Space` مشخص شود تا مشخص شود که کدام `Block` مربوط به چه قسمتی می باشد که در اصطلاح به این `Wide Space` در زبان پایتون `indented Block` گفته می شود شما می توانید این `Wide Space` را با استفاده از کلید `TAB` ایجاد کنید.

دقت کنید که بعد از شرط `if` بایستی حتما از علامت : استفاده شود و سپس `Enter` را بزنید تا وارد خط بعد شوید و سپس بایستی از `Wide Space` استفاده کنید به عبارت دیگر بایستی یکبار کلید `TAB` را بزنید تا بلاک مورد نظر زیر `if` قرار بگیرد و بعد از ایجاد بلاک دو بار `Enter` را بزنید تا از `if` خارج شوید.

```
1.>>> x = 20
2.>>> if x > 5:
3....     print("x is greater than 5")
4....
5.x is greater than 5
```

در صورتی که `if` بالا را بدون استفاده از `Wide Space` وارد کنید با خطا زیر روبرو می شوید:

```
1.>>> x = 20
2.>>> if x > 5:
```

```

3.... print("x is greater than 5")
4.   File "<stdin>", line 2
5.       print("x is greater than 5")
6.       ^
7.IndentationError: expected an indented block

```

دستور else

همانطور که در بخش قبلی مشاهده کردید دستورات داخل if در صورتی اجرا می شدند که شرط آن True باشد، اگر شرط False شد چه دستوراتی باید اجرا شوند؟؟

میتوان با استفاده دستور else مشخص کرد که اگر شرط دستور if غلط بود، دستورات داخل else اجرا شود، میتوان گفت که else به معنای "مگر نه" است:

```

1.>>> a = 10
2.>>> b = 5
3.>>> if b > a:
4....     print("b is greater than a")
5.... else:
6....     print("a is greater than b")
7....
8.a is greater than b

```

خروجی #

در مرحله اول شرط داخل if بررسی می شود و به خاطر اینکه False است، دستورات داخل else اجرا خواهند شد و در خروجی رشته a is greater than b چاپ می شود.

در صورتی که نیاز باشد برنامه چند شرط را بررسی کند یعنی در صورت False شدن شرط اول شرط دوم را بررسی کند و به همین شکل تا آخرین شرطی که ما برای آن مشخص کردیم ادامه دهد میتوان از دستور elif استفاده کرد (elif عبارت کوتاه شده else if است).

و باید به این نکته توجه داشت: دستور if تنها میتواند یک دستور else داشته باشد، اما میتواند شامل چندین دستور elif باشد.

مثالی برای دستورات if, else و elif:

```
1.>>> num = 3.14
2.>>> if num>0:      # شرط اول
3....     print("Positive Number")
4.... elif num == 0:  # شرط دوم
5....     print("Zero")
6.... else:
7....     print("negetive number")
8....
9.Positive Number    # خروجی
```

در مثال بالا شرط اول بررسی میشود و در صورتی که True بود رشته Positive Number در خروجی چاپ خواهد شد اما اگر False شد برنامه شرط دوم را بررسی میکند که با دستور elif نوشته شده است و در صورتی که هیچکدام True نباشند دستورات داخل else اجرا خواهند شد و رشته negative number در خروجی چاپ خواهد شد.

حلقه ها در پایتون

حلقه for

حلقه For زمانی استفاده می شود که شما می خواهید یک Code را برای گروهی از Sequence ها یا Element ها اجرا کنید که این Sequence می تواند لیست یا String یا هر چیز دیگری باشد.

استفاده از حلقه for در LIST

```
1.>>> vendor = ["cisco", "Microsoft","hp", "avaya"]
2.>>> for i in vendor:
3....     print(i)
4....
5.cisco
6.Microsoft
7.hp
8.avaya
```

استفاده از حلقه for در String

```
1.>>> var = "python"
2.>>> for x in var:
3....     print(x)
4....
5.p
6.y
7.t
8.h
9.o
10.n
```

استفاده از تابع Range در حلقه For

```
1.>>> for i in range(5):
2....     print(i)
3....
4.0
5.1
6.2
7.3
8.4
```

ایجاد لیستی از آدرس های IP با استفاده از حلقه For و تابع Range

```
1.>>> for x in range(90,100):
2....     ip = "192.168.1." + str(x)
3....     print(ip)
4....
5.192.168.1.90
6.192.168.1.91
7.192.168.1.92
8.192.168.1.93
9.192.168.1.94
10.192.168.1.95
11.192.168.1.96
12.192.168.1.97
13.192.168.1.98
14.192.168.1.99
```

استفاده از else در حلقه for

شما می توانید در حلقه های For از Else هم استفاده کنید برای زمانی که حلقه به اتمام رسیده و می خواهید بعد از اتمام حلقه کاری انجام شود:

```
1.>>> vendor = ["cisco", "Microsoft","hp", "avaya"]
2.>>> for i in vendor:
3....     print(i)
```

```
4.... else:
5....     print("The End")
6....
7.cisco
8.Microsoft
9.hp
10.avaya
11.The End
```

حلقه while

یکی دیگر از مدل های Loop در زبان پایتون While می باشد بر خلاف For Loop که یک Code را برای یک Sequence از Element ها اجرا می کرد While Loop یک Code را تا زمانی که شرط حلقه True باشد اجرا می کند و تا زمانی شرط False نشود Loop تا بینهایت ادامه دارد :

```
1.>>> x = 1
2.>>> while x <= 10:
3....     print (x)
4....     x = x+1
5....
6.1
7.2
8.3
9.4
10.5
11.6
12.7
13.8
14.9
15.10
```

استفاده از else در حلقه while

شما می توانید در حلقه های While از Else نیز استفاده کنید و برای زمانی استفاده می شود که می خواهید مشخص کنید که در صورتی که شرط حلقه False می شود چه کاری انجام شود:

```
1.>>> while x <= 5:
2....     x = x + 1
3....     print(x)
4.... else:
5....     print("out of while loop")
6....
7.2
8.3
9.4
10.5
11.6
12.out of while loop
```

حلقه های تو در تو (Nested Loop)

شما می توانید یک Loop را در داخل یک Loop دیگر تعریف کنید.

Nested If Loop

```
1. Nested If Loop
2. >>> x = "cisco"
3. >>> if "i" in x:
4. ....     if len(x) > 3:
5. ....         print(x)
6. ....
7. cisco
```

همچنین میتوان دستورات if بالا را به این صورت نوشت:

```
1. >>> x = "cisco"
2. >>> if ("i" in x) and (len(x) > 3):
3. ....     print(x)
4. ....
5. cisco
```

Nested For Loop

```
1. >>> list1 = [4,5,6]
2. >>> list2 = [10,20,30]
3. >>> for i in list1:
4. ....     for j in list2:
5. ....         print(i * j)
6. ....
7. 40
8. 80
9. 120
10. 50
11. 100
12. 150
13. 60
14. 120
```

```
1.>>> x = 1
2.>>> while x <= 5:
3....     z = 3
4....     x = x + 1
5....     while z <= 5:
6....         print(z)
7....         z = z + 1
8....
9.3
10. 4
11. 5
12. 3
13. 4
14. 5
15. 3
16. 4
17. 5
18. 3
19. 4
20. 5
21. 3
22. 4
23. 5
```

استفاده ترکیبی از حلقه های تو در تو

```
1.>>> for i in range(1,10):
2....     if i%2 != 0 :
3....         print(i)
4....
5.1
6.3
7.5
8.7
9.9
```

Break و Continue در حلقه ها

تفاوت میان Break و Continue در این است که در زمانی که شما Break را استفاده می کنید کد زیر Break دیگر اجرا نمی شود و سراغ مقادیر بعدی نمی رود ولی در صورتی که شما از Continue استفاده کنید زمانی که شرط مربوط به Continue برقرار شود کد مربوط به آن اجرا نمی شود و از مقادیر بعدی استفاده می کند به عبارت دیگر Loop را برای مقادیر بعدی ادامه می دهد.

مثال break :

```
1.>>> for i in "python":
2....     print(i)
3....     if i == "h":
4....         break
5....
6.p
7.y
8.t
9.h
```

```
1.مثالContinue :
2.>>> for i in "python":
3....     if i == "h":
4....         continue
5....     print(i)
6....
7.p
8.y
9.t
10.o
11.n
```

استفاده از pass

در صورتی که شما بخواهید کدی را بنویسید که بعداً بخواهید از آن استفاده کنید ولی فعلاً نمی‌خواهید اجرا شود می‌توانید از دستور Pass استفاده کنید:

```
1.>>> for i in range(10):  
2....     pass  
3....
```


مدیریت خطا در پایتون

در صورتی شما در زبان پایتون Syntax Error را اشتباه وارد کنید به شما Syntax Error می دهد شما می توانید این Syntax Error ها را با استفاده از دستور Try کنترل کنید .

```
1. >>> for i in range(10):
2. ...     print(i / 0)
3. ...
4. Traceback (most recent call last):
5.   File "<stdin>", line 2, in <module>
6. ZeroDivisionError: division by zero
7. >>> for i in range(10):
8. ...     try:
9. ...         print(i / 0)
10. ...     except ZeroDivisionError:
11. ...         print("Zero division not
    Allowed!!")
12. ...
13. Zero division not Allowed!!
14. Zero division not Allowed!!
15. Zero division not Allowed!!
16. Zero division not Allowed!!
17. Zero division not Allowed!!
18. Zero division not Allowed!!
19. Zero division not Allowed!!
20. Zero division not Allowed!!
21. Zero division not Allowed!!
22. Zero division not Allowed!!
```

استفاده از else به همراه try

شما به همراه Try می توانید از دستور Else هم استفاده کنید که در صورتی که Error رخ ندهد عمل می کند:

```
1.>>> try:
2....     print(4 / 2)
3.... except NameError:
4....     print("name Error")
5.... else:
6....     print("no error")
7....
8.2.0
9.no error
```

استفاده از Finally

شما می توانید به همراه Try از دستور Finally بجای دستور Else هم استفاده کنید که در این صورت در هر شرایطی دستورات بعد از Finally نمایش داده خواهد شد:

```
1.>>> try:
2....     print(4 / 2)
3.... except NameError:
4....     print("name Error")
5.... finally:
6....     print("I dont care errors :)")
7....
8.2.0
9.I dont care errors :)
```

آشنایی با توابع

یکی از مهمترین مباحث در زبان برنامه نویسی پایتون می باشد، به دسته بندی و مرتب سازی کد برای استفاده مجدد Function نامیده می شود.

شما می توانید با استفاده از دستور `def` یک Function را ایجاد کنید از طرفی می توانید برای یک Function یک Description تعریف کنید :

```
1.>>> def my_function():
2....     """ this is my function"""
3....     print("hello world!!")
4....
```

یک Function می تواند شامل پارامتر ورودی هم باشد ولی می تواند بدون پارامتر هم باشد. برای استفاده از یک Function بایستی آن را مانند زیر صدا بزنید:

```
1.>>> def my_function():
2....     """ this is my function"""
3....     print("hello world!!")
4....
5.>>> my_function()
6.hello world!!
```

همچنین با استفاده از متد `help()` میتوان Description یک تابع را مشاهده کرد:

```
1.>>> def my_function():
2....     """ this is my function"""
3....     print("hello world!!")
4....
5.>>> print(help(my_function))
6.Help on function my_function in module __main__:
7.
8.my_function()
9.    this is my function
10.
11. None
```

ایجاد Function با پارامتر ورودی

```
1.>>> def print_name(name):
2....     print(name)
3....
4.>>> print_name("Mobin")
5.Mobin
```

یک Function می تواند شامل چندین پارامتر ورودی هم باشد:

```
1.>>> def sum(x,y):
2....     sum = x + y
3....     print(sum)
4....
5.>>> sum(2,3)
6.5
```

دستور return

شما می توانید در یک Function از متغیر هم استفاده کنید ولی بایستی از دستور Return هم استفاده کنید تا خروجی متغیر در زمان صدا زدن تابع نمایش داده شود:

```
1.>>> def sum(x,y):
2....     sum = x + y
3....     return sum
4....
5.>>> sum(2,3)
6.5
```

اعمال مقادیر پیشفرض برای تابع

شما برای یک Function می توانید از مقادیر پیش فرض هم استفاده کنید که در صورتی که کاربر مقداری وارد نکرد این مقدار پیش فرض استفاده می شود و در صورتی که کاربر مقدار مورد نظر را وارد کند با مقدار پیش فرض جایگزین خواهد شد:

```
1.>>> def my_func(x,y,z = 3):
2....     n = (x + y) * z
3....     return n
4....
5.>>> my_func(1,2)
6.9
7.>>> my_func(1,2,4)
8.12
```

Name Spaces

منظور از Name Spaces استفاده از متغیرهای خارج از Function در درون Function و یا استفاده از متغیرهای داخل Function در خارج از Function می باشد.

یک راه برای اینکه شما بتوانید از یک متغیری که خارج از Function است در درون Function استفاده کنید این است که نوع متغیر را در درون Function از نوع Global ایجاد کنید به عبارت دیگر هدف شما این است که می خواهید یک متغیر را از محیط Global در درون Function خود استفاده کنید و در صورتی که می خواهید یک متغیر را در خارج از Function استفاده کنید کفایت آن متغیر در در تابع مورد نظر Return کنید:

```
1.>>> x = 10
2.>>> def my_func():
3....     global x
4....     print(x * 2)
5....
6.>>> my_func()
7.20
```