پایتون

به زبان ساده

فصل اول: مبانی زبان پایتون

بی شک این اثر، خالی از اشکال نیست و از شما خوانندگان عزیز می‌خواهم که با نظرات و پیشنهادات خود بنده را در تکمیل و رفع نواقص آن از طریق پست الکترونیکی younes.ebrahimi.1391@gmail.com یاری بفرمایید.

برای مطالعه مطالب بیشتر به سایت **www.w3-farsi.com** مراجعه فرمایید.

راه‌های ارتباط با نویسنده

وب سایت: www.w3-farsi.com

لینک تلگرام: https://telegram.me/ebrahimi_younes

ID تلگرام: @ebrahimi_younes

پست الکترونیکی: younes.ebrahimi.1391@gmail.com

فصل اول

مبانی زبان پایتون

پایتون چیست

پایتون (Python) یک زبان برنامه‌نویسی همه منظوره، شیءگرا و متن باز است که توسط خودو فان راسام (Guido van Rossum) در سال ۱۹۹۱ در کشور هلند طراحی شد. این زبان از زبان‌های برنامه‌نویسی مفسر بوده و به صورت کامل یک زبان شیءگرا است که به زبانهای تفسیری Perl و Ruby شباهت دارد و از مدیریت خودکار حافظه استفاده می‌کند.

پایتون، کد باز (Open Source) است، زبانی که گوگل و یا یاهو از آن به عنوان یکی از اصلی‌ترین ابزارهای توسعه استفاده می‌کنند. برنامه‌های پایتون مثل PHP قابل اجرا روی اغلب سیستم عامل‌هاست. پایتون، دستور زبانی شبیه گفتار ساده‌ی انگلیسی دارد و با دارا بودن ۳۳ کلمه کلیدی جزء ساده‌ترین زبان‌ها است.

سادگی و خوانایی از ویژگی‌های بارز زبان برنامه‌نویسی پایتون است، آنچنان ساده که حتی کودکان نیز قادر به آموختن آن هستند و قدرت در کنار این سادگی و خوانایی، معجزه پایتون می‌باشد. از نگاه هر برنامه‌نویسی، برنامه‌های پایتون مجموعه‌ای از کدهای زیبا هستند، بدون هیچ آشفتگی و پیچیدگی. جالب است بدانید مایکروسافت نیز این زبان را با نام IronPython در تکنولوژی .Net خود گنجانده است.

هم اکنون پایتون در شرکت‌ها و سازمان‌های بزرگی مثل ناسا و گوگل و یاهو و ... به صورت گسترده مورد استفاده قرار می‌گیرد. تا کنون نسخه‌های مختلفی از این زبان ارائه شده است که لیست آنها را در جدول زیر مشاهده می‌کنید:

نسخه	تاریخ پیدایش
Python 1.0	January 1994
Python 1.2	April 10, 1995
Python 1.3	October 12, 1995
Python 1.4	October 25, 1996
Python 1.5	December 31, 1997
Python 1.6	September 5, 2000
Python 2.0	October 16, 2000
Python 2.1	April 17, 2001
Python 2.2	December 21, 2001
Python 2.3	July 29, 2003
Python 2.4	November 30, 2004

Python 2.5	September 19, 2006
Python 2.6	October 1, 2008
Python 2.7	July 3, 2010
Python 3.0	December 3, 2008
Python 3.1	June 27, 2009
Python 3.2	February 20, 2011
Python 3.3	September 29, 2012
Python 3.4	March 16, 2014
Python 3.5	September 13, 2015
Python 3.6	December 23, 2016
Python 3.7	October 20, 2018

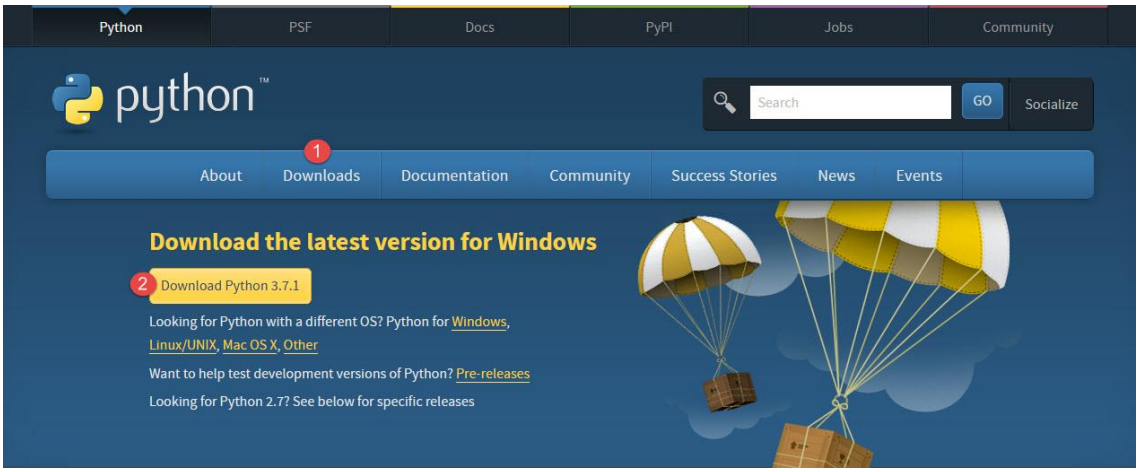
حال که با پایتون به طور مختصر آشنا شدید، در درس‌های بعد در مورد این زبان برنامه‌نویسی بیشتر توضیح می‌دهیم.

دانلود و نصب Python 3.7

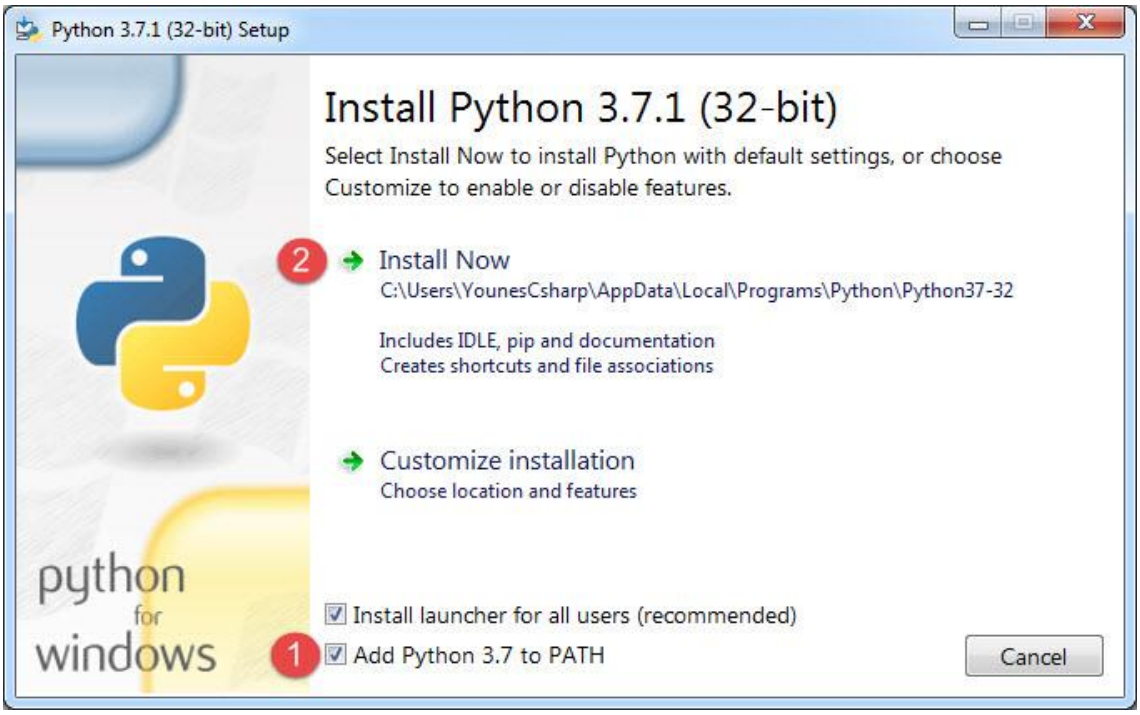
برای برنامه‌نویسی به زبان‌های مختلف محیط‌های توسعه‌ی یکپارچه یا IDE های مختلفی وجود دارند که به برنامه نویسان در نوشتن و ویرایش کدها، پیدا کردن خطاها، نمایش خروجی، و برخی موارد دیگر کمک می‌کنند. برای اجرای کدهای Python محیط‌های مختلفی وجود دارد که ساده‌ترین آنها، IDEL می‌باشد. برای دانلود این محیط کدنویسی، بر روی لینک زیر کلیک کنید:

```
https://www.python.org/downloads
```

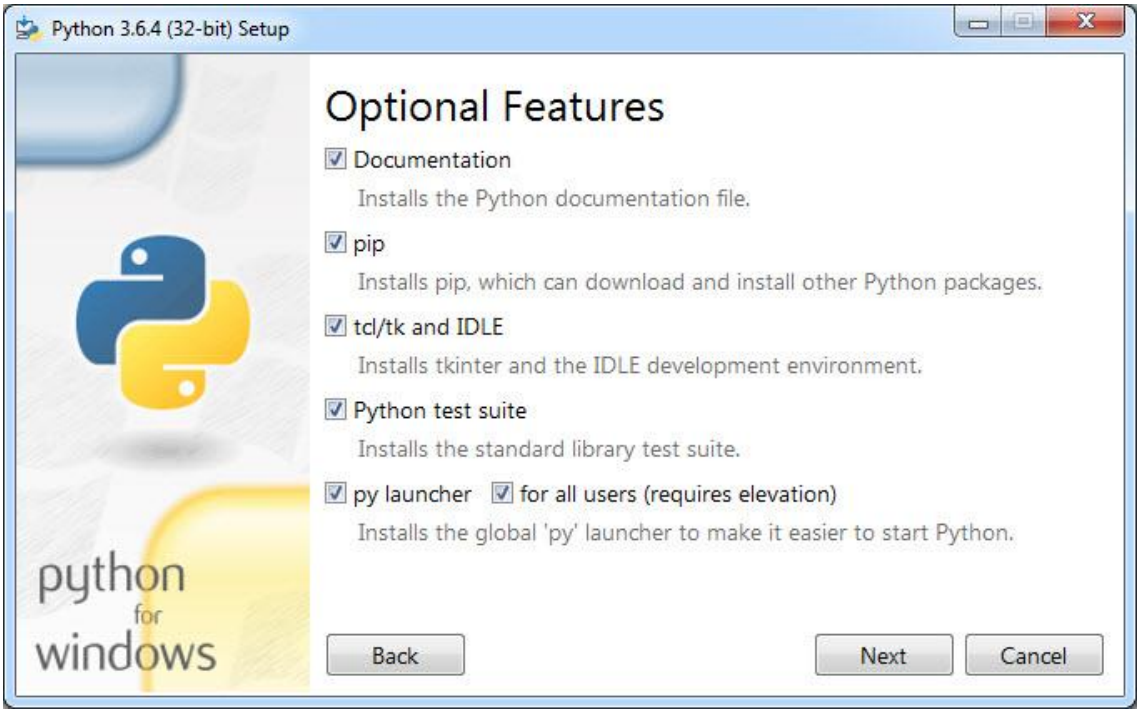
با کلیک بر روی لینک بالا، صفحه‌ای به صورت زیر باز می‌شود، که در این صفحه همانطور که در شکل زیر مشاهده می‌کنید، بر روی دکمه‌ای که با فلش نشان داده شده کلیک کرده، تا آخرین نسخه IDEL دانلود شود:



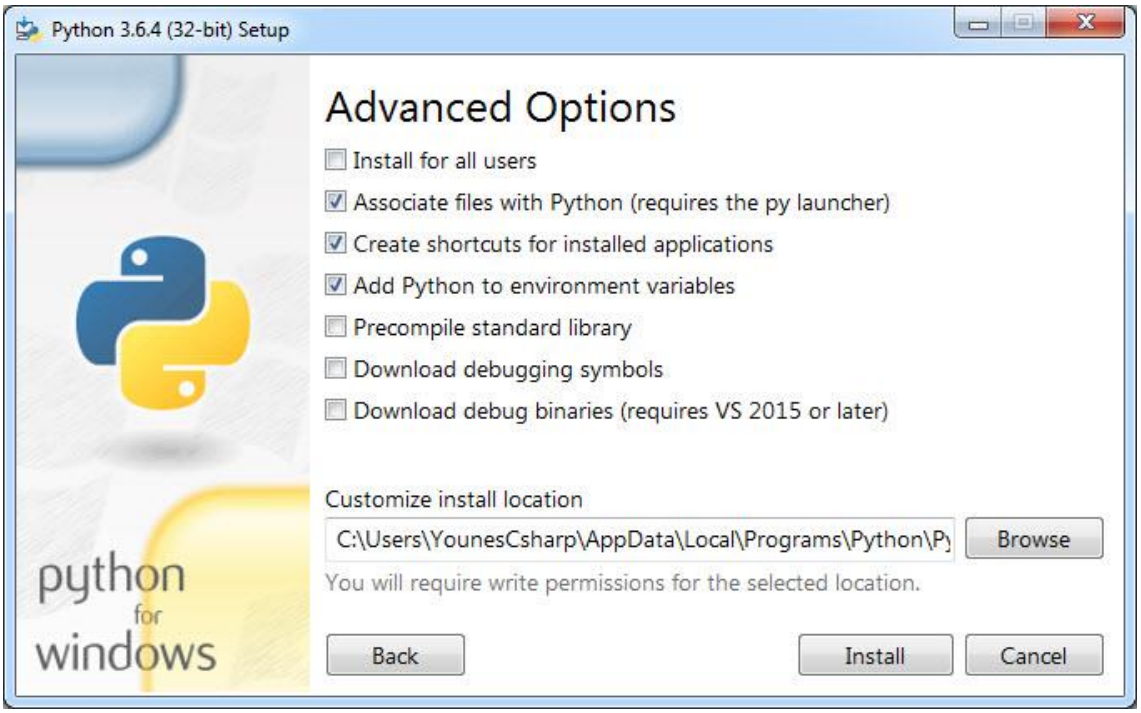
بعد از دانلود فایل مورد نظر به محل ذخیره آن رفته و بر روی فایل دو بار کلیک کنید. سپس در صفحه‌ای که به صورت زیر نمایش داده می شود، تیک مورد نظر را زده و سپس بر روی گزینه `Customize installiation` کلیک کنید:



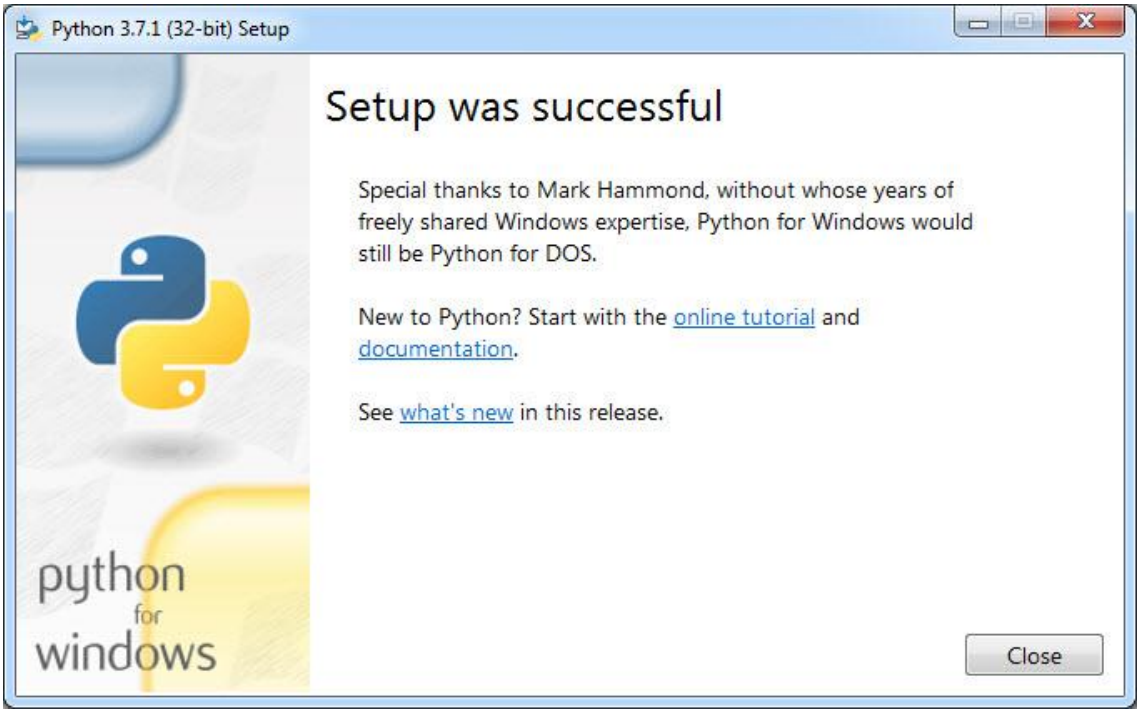
بعد از کلیک بر روی گزینه مذکور صفحه‌ای به صورت زیر نمایش داده می‌شود. در این صفحه بر روی دکمه `Next` کلیک کنید:



در صفحه بعد بر روی دکمه `Install` کلیک کرده و منتظر بمانید تا برنامه نصب شود:

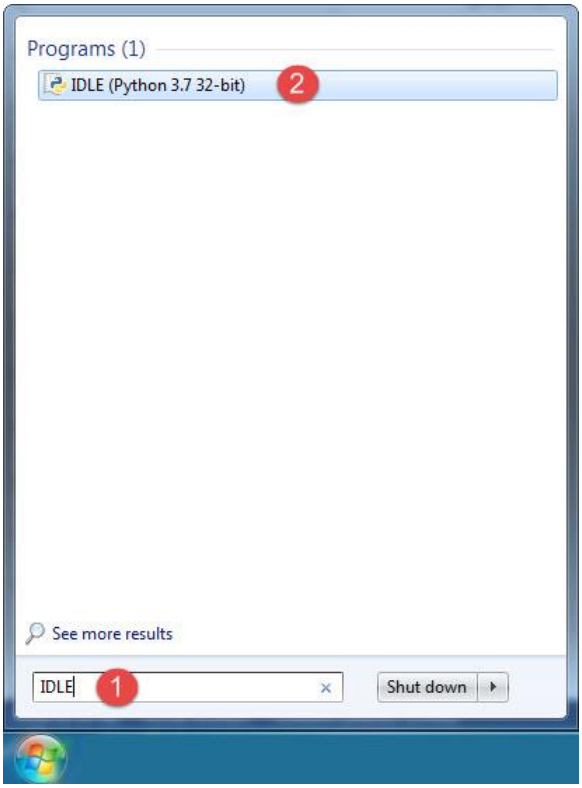


در آخر و بعد از نصب کامل برنامه پیغامی مبنی بر موفقیت آمیز بودن، نصب برنامه به شما نمایش داده می‌شود و شما می‌توانید دکمه `Close` را بزنید:

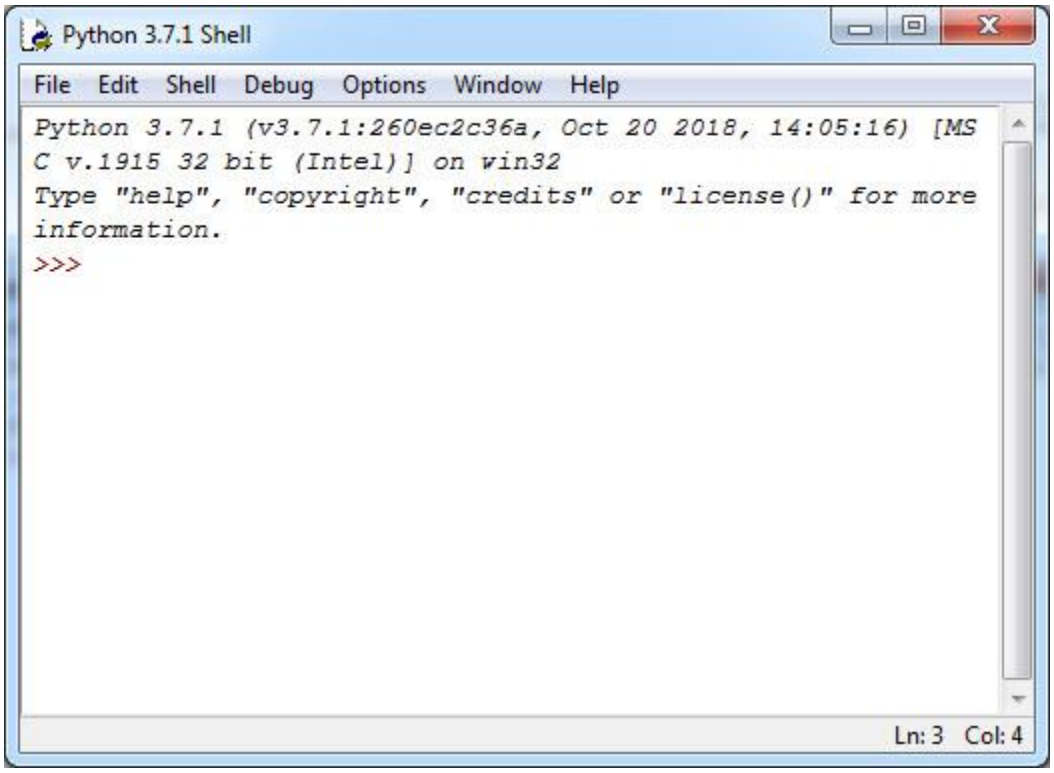


ساخت یک برنامه ساده

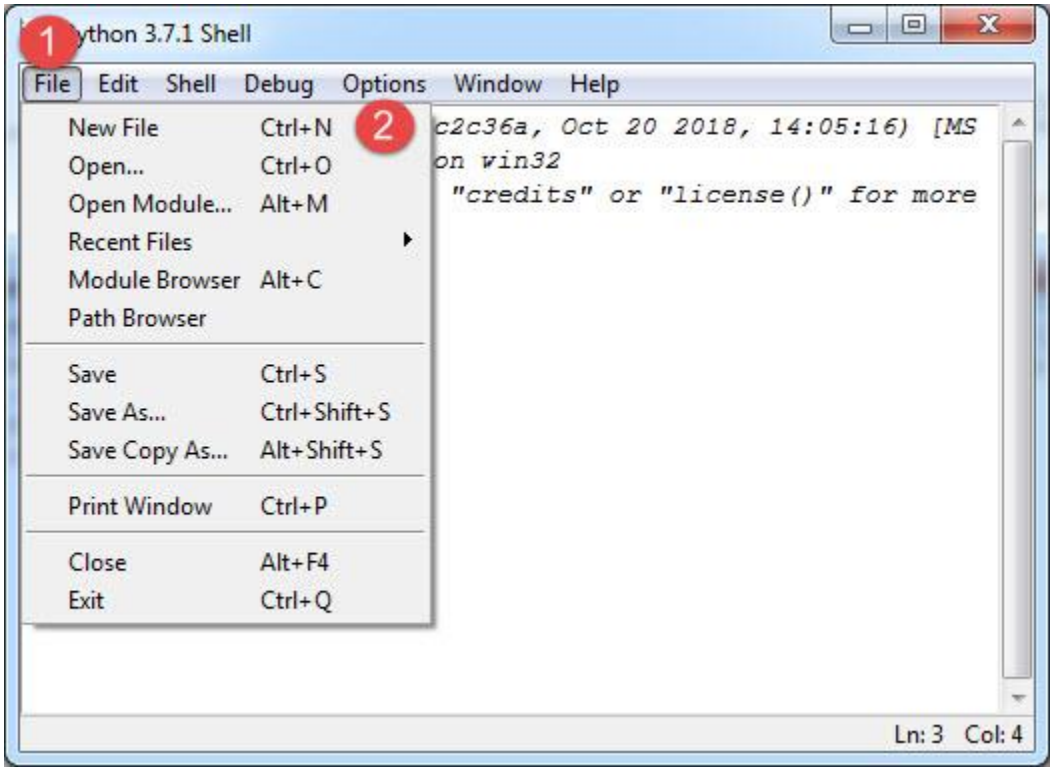
اجازه بدهید یک برنامه بسیار ساده به زبان پایتون بنویسیم. این برنامه یک پیغام را نمایش می‌دهد. از منوی Start محیط برنامه‌نویسی IDEL را به صورت زیر اجرا کنید:



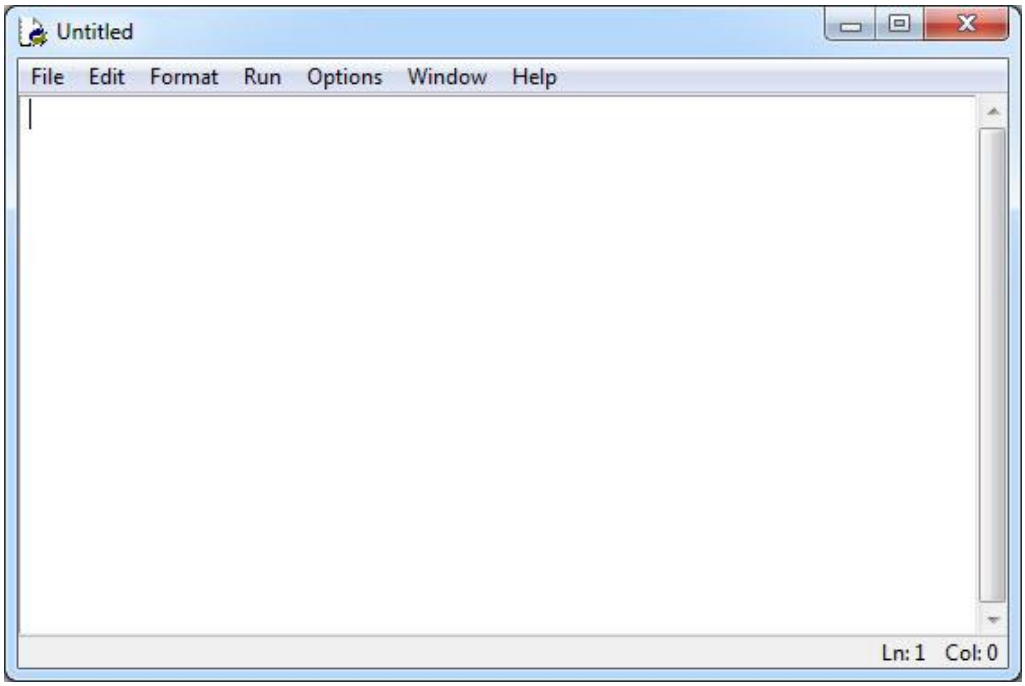
با کلیک بر روی IDEL صفحه‌ای به صورت زیر نمایش داده می‌شود:



در صفحه باز شده به صورت زیر بر روی منوی File و سپس گزینه New File کلیک کنید:

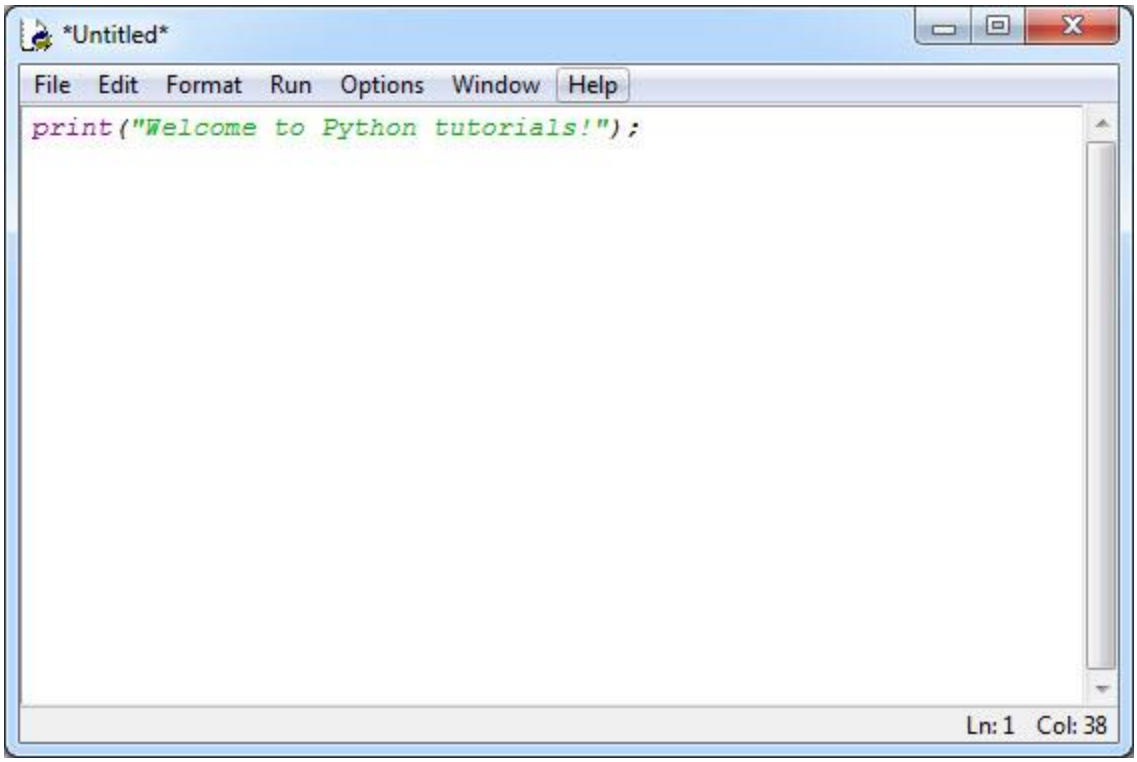


با کلیک بر روی گزینه New File صفحه‌ای به صورت زیر نمایش داده می‌شود که شما می‌توانید کدهای خود را در داخل آن بنویسید:

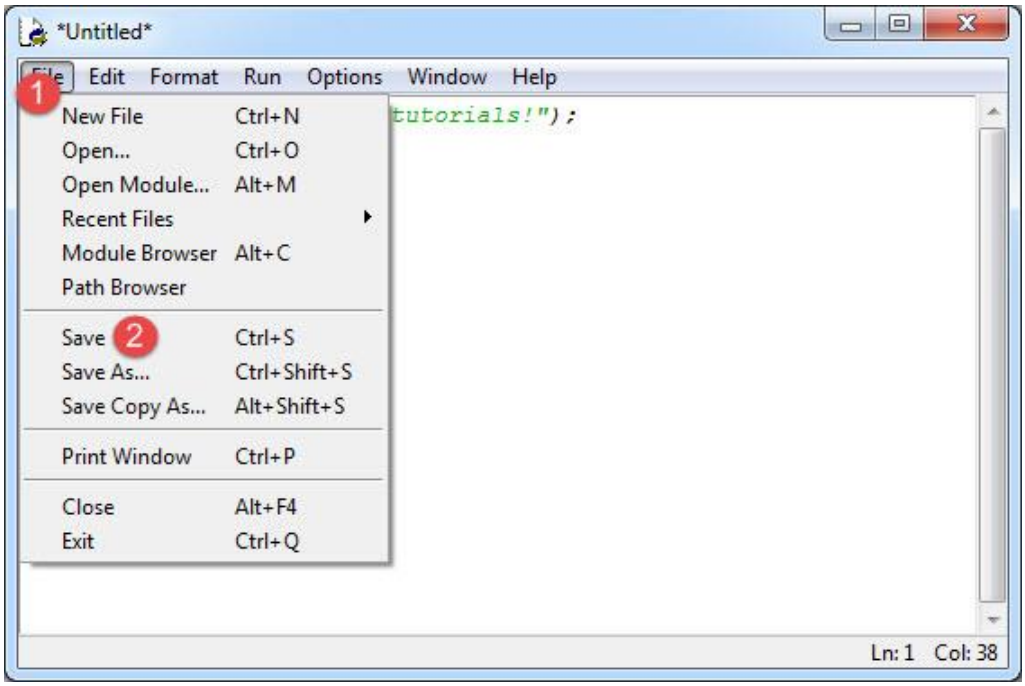


در پنجره بالا کدهای زیر را بنویسید:

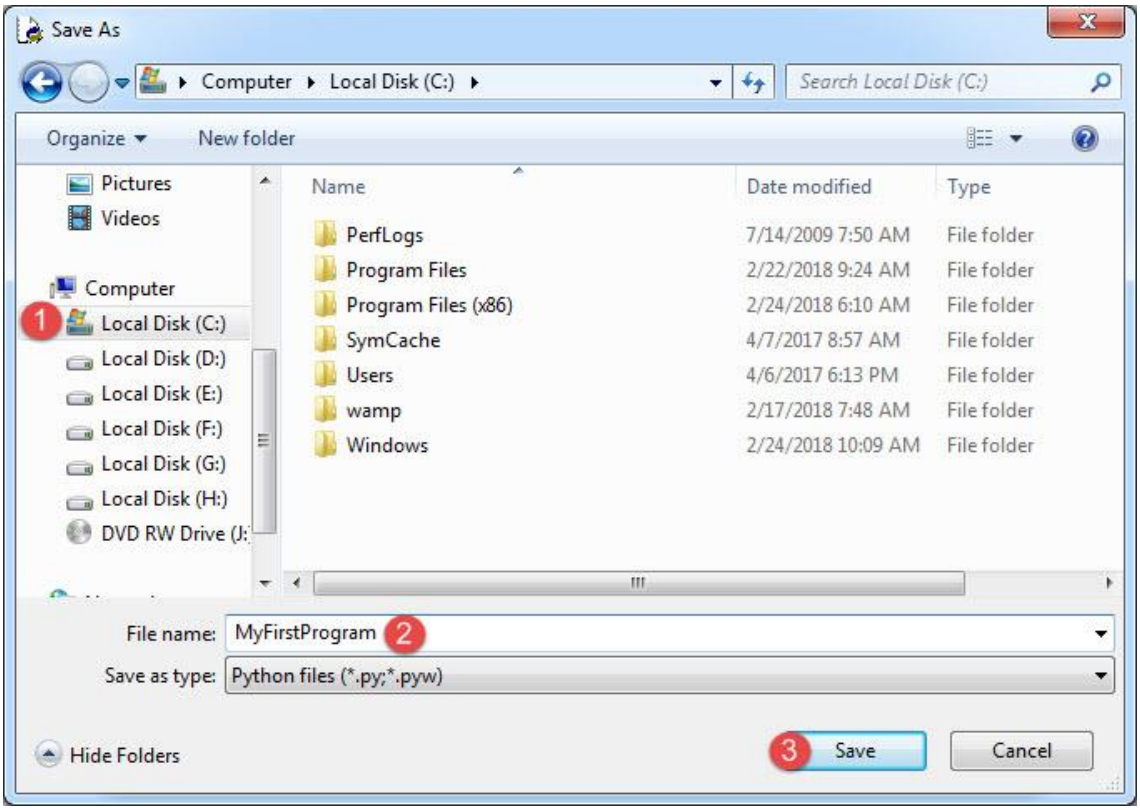
```
print("Welcome to Python Tutorials!")
```



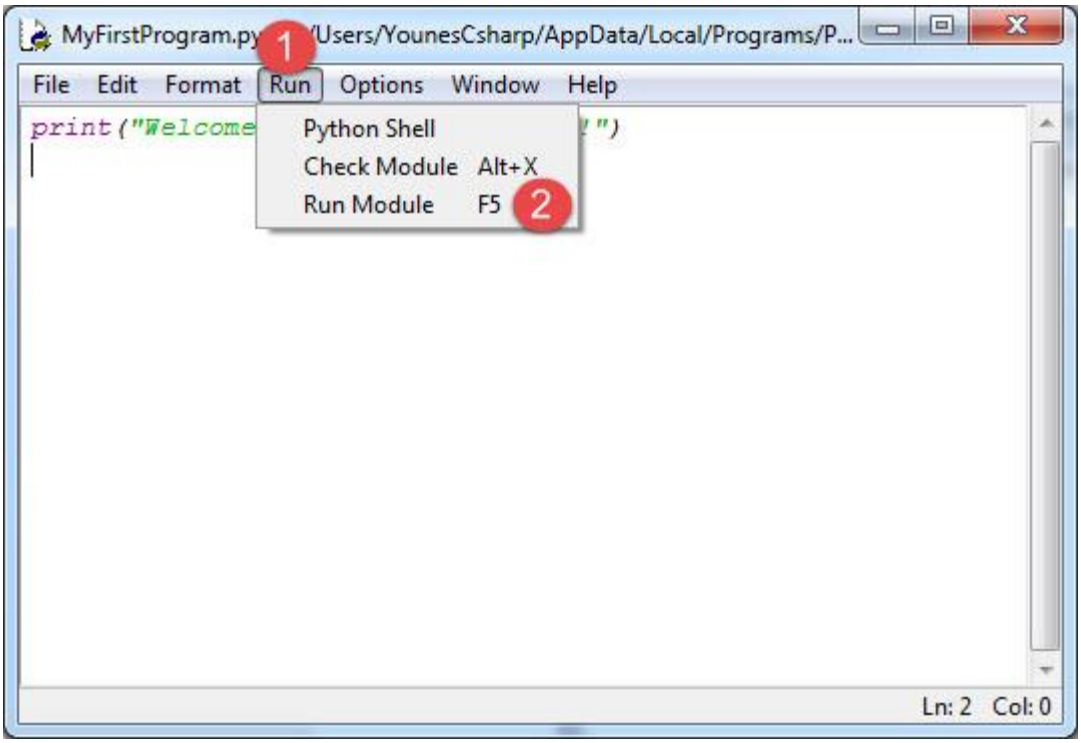
مانند شکل زیر از منوی File گزینه Save را بزنید:



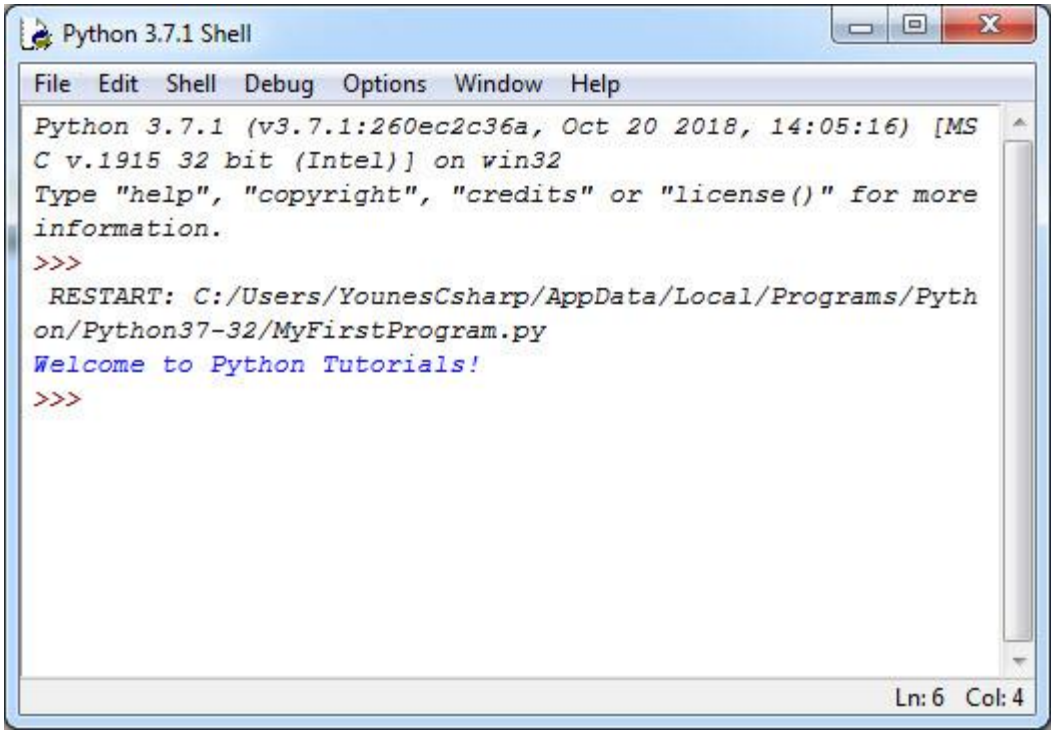
سپس یک مسیر برای ذخیره فایل انتخاب کنید. ما در شکل زیر فایل را در درایو C ذخیره کرده‌ایم:



بعد از ذخیره فایل به محیط کدنویسی بر گشته و از منوی Run گزینه Run Module و یا دکمه F5 را بزنید:



مشاهده می‌کنید که برنامه اجرا شده و پیغام Welcome to Python tutorials! چاپ می‌شود:



مثال بالا ساده‌ترین برنامه‌ای است که شما می‌توانید در Python بنویسید. هدف در مثال بالا نمایش یک پیغام در صفحه نمایش است. هر زبان برنامه‌نویسی دارای قواعدی برای کدنویسی است. پایتون دارای توابع از پیش تعریف شده‌ای است که هر کدام برای مقاصد خاصی به کار می‌روند. هر چند که در آینده در مورد توابع بیشتر توضیح می‌دهیم ولی در همین حد به توضیح تابع بسنده می‌کنیم که توابع مجموعه‌ای از کدها هستند که دارای یک نام بوده و در جلوی نام آنها علامت () قرار می‌گیرد. یکی از این توابع، تابع print() است. از تابع print() برای چاپ یک رشته استفاده می‌شود. یک رشته گروهی از کاراکترها است، که به وسیله دابل کوتیشن (") محصور شده است. مانند: "Welcome to Python Tutorials!". یک کاراکتر می‌تواند یک حرف، عدد، علامت یا ... باشد. در کل مثال بالا نحوه استفاده از تابع print() است. توضیحات بیشتر در درس‌های آینده آمده است. پایتون فضای خالی بالا را نادیده می‌گیرد و از کد زیر اشکال نمی‌گیرد:

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

همیشه به یاد داشته باشید که Python به بزرگی و کوچکی حروف حساس است. یعنی به طور مثال MAN و man در Python با هم فرق دارند. رشته‌ها و توضیحات از این قاعده مستثنی هستند که در درس‌های آینده توضیح خواهیم داد. مثلاً کدهای زیر با خطا مواجه می‌شوند و اجرا نمی‌شوند:

```
Print("Welcome to Python Tutorials!")
PRINT("Welcome to Python Tutorials!")
Print("Welcome to Python Tutorials!")
```

تغییر در بزرگی و کوچکی حروف از اجرای کدها جلوگیری می‌کند. اما کد زیر کاملاً بدون خطا است:

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

نکاتی در مورد کدنویسی در پایتون

در زبان‌هایی مثل جاوا و سی شارپ، از علامت آکولاد ({}) برای ایجاد یک بلاک کد

```
Block
{
    statement
}
```

ولی در زبان پایتون از ترکیب علامت دو نقطه (:) و تو رفتگی برای اینکار استفاده می‌شود:

```
Block:
    statement
```

ما با دونقطه به پایتون می‌گوییم که قصد داریم یک بلوک کد را آغاز کنیم و با تو رفتگی ابتدای خطوط دستورات آن بلوک را تعریف می‌کنیم. برای تورفتگی می‌توانیم از ۳ یا ۴ یا ۱۰ فضای خالی استفاده کنیم. میزان این فضای خالی تا زمانی که در تمام کد رعایت شود، اهمیتی ندارد. در کد زیر به اهمیت تو رفتگی‌ها پی می‌برید:

```
Block1:
    statement
    statement
Block2:
    statement
    Block3:
        statement
        statement
        Block4:
            statement
    statement
statement
```

در کد بالا، بلاک اول (Block1) از خط ۱ تا ۳ را شامل می‌شود. به این نکته توجه کنید که خطوط بعد از علامت: حتماً باید دارای تو رفتگی باشند. بلاک دوم (Block2) از خط ۴ شروع و به خط ۱۱ ختم می‌شود. نکته‌ای که باید در اینجا دوباره به آن اشاره کنیم این است که دستور یا بلاک‌هایی که دارای فاصله‌های برابر از سمت چپ هست جزو یک بلاک می‌باشند. مثلاً در کد بالا خطوط ۲ و ۳ جز Block1 هستند چون تو رفتگی آنها از سمت چپ برابر است و اگر مثلاً فاصله‌های خط ۳ از سمت چپ را حذف کنیم دیگر جز بلاک محسوب نمی شود. یک بلاک را می‌توان زیر مجموعه بلاک دیگر کرد. مثلاً در خط ۶، Block3 را زیر مجموعه Block2 و در خط ۹، Block4 را زیر مجموعه Block3 کرده‌ایم. در نهایت خط ۱۲ جز هیچکدام از بلاک‌ها نیست و مستقل اجرا می‌شود. گاهی اوقات و هنگام کدنویسی، لازم است که رشته‌های طولانی را در چند خط بنویسید. برای اینکار در پایتون می‌توان از علامت \ به صورت زیر استفاده کنید:

```
print("Welcome \
to \
Python \
Tutorials!")
```

توضیحات

وقتی که کدی تایپ می‌کنید شاید بخواهید که متنی جهت یادآوری وظیفه آن کد به آن اضافه کنید. در Python (و بیشتر زبانهای برنامه‌نویسی) می‌توان این کار را با استفاده از توضیحات انجام داد. توضیحات متونی هستند که توسط مفسر نادیده گرفته می‌شوند و به عنوان بخشی از کد محسوب نمی‌شوند.

هدف اصلی از ایجاد توضیحات، بالا بردن خوانایی و تشخیص نقش کدهای نوشته شده توسط شما، برای دیگران است. فرض کنید که می‌خواهید در مورد یک کد خاص، توضیح بدهید، می‌توانید توضیحات را در بالای کد یا کنار آن بنویسید. از توضیحات برای مستند سازی برنامه هم استفاده می‌شود. در برنامه زیر نقش توضیحات نشان داده شده است:

```
#This line will print the message hello world
print("Hello World!")
```

در کد بالا، خط اول کد بالا یک توضیح درباره خط دوم است که به کاربر اعلام می‌کند که وظیفه خط دوم چیست؟ با اجرای کد بالا فقط جمله Hello World چاپ شده و خط اول در خروجی نمایش داده نمی‌شود چون مفسر توضیحات را نادیده می‌گیرد. همانطور که مشاهده

می‌کنید برای درج توضیحات در پایتون از علامت # استفاده می‌شود. برای توضیحات طولانی هم باید در ابتدای هر خط از توضیح این علامت درج شود:

```
#This line will print
#the message hello world
print("Hello World!")
```

کاراکترهای کنترلی

کاراکترهای کنترلی، کاراکترهای ترکیبی هستند که با یک بک اسلش (\) شروع می‌شوند و به دنبال آنها یک حرف یا عدد می‌آید و یک رشته را با فرمت خاص نمایش می‌دهند. برای مثال برای ایجاد یک خط جدید و قرار دادن رشته در آن می‌توان از کاراکتر کنترلی \n استفاده کرد:

```
print("Hello\nWorld!")
```

```
Hello
World
```

مشاهده کردید که مفسر بعد از مواجهه با کاراکتر کنترلی \n نشانگر ماوس را به خط بعد برده و بقیه رشته را در خط بعد نمایش می‌دهد.

جدول زیر لیست کاراکترهای کنترلی و کارکرد آنها را نشان می‌دهد:

عملکرد	کاراکتر کنترلی	عملکرد	کاراکتر کنترلی
Form Feed	\f	چاپ کوتیشن	\’
خط جدید	\n	چاپ دابل کوتیشن	\”
سر سطر رفتن	\r	چاپ بک اسلش	\\
حرکت به صورت افقی	\t	چاپ فضای خالی	\0
حرکت به صورت عمودی	\v	صدای بیپ	\a
چاپ کاراکتر یونیکد	\u	حرکت به عقب	\b

ما برای استفاده از کاراکترهای کنترلی، از بک اسلش (\) استفاده می‌کنیم. از آنجاییکه \ معنای خاصی به رشته‌ها می‌دهد برای چاپ بک اسلش (\) باید از (\\) استفاده کنیم:

```
print("We can print a \\ by using the \\\\ escape sequence.")
```

```
We can print a \ by using the \\ escape sequence.
```

یکی از موارد استفاده از \\، نشان دادن مسیر یک فایل در ویندوز است:

```
print("C:\\Program Files\\Some Directory\\SomeFile.txt")
```

```
C:\Program Files\Some Directory\SomeFile.txt
```

از آنجاییکه از دابل کوتیشن (“) برای نشان دادن رشته‌ها استفاده می‌کنیم برای چاپ آن از ”\ استفاده می‌کنیم:

```
print("I said, \"Motivate yourself!\".")
```

```
I said, "Motivate yourself!".
```

همچنین برای چاپ کوتیشن (‘) از ’\ استفاده می‌کنیم:

```
print("The programmer\'s heaven.")
```

```
The programmer's heaven.
```

برای ایجاد فاصله بین حروف یا کلمات از t\ استفاده می‌شود:

```
print("Left\tRight")
```

```
Left  Right
```

برای چاپ کاراکترهای یونیکد می‌توان از u\ استفاده کرد. برای استفاده از u\، مقدار در مبنای ۱۶ کاراکتر را درست بعد از علامت u\ قرار می‌دهیم. برای مثال اگر بخواهیم علامت کپی رایت (©) را چاپ کنیم، باید بعد از علامت u\ مقدار 00A9 را قرار دهیم مانند:

```
print("\u00A9")
```

```
©
```

برای مشاهده لیست مقادیر مبنای ۱۶ برای کاراکترهای یونیکد به لینک زیر مراجعه نمایید:

```
http://www.ascii.cl/htmlcodes.htm
```

اگر مفسر به یک کاراکتر کنترلی غیر مجاز برخورد کند، برنامه پیغام خطا می‌دهد. بیشترین خطا زمانی اتفاق می افتد که برنامه نویس برای چاپ اسلش (\) از \\ استفاده می‌کند.

متغیر

متغیر مکانی از حافظه است که شما می‌توانید مقادیری را در آن ذخیره کنید. می‌توان آن را به عنوان یک ظرف تصور کرد که داده‌های خود را در آن قرار داده‌اید. محتویات این ظرف می‌تواند پاک شود یا تغییر کند. هر متغیر دارای یک نام نیز هست. که از طریق آن می‌توان متغیر را از دیگر متغیرها تشخیص داد و به مقدار آن دسترسی پیدا کرد. همچنین دارای یک مقدار می‌باشد که می‌تواند توسط کاربر انتخاب شده باشد یا نتیجه یک محاسبه باشد. مقدار متغیر می‌تواند تهی نیز باشد. متغیر دارای نوع نیز هست بدین معنی که نوع آن با نوع داده‌ای که در آن ذخیره می‌شود یکی است.

متغیر دارای عمر نیز هست که از روی آن می‌توان تشخیص داد که متغیر باید چقدر در طول برنامه مورد استفاده قرار گیرد. و در نهایت متغیر دارای محدوده استفاده نیز هست که به شما می‌گوید که متغیر در چه جای برنامه برای شما قابل دسترسی است. ما از متغیرها به عنوان یک انبار موقتی برای ذخیره داده استفاده می‌کنیم. هنگامی که یک برنامه ایجاد می‌کنیم احتیاج به یک مکان برای ذخیره داده، مقادیر یا داده‌هایی که توسط کاربر وارد می‌شوند، داریم. این مکان، همان متغیر است.

برای این از کلمه متغیر استفاده می‌شود چون ما می‌توانیم بسته به نوع شرایط هر جا که لازم باشد، مقدار آن را تغییر دهیم. متغیرها موقتی هستند و فقط موقعی مورد استفاده قرار می‌گیرند که برنامه در حال اجراست و وقتی شما برنامه را می‌بندید محتویات متغیرها نیز پاک می‌شود. قبلاً ذکر شد که به وسیله نام متغیر می‌توان به آن دسترسی پیدا کرد. برای نامگذاری متغیرها باید قوانین زیر را رعایت کرد:

- نام متغیر باید با یکی از حروف الفبا (a-z or A-Z) یا علامت _ شروع شود.
- نمی‌تواند شامل کاراکترهای غیرمجاز مانند $, ^, ?, #. باشد.
- نمی‌توان از کلمات رزرو شده در پایتون برای نام متغیر استفاده کرد.
- نام متغیر نباید دارای فضای خالی (spaces) باشد.

اسامی متغیرها نسبت به بزرگی و کوچکی حروف حساس هستند. در پایتون دو حرف مانند a و A دو کاراکتر مختلف به حساب می‌آیند.

دو متغیر با نامهای myNumber و MyNumber دو متغیر مختلف محسوب می‌شوند چون یکی از آنها با حرف کوچک m و دیگری با حرف بزرگ M شروع می‌شود. متغیر دارای نوع هست که نوع داده‌ای را که در خود ذخیره می‌کند را نشان می‌دهد. در درس بعد در مورد انواع داده‌ها در پایتون توضیح می‌دهیم. لیست کلمات کلیدی پایتون، که نباید از آنها در نامگذاری متغیرها استفاده کرد در زیر آمده است:

```
False               def                 if                  raise
None                del                 import              return
True                elif                in                  try
and                 else                is                  while
as                  except              lambda              with
assert              finally             nonlocal            yield
break               for                 not
class               from                or
continue            global              pass
```

انواع داده

انواع داده‌هایی که در پایتون وجود دارند عبارتند از:

نوع		توضیح
عددی (Numeric)	integer	شامل اعداد مثبت و منفی صحیح می‌باشد.
	float	شامل اعداد اعشاری می‌باشد.
	complex	شامل اعداد مختلط می‌باشد. نوع داده مختلط نوع غیر قابل تغییری است که یک جفت float را نگهداری می‌کند که یک بخش آن نشان دهنده قسمت حقیقی و یک بخش آن نشان دهنده قسمت موهومی عدد مختلط است. مانند (7.1j + 3)
رشته‌ای (String)		به مجموعه‌ای از کاراکترها که بین دو علامت کوتیشن یا دابل کوتیشن قرار گرفته باشند، اطلاق می‌شود.
لیست (List)		مجموعه‌ای از آیتم‌ها هستند که بین دو علامت [] قرار گرفته و با علامت کاما (,) از هم جدا شده‌اند.
تاپل (Tuple)		مجموعه‌ای از آیتم‌ها هستند که بین دو علامت () قرار گرفته و با علامت کاما (,) از هم جدا شده‌اند.
دیکشنری (Dictionary)		مجموعه‌ای از آیتم‌ها هستند که به صورت کلید و مقدار بوده، بین دو علامت {} قرار گرفته، و با علامت کاما (,) از هم جدا شده‌اند.
boolean		شامل دو مقدار true یا false می‌باشد.

در مورد انواع داده‌های بالا و نحوه استفاده از آنها در متغیرها، در درس بعد توضیح می‌دهیم.

استفاده از متغیرها

بر خلاف زبان‌هایی مثل جاوا و سی شارپ، که هنگام تعریف متغیر باید نوع متغیر را هم مشخص می‌کردیم، در پایتون کافیست که فقط نام متغیر را نوشته و به وسیله علامت مساوی یک مقدار به آن اختصاص دهیم:

```
variableName = Value
```

در مثال زیر نحوه تعریف و مقداردهی متغیرها نمایش داده شده است:

```
1  intVar        = 10
2  floatVar      = 12.5
3  boolVar       = True
4  StringVar     = "Hello World!"
5  listVar       = [1,5,8]
6  tupleVar      = ("Python","Programming","begginer")
7  dictionaryVar = {'Name': 'jack', 'family': 'Scalia', 'Age': 7}
8
9  print("intVar        = {0}".format(intVar))
10 print("floatVar      = {0}".format(floatVar))
11 print("boolVar       = {0}".format(boolVar))
12 print("StringVar     = {0}".format(StringVar))
13 print("listVar       = {0}".format(listVar))
14 print("tupleVar      = {0}".format(tupleVar))
15 print("dictionaryVar = {0}".format(dictionaryVar))
```

```
intVar        = 10
floatVar      = 12.5
boolVar       = True
StringVar     = Hello World!
listVar       = [1, 5, 8]
tupleVar      = ('Python', 'Programming', 'begginer')
dictionaryVar = {'Name': 'jack', 'family': 'Scalia', 'Age': 7}
```

در خطوط ۷-۱، متغیرها تعریف شده‌اند. اما نوع این متغیرها چیست؟ پایتون نوع متغیرها را بسته به مقداری که به آنها اختصاص داده می‌شود در نظر می‌گیرد. مثلاً نوع متغیر StringVar در خط ۴ از نوع رشته است، چون یک مقدار رشته‌ای به آن اختصاص داده شده است. به خطوط ۵، ۶ و ۷ کد بالا توجه کنید. در خط ۵ یک متغیر تعریف شده است و نوع داده‌ای که به آن اختصاص داده شده است از نوع list است. همانطور که در درس قبل اشاره شد، برای تعریف list علامت [] به کار می‌رود و آیتم‌های داخل آن به وسیله کاما از هم جدا می‌شوند:

```
listVar = [1, 5, 8]
```

در خط ۶ هم یک متغیر تعریف شده است و یک مقدار از نوع tuple به آن اختصاص داده شده است. در تعریف tuple به جای علامت [] از () استفاده می‌شود. تفاوت بین این دو را در درس‌های آینده بیشتر توضیح می‌دهیم. و اما در خط ۷ یک نوع دیکشنری تعریف شده است. برای تعریف دیکشنری بین کلید و مقدار علامت: و بین کلید/مقدارها هم علامت, قرار می‌گیرد:

```
dictionaryVar = {Key1:Value1, Key2:Value2, Key3:Value3}
```

مثلاً در مثال بالا یک دیکشنری تعریف کرده‌ایم که سه آیتم یا کلید/مقدار دارد که بین آنها علامت کاما (,) قرار داده‌ایم. ولی بین یک کلید و مقدار مربوط به آن علامت: قرار گرفته است. برای اختصاص یک مقدار به چند متغیر می‌توان به صورت زیر عمل کرد:

```
identifier1 =  identifier2 = ... indentifierN = Value
```

به مثال زیر توجه کنید:

```
num1 = num2 = num3 = num4 = num5 = 10
message1 = message2 = message3 = "Hello World!"

print(num1)
print(num4)
print(message1)
print(message3)
```

```
10
10
Hello World!
Hello World!
```

دقت کنید که برای متغیرهای تعریف شده در حالت بالا یک خانه حافظه تخصیص داده می‌شود، یعنی مقدار ۱۰ در حافظه ذخیره شده و متغیرهای num1 و num2 و num3 و num4 و num5 به آن خانه از حافظه اشاره می‌کنند. همچنین می‌توان چند متغیر را تعریف کرد و برای هر یک از آن‌ها مقدار جداگانه‌ای مشخص نمود:

```
identifier1, identifier2, ... indentifierN = Value1, Value2, ... ValueN
```

به مثال زیر توجه کنید:

```
num1, num2, message1 = 10, 12.5, "Hello World!"

print(num1)
print(num2)
print(message1)
```

```
10
12.5
Hello World!
```

همانطور که در درس قبل هم اشاره کردیم، یک رشته در اصل یک مجموعه از کاراکترهاست که در داخل علامت "" یا ' ' قرار دارند. هر کدام از این کاراکترها دارای یک اندیس است که به وسیله آن اندیس قابل دسترسی هستند. اندیس کاراکترها در رشته از ۰ شروع می‌شود. به رشته زیر توجه کنید:

```
message = "Hello World!"
```

در رشته بالا اندیس کاراکتر O برابر ۴ است. برای درک بهتر به شکل زیر توجه کنید:

```
H e l l o  W o r l d !
0 1 2 3 4 5 6 7 8 9 10 11
```

حال برای چاپ یک کاراکتر (مثل W) از این رشته کافیست که به صورت زیر عمل کنیم:

```
message = "Hello World! "
print(message[6])
```

```
W
```

همانطور که در کد بالا مشاهده می‌کنید کافیست که نام متغیر را نوشته، در جلوی آن یک جفت کروشه و در داخل کروشه‌ها اندیس آن کاراکتری را که می‌خواهیم چاپ شود را بنویسیم. چاپ مقدار با استفاده از اندیس در مورد List و Tuple هم صدق می‌کند:

```
listVar  = [1, 5, 8]
tupleVar = ("Python", "Programming", "begginer")

print(listVar[2])
print(tupleVar[1])
```

```
8
Programming
```

و اما در مورد دیکشنری، شما باید نام کلید را بنویسید تا مقدار آن برای شما نمایش داده شود:

```
dictionaryVar = {'Name': 'jack', 'Family': 'Scalia', 'Age': 7}

print(dictionaryVar['Family'])
```

```
Scalia
```

نکته‌ای که بهتر است در همینجا به آن اشاره کنیم این است که کلید/مقدارها در دیکشنری می‌توانند از هر نوعی باشند و شما برای چاپ مقدار مربوط به یک کلید باید نام کلید را دقیق بنویسید. به مثال زیر توجه کنید:

```
dictionaryVar   = {1:'Jack', '2':'Scalia', 3:7}

print(dictionaryVar['2'])
```

```
Scalia
```

در مثال بالا ما مقدار کلید '۲' را چاپ کرده‌ایم. حال اگر به جای '۲' عدد ۲ را بنویسیم، یعنی علامت کوتیشن را نگذاریم با خطا مواجه می شویم:

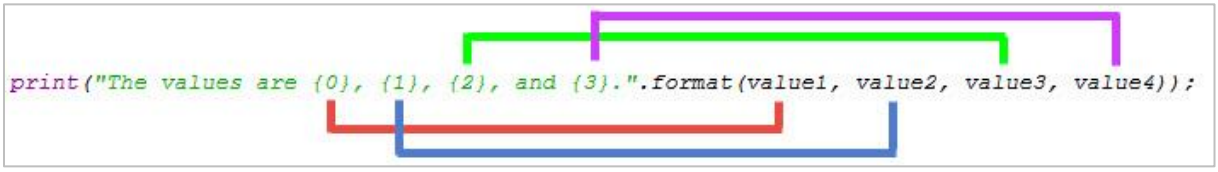
```
dictionaryVar = {1:'Jack', '2':'Scalia', 3:7}

print(dictionaryVar[2])
```

```
Traceback (most recent call last):
  File "C:/MyFirstProgram.py", line 3, in
    print(dictionaryVar[2])
KeyError: 2
```

جانگهدار (Placeholders)

به تابع ()print در خطوط (۱۵-۹) توجه کنید. این تابع به دو قسمت تقسیم شده است. قسمت اول یک رشته قالب بندی شده است و و قسمت دوم هم شامل متدی به نام ()format است که دارای مقدار یا مقادیری است که توسط رشته قالب بندی شده مورد استفاده قرار می‌گیرند. اگر به دقت نگاه کنید رشته قالب بندی شده دارای عدد صفری است که در داخل دو آکولاد محصور شده است. البته عدد داخل دو آکولاد می‌تواند از صفر تا n باشد. به این اعداد جانگهدار می گویند. این اعداد بوسیله مقدار یا مقادیری که در داخل تابع ()format هستند جایگزین می‌شوند. به عنوان مثال جانگهدار {0} به این معناست که اولین مقدار داخل تابع ()format در آن قرار می‌گیرد. برای روشن شدن مطلب به شکل زیر توجه کنید:

```
print("The values are {0}, {1}, {2}, and {3}.".format(value1, value2, value3, value4))
```



جانگهدارها از صفر شروع می‌شوند. تعداد جانگهدارها باید با تعداد مقادیری که در داخل تابع ()format آورده شده‌اند برابر باشد. برای مثال اگر شما چهار جانگهدار مثل بالا داشته باشید باید چهار مقدار هم برای آنها بعد از رشته قالب بندی شده در نظر بگیرید. اولین جانگهدار با اولین مقدار و دومین جا نگهدار با دومین مقدار جایگزین می‌شود. در ابتدا فهمیدن این مفهوم برای کسانی که تازه برنامه‌نویسی را شروع کرده‌اند سخت است اما در درسهای آینده مثالهای زیادی در این مورد مشاهده خواهید کرد.

عبارات و عملگرها

ابتدا با دو کلمه آشنا شوید:

- عملگر: نمادهایی هستند که اعمال خاص انجام می‌دهند.
- عملوند: مقادیری که عملگرها بر روی آنها عملی انجام می‌دهند.

مثلاً X+Y: یک عبارت است که در آن X و Y عملوند و علامت + عملگر به حساب می‌آیند.

زبانهای برنامه‌نویسی جدید دارای عملگرهایی هستند که از اجزاء معمول زبان به حساب می‌آیند. پایتون دارای عملگرهای مختلفی از جمله عملگرهای ریاضی، تخصیصی، مقایسه‌ای، منطقی و بیتی می‌باشد. از عملگرهای ساده ریاضی می‌توان به عملگر جمع و تفریق اشاره کرد.

سه نوع عملگر در پایتون وجود دارد:

یگانی – به یک عملوند نیاز دارد

دودویی – به دو عملوند نیاز دارد

سه تایی – به سه عملوند نیاز دارد

انواع مختلف عملگر که در این بخش مورد بحث قرار می‌گیرند، عبارتند از:

- عملگرهای ریاضی
- عملگرهای تخصیصی
- عملگرهای مقایسه‌ای
- عملگرهای منطقی
- عملگرهای بیتی

عملگرهای ریاضی

پایتون از عملگرهای ریاضی برای انجام محاسبات استفاده می‌کند. جدول زیر عملگرهای ریاضی پایتون را نشان می‌دهد:

عملگر	مثال	نتیجه
+	var1 = var2 + var3	Var1 برابر است با حاصل جمع var2 و var3
-	var1 = var2 - var3	Var1 برابر است با حاصل تفریق var2 و var3
*	var1 = var2 * var3	Var1 برابر است با حاصلضرب var2 در var3
/	var1 = var2 / var3	Var1 برابر است با حاصل تقسیم var2 بر var3
%	var1 = var2 % var3	Var1 برابر است با باقیمانده تقسیم var2 و var3
**	var1 = var2 ** var3	Var1 برابر است با مقدار var2 به توان var3
//	var1 = var2 // var3	Var1 برابر است با var2 تقسیم بر var3 (نتیجه به صورت صحیح نمایش داده می‌شود).

مثال بالا در از نوع عددی استفاده شده است. اما استفاده از عملگرهای ریاضی برای نوع رشته‌ای نتیجه متفاوتی دارد. اگر از عملگر + برای رشته‌ها استفاده کنیم دو رشته را با هم ترکیب کرده و به هم می‌چسباند. حال می‌توانیم با ایجاد یک برنامه نحوه عملکرد عملگرهای ریاضی در پایتون را یاد بگیریم:

```
#Assign test values
num1 = 5
num2 = 3

#Demonstrate use of mathematical operators
print("The sum of {0} and {1} is {2}."              .format(num1, num2, (num1 +  num2)))
print("The difference of {0} and {1} is {2}."       .format(num1, num2, (num1 -  num2)))
print("The product of {0} and {1} is {2}."          .format(num1, num2, (num1 *  num2)))
```

```
print("The quotient of {0} and {1} is {2:.2f}."     .format(num1, num2, (num1 /  num2)))
print("The remainder of {0} divided by {1} is {2}." .format(num1, num2, (num1 %  num2)))
print("The result of {0} power {1} is {2}."         .format(num1, num2, (num1 ** num2)))
print("The quotient of {0} and {1} is {2}."         .format(num1, num2, (num1 // num2)))

#Demonstrate concatenation on strings using the + operator
msg1 = "Hello "
msg2 = "World!"
print(msg1 + msg2)
```

```
The sum of 5 and 3 is 8.
The difference of 5 and 3 is 2.
The product of 5 and 3 is 15.
The quotient of 5 and 3 is 1.67.
The remainder of 5 divided by 3 is 2.
The result of 5 power 3 is 125.
The quotient of 5 and 3 is 1.
Hello World!
```

برنامه بالا نتیجه هر عبارت را نشان می‌دهد. در این برنامه از تابع `print()` برای نشان دادن نتایج در سطرهای متفاوت استفاده شده است. در خط ۹ برای اینکه ارقام کسری بعد از عدد حاصل دو رقم باشند از `{2:.2f}` استفاده می‌کنیم. `{2:.2f}` در این جا بدین معناست که عدد را تا دو رقم اعشار نمایش بده. پایتون خط جدید و فاصله و فضای خالی را نادیده می‌گیرد. در خط ۱۷ مشاهده می‌کنید که دو رشته به وسیله عملگر + به هم متصل شده‌اند. نتیجه استفاده از عملگر + برای چسباندن دو کلمه "`Hello`" و "`World!`" رشته "`Hello World!`" خواهد بود. به فاصله‌های خالی بعد از اولین کلمه توجه کنید اگر آنها را حذف کنید از خروجی برنامه نیز حذف می‌شوند.

عملگرهای تخصیصی (جایگزینی)

نوع دیگر از عملگرهای پایتون عملگرهای جایگزینی نام دارند. این عملگرها مقدار متغیر سمت راست خود را در متغیر سمت چپ قرار می‌دهند. جدول زیر انواع عملگرهای تخصیصی در پایتون را نشان می‌دهد:

عملگر	مثال	نتیجه
=	var1 = var2	مقدار var1 برابر است با مقدار var2
+=	var1 += var2	مقدار var1 برابر است با حاصل جمع var1 و var2
-=	var1 -= var2	مقدار var1 برابر است با حاصل تفریق var1 و var2
*=	var1 *= var2	مقدار var1 برابر است با حاصل ضرب var1 در var2
/=	var1 /= var2	مقدار var1 برابر است با حاصل تقسیم var1 بر var2

%=	var1 %= var2	مقدار var1 برابر است با باقیمانده تقسیم var1 بر var2
**=	var1 **= var2	مقدار var1 برابر است با var1 به توان var2
//=	var1 //= var2	مقدار var1 برابر است با حاصل تقسیم var1 بر var2

از عملگر =+ برای اتصال دو رشته نیز می‌توان استفاده کرد. استفاده از این نوع عملگرها در واقع یک نوع خلاصه نویسی در کد است. مثلاً شکل اصلی کد var1 += var2 به صورت var1 = var1 + var2 می‌باشد. این حالت کدنویسی زمانی کارایی خود را نشان می‌دهد که نام متغیرها طولانی باشد. برنامه زیر چگونگی استفاده از عملگرهای تخصیصی و تأثیر آنها را بر متغیرها نشان می‌دهد:

```
 print("Assigning 10 to number...")
number = 10
print("Number = {0}" .format(number))

print("Adding 10 to number...")
number += 10
print("Number = {0}" .format(number))

print("Subtracting 10 from number...")
number -= 10
print("Number = {0}" .format(number))
```

```
Assigning 10 to number...
Number = 10
Adding 10 to number...
Number = 20
Subtracting 10 from number...
Number = 10
```

در برنامه از ۳ عملگر تخصیصی استفاده شده است. ابتدا یک متغیر و مقدار 10 با استفاده از عملگر = به آن اختصاص داده شده است. سپس به آن با استفاده از عملگر =+ مقدار ۱۰ اضافه شده است. و در آخر به وسیله عملگر =- عدد ۱۰ از آن کم شده است.

عملگرهای مقایسه‌ای

از عملگرهای مقایسه‌ای برای مقایسه مقادیر استفاده می‌شود. نتیجه این مقادیر یک مقدار بولی (منطقی) است. این عملگرها اگر نتیجه مقایسه دو مقدار درست باشد مقدار ۱ و اگر نتیجه مقایسه اشتباه باشد مقدار ۰ را نشان می‌دهند. این عملگرها به طور معمول در دستورات شرطی به کار می‌روند به این ترتیب که باعث ادامه یا توقف دستور شرطی می‌شوند. جدول زیر عملگرهای مقایسه‌ای در پایتون را نشان می‌دهد:

عملگر	مثال	نتیجه
==	var1 = var2 == var3	var1 در صورتی True است که مقدار var2 با مقدار var3 برابر باشد در غیر اینصورت False است.
!=	var1 = var2 != var3	var1 در صورتی True است که مقدار var2 با مقدار var3 برابر نباشد در غیر اینصورت False است.
<>	var1 = var2 <> var3	var1 در صورتی True است که مقدار var2 با مقدار var3 برابر نباشد در غیر اینصورت False است.
<	var1 = var2 < var3	var1 در صورتی True است که مقدار var2 کوچک‌تر از var3 مقدار باشد در غیر اینصورت False است.
>	var1 = var2 > var3	var1 در صورتی True است که مقدار var2 بزرگ‌تر ازمقدار var3 باشد در غیر اینصورت False است.
<=	var1 = var2 <= var3	var1 در صورتی True است که مقدار var2 کوچک‌تر یا مساوی مقدار var3 باشد در غیر اینصورت False است.
>=	var1 = var2 >= var3	var1 در صورتی True است که مقدار var2 بزرگ‌تر یا مساوی var3 مقدار باشد در غیر اینصورت False است.

برنامه زیر نحوه عملکرد این عملگرها را نشان می‌دهد:

```
num1 = 10
num2 = 5

print("{0} == {1} : {2}" .format(num1, num2, num1 == num2))
print("{0} != {1} : {2}" .format(num1, num2, num1 != num2))
print("{0} <> {1} : {2}" .format(num1, num2, num1 != num2))
print("{0} <  {1} : {2}" .format(num1, num2, num1 <  num2))
print("{0} >  {1} : {2}" .format(num1, num2, num1 >  num2))
print("{0} <= {1} : {2}" .format(num1, num2, num1 <= num2))
print("{0} >= {1} : {2}" .format(num1, num2, num1 >= num2))
```

```
10 == 5 : False
10 != 5 : True
10 <> 5 : True
10 <  5 : False
```

```
10 >  5 : True
10 <= 5 : False
10 >= 5 : True
```

در مثال بالا ابتدا دو متغیر را که می‌خواهیم با هم مقایسه کنیم را ایجاد کرده و به آنها مقادیری اختصاص می‌دهیم. سپس با استفاده از یک عملگر مقایسه‌ای آنها را با هم مقایسه کرده و نتیجه را چاپ می‌کنیم. به این نکته توجه کنید که هنگام مقایسه دو متغیر از عملگر == به جای عملگر = باید استفاده شود. عملگر = عملگر تخصیصی است و در عبارتی مانند x = y مقدار y را در به x اختصاص می‌دهد. عملگر == عملگر مقایسه‌ای است که دو مقدار را با هم مقایسه می‌کند مانند x==y و اینطور خوانده می‌شود x برابر است با y.

عملگرهای منطقی

عملگرهای منطقی بر روی عبارات منطقی عمل می‌کنند و نتیجه آنها نیز یک مقدار بولی است. از این عملگرها اغلب برای شرطهای پیچیده استفاده می‌شود. همانطور که قبلاً یاد گرفتید مقادیر بولی می‌توانند `False` یا `True` باشند. فرض کنید که `var2` و `var3` دو مقدار بولی هستند.

عملگر	مثال
and	`var1 = var2 and var3`
or	`var1 = var2 or var3`
not	`var1 = not (var1)`

عملگر منطقی and

اگر مقادیر دو طرف عملگر and، `True` باشند عملگر and مقدار `True` را بر می‌گرداند. در غیر اینصورت اگر یکی از مقادیر یا هر دوی آنها `False` باشند مقدار `False` را بر می‌گرداند. در زیر جدول درستی عملگر and نشان داده شده است:

X and Y	Y	X
True	True	True
False	False	True
False	True	False
False	False	False

برای درک بهتر تأثیر عملگر and یاد آوری می‌کنم که این عملگر فقط در صورتی مقدار True را نشان می‌دهد که هر دو عملوند مقدارشان True باشد. در غیر اینصورت نتیجه تمام ترکیبهای بعدی False خواهد شد. استفاده از عملگر and مانند استفاده از عملگرهای مقایسه‌ای است. به عنوان مثال نتیجه عبارت زیر درست (True) است اگر سن (age) بزرگ‌تر از ۱۸ و salary کوچک‌تر از ۱۰۰۰ باشد.

```
result = (age > 18) and (salary < 1000)
```

عملگر and زمانی کارامد است که ما با محدود خاصی از اعداد سرو کار داریم. مثلاً عبارت 10 <= x <= 100 بدین معنی است که x می تواند مقداری شامل اعداد ۱۰ تا ۱۰۰ را بگیرد. حال برای انتخاب اعداد خارج از این محدوده می‌توان از عملگر منطقی and به صورت زیر استفاده کرد.

```
inRange = (number <= 10) and (number >= 100)
```

عملگر منطقی or

اگر یکی یا هر دو مقدار دو طرف عملگر or، درست (True) باشد، عملگر or مقدار True را بر می‌گرداند. جدول درستی عملگر or در زیر نشان داده شده است:

X	Y	X or Y
True	True	True
True	False	True
False	True	True
False	False	False

در جدول بالا مشاهده می‌کنید که عملگر or در صورتی مقدار False را بر می‌گرداند که مقادیر دو طرف آن False باشند. کد زیر را در نظر بگیرید. نتیجه این کد در صورتی درست (True) است که رتبه نهایی دانش آموز (finalGrade) بزرگ‌تر از ۷۵ یا یا نمره نهایی امتحان آن ۱۰۰ باشد.

```
isPassed = (finalGrade >= 75) or (finalExam == 100)
```

عملگر منطقی not

برخلاف دو اپراتور or و and عملگر منطقی NOT یک عملگر یگانی است و فقط به یک عملوند نیاز دارد. این عملگر یک مقدار یا اصطلاح بولی را نفی می‌کند. مثلاً اگر عبارت یا مقدار True باشد آنرا False و اگر False باشد آنرا True می‌کند. جدول زیر عملکرد اپراتور NOT را نشان می‌دهد:

not X	X
False	True
True	False

نتیجه کد زیر در صورتی درست است که age (سن) بزرگ‌تر یا مساوی ۱۸ نباشد.

```
isMinor = not(age >= 18)
```

عملگرهای بیتی

عملگرهای بیتی به شما اجازه می‌دهند که شکل باینری انواع داده‌ها را دستکاری کنید. برای درک بهتر این درس توصیه می‌شود که شما سیستم باینری و نحوه تبدیل اعداد دهدهی به باینری را از لینک زیر یاد بگیرید:

```
http://www.w3-farsi.com/?p=5698
```

در سیستم باینری (دودویی) که کامپیوتر از آن استفاده می‌کند وضعیت هر چیز یا خاموش است یا روشن. برای نشان دادن حالت روشن از عدد ۱ و برای نشان دادن حالت خاموش از عدد ۰ استفاده می‌شود. بنابراین اعداد باینری فقط می‌توانند صفر یا یک باشند. اعداد باینری را اعداد در مبنای ۲ و اعداد اعشاری را اعداد در مبنای ۱۰ می گویند. یک بیت نشان دهنده یک رقم باینری است و هر بایت نشان دهنده ۸ بیت است. به عنوان مثال برای یک داده از نوع int به ۳۲ بیت یا ۴ بایت فضا برای ذخیره آن نیاز داریم، این بدین معناست که اعداد از ۳۲ رقم ۰ و ۱ برای ذخیره استفاده می‌کنند. برای مثال عدد ۱۰۰ وقتی به عنوان یک متغیر از نوع int ذخیره می‌شود در کامپیوتر به صورت زیر خوانده می‌شود:

```
00000000000000000000000001100100
```

عدد ۱۰۰ در مبنای ده معادل عدد ۱۱۰۰۱۰۰ در مبنای ۲ است. در اینجا ۷ رقم سمت راست نشان دهنده عدد ۱۰۰ در مبنای ۲ است و مابقی صفرهای سمت راست برای پر کردن بیتهایی است که عدد از نوع int نیاز دارد. به این نکته توجه کنید که اعداد باینری از سمت راست به چپ خوانده می‌شوند. عملگرهای بیتی پایتون در جدول زیر نشان داده شده‌اند:

عملگر	مثال
&	x = y & z
|	x = y | z
^	x = y ^ z

~	x = ~y
&=	x &= y
|=	x |= y
^=	x ^= y

عملگر بیتی (&)AND

عملگر بیتی AND کاری شبیه عملگر منطقی AND انجام می‌دهد با این تفاوت که این عملگر بر روی بیتها کار می‌کند. اگر مقادیر دو طرف آن ۱ باشد مقدار ۱ را بر می‌گرداند و اگر یکی یا هر دو طرف آن صفر باشد مقدار صفر را بر می‌گرداند. جدول درستی عمگر بیتی AND در زیر آمده است:

X AND Y	Y	X
1	1	1
0	0	1
0	1	0
0	0	0

در زیر نحوه استفاده از عملگر بیتی AND آمده است:

```
result = 5 & 3
print(result)
```

```
1
```

همانطور که در مثال بالا مشاهده می‌کنید نتیجه عملکرد عملگر AND بر روی دو مقدار ۵ و ۳ عدد یک می‌شود. اجازه بدهید ببینیم که چطور این نتیجه را به دست می‌آید:

```
5:  00000000000000000000000000000101
3:  00000000000000000000000000000011
-----------------------------------
1:  00000000000000000000000000000001
```

ابتدا دو عدد ۵ و ۳ به معادل باینری‌شان تبدیل می‌شوند. از آنجاییکه هر عدد صحیح (int) ۳۲ بیت است از صفر برای پر کردن بیتهای خالی استفاده می‌کنیم. با استفاده از جدول درستی عملگر بیتی AND می‌توان فهمید که چرا نتیجه عدد یک می‌شود.

عملگر بیتی (|)OR

اگر مقادیر دو طرف عملگر بیتی OR هر دو صفر باشند نتیجه صفر در غیر اینصورت ۱ خواهد شد. جدول درستی این عملگر در زیر آمده است:

X	Y	X OR Y
1	1	1
1	0	1
0	1	1
0	0	0

نتیجه عملگر بیتی OR در صورتی صفر است که عملوندهای دو طرف آن صفر باشند. اگر فقط یکی از دو عملوند یک باشد نتیجه یک خواهد شد. به مثال زیر توجه کنید:

```
result = 7 | 9
print(result)
```

```
15
```

وقتی که از عملگر بیتی OR برای دو مقدار در مثال بالا (۷ و ۹) استفاده می‌کنیم نتیجه ۱۵ می‌شود. حال بررسی می‌کنیم که چرا این نتیجه به دست آمده است؟

```
 7: 00000000000000000000000000000111
 9: 00000000000000000000000000001001
---------------------------------
15: 00000000000000000000000000001111
```

با استفاده از جدول درستی عملگر بیتی OR می‌توان نتیجه استفاده از این عملگر را تشخیص داد. عدد ۱۱۱۱ باینری معادل عدد ۱۵ صحیح است.

عملگر بیتی (^)XOR

جدول درستی این عملگر در زیر آمده است:

X	Y	X XOR Y
1	1	0
1	0	1

0	1	1
0	0	0

در صورتیکه عملوندهای دو طرف این عملگر هر دو صفر یا هر دو یک باشند نتیجه صفر در غیر اینصورت نتیجه یک می‌شود. در مثال زیر تأثیر عملگر بیتی XOR را بر روی دو مقدار مشاهده می‌کنید:

```
result = 5 ^ 7

print(result)
```

```
2
```

در زیر معادل باینری اعداد بالا (۵ و ۷) نشان داده شده است.

```
5: 00000000000000000000000000000101
7: 00000000000000000000000000000111
-----------------------------------
2: 00000000000000000000000000000010
```

با نگاه کردن به جدول درستی عملگر بیتی XOR، می‌توان فهمید که چرا نتیجه عدد ۲ می‌شود.

عملگر بیتی NOT (~)

این عملگر یک عملگر یگانی است و فقط به یک عملوند نیاز دارد. در زیر جدول درستی این عملگر آمده است:

X	NOT X
1	0
0	1

عملگر بیتی NOT مقادیر بیتها را معکوس می‌کند. در زیر چگونگی استفاده از این عملگر آمده است:

```

result = ~7

print(result)
```

```
-8
```

به نمایش باینری مثال بالا که در زیر نشان داده شده است توجه نمایید.

```
 7: 00000000000000000000000000000111
-----------------------------------

-8: 11111111111111111111111111111000
```

عملگر بیتی تغییر مکان (shift)

این نوع عملگرها به شما اجازه می‌دهند که بیتها را به سمت چپ یا راست جا به جا کنید. دو نوع عملگر بیتی تغییر مکان وجود دارد که هر کدام دو عملوند قبول می‌کنند. عملوند سمت چپ این عملگرها حالت باینری یک مقدار و عملوند سمت راست تعداد جابه جایی بیت‌ها را نشان می‌دهد.

عملگر	نام	مثال
<<	تغییر مکان به سمت چپ	x = y << 2
>>	تغییر مکان به سمت راست	x = y >> 2

عملگر تغییر مکان به سمت چپ

این عملگر بیتهای عملوند سمت چپ را به تعداد n مکان مشخص شده توسط عملوند سمت راست، به سمت چپ منتقل می‌کند. به عنوان مثال:

```
result = 10 << 2

print(result)
```

```
 40
```

در مثال بالا ما بیت‌های مقدار ۱۰ را دو مکان به سمت چپ منتقل کرده‌ایم، حال بیایید تأثیر این انتقال را بررسی کنیم:

```
10: 00000000000000000000000000001010
------------------------------------
40: 00000000000000000000000000101000
```

مشاهده می‌کنید که همه بیت‌ها به اندازه دو واحد به سمت چپ منتقل شده‌اند. در این انتقال دو صفر از صفرهای سمت چپ کم می‌شود و در عوض دو صفر به سمت راست اضافه می‌شود.

عملگر تغییر مکان به سمت راست

این عملگر شبیه به عمگر تغییر مکان به سمت چپ است با این تفاوت که بیت‌ها را به سمت راست جا به جا می‌کند. به عنوان مثال:

```
result = 100 >> 4

print(result)
```

```
 6
```

با استفاده از عملگرتغییر مکان به سمت راست بیت‌های مقدار ۱۰۰ را به اندازه ۴ واحد به سمت چپ جا به جا می‌کنیم. اجازه بدهید تأثیر این جا به جایی را مورد بررسی قرار دهیم:

```
100: 00000000000000000000000001100100
----------------------------------
  6: 00000000000000000000000000000110
```

هر بیت به اندازه ۴ واحد به سمت راست منتقل می‌شود، بنابراین ۴ بیت اول سمت راست حذف شده و چهار صفر به سمت چپ اضافه می‌شود.

عملگرهای خاص

علاوه بر عملگرهایی که تا کنون ذکر شد، پایتون دارای عملگرهای خاص زیر نیز می‌باشد:

- عملگرهای membership که بررسی می‌کنند آیا متغیر مورد نظر در یک مجموعه (sequence) همچون رشته، list یا tuple وجود دارد یا خیر.
- عملگرهای Identity که مکان‌های قرار گیری دو شیء را با هم مقایسه می‌کند (بررسی می‌کنند آیا دو شیء با هم برابر هستند یا خیر).

در جدول زیر انواع عملگرهای membership و Identity ذکر شده‌اند:

عملگر		توضیح
Memberships operator	in	در صورت یافتن متغیر مورد نظر در مجموعه‌ی مشخص شده، True و در غیر این صورت False را برمی گرداند.
	not in	در صورت یافت نشدن متغیر مورد نظر در مجموعه‌ی مشخص شده، True و در غیر این صورت False را برمی گرداند.
Identity operator	is	اگر متغیرهای هر دو طرف عملگر به شیء یکسان اشاره داشته باشند، True و در غیر این صورت False را برمی گرداند.
	is not	چنانچه متغیر در دو طرف عملگر به شیء یکسان اشاره داشته باشد، False و در غیر این صورت True را بر می‌گرداند.

به مثال‌های زیر توجه کنید:

```
print(5 in     [3, 8, 5, 10])
```

```
print(5 not in [3, 8, 5, 10])
```

```
True
False
```

در خط اول کد بالا، چک می‌شود که آیا عدد ۵ در مجموعه [3, 8, 5, 10] وجود دارد یا نه؟ و چون وجود دارد مقدار True بر گردانده می‌شود. در خط دوم هم که کاملاً مشخص است که اگر عدد ۵ در مجموعه وجود نداشته باشد مقدار True بر گردانده می‌شود ولی چون عدد ۵ وجود دارد مقدار False برگردانده می‌شود.

```
number1 = 5
number2 = 6

print(number1 is     number2)
print(number1 is not number2)
```

```
True
False
```

در مثال بالا و در اولین مقایسه گفته شده است که آیا number1 همان number2 است و چون چنین نیست مقدار False و در مقایسه دوم هم گفته شده است که آیا number1 برابر number2 نیست؟ و چون برابر نیستند مقدار True برگردانده شده است.

گرفتن ورودی از کاربر

پایتون تابع input() را برای گرفتن ورودی از کاربر، در اختیار شما قرار می‌دهد. همانطور که از نام این تابع پیداست، تمام کاراکترهایی را که شما در محیط برنامه‌نویسی تایپ می‌کنید تا زمانی که دکمه enter را می‌زنید، می‌خواند. به برنامه زیر توجه کنید:

```
1  name   = input("Enter your name: ")
2  age    = input("Enter your age: ")
3  height = input("Enter your height: ")
4
5  #Print a blank line
6  print()
7
8  #Show the details you typed
9  print("Name is {0}.".format(name))
10 print("Age is {0}.".format(age))
11 print("Height is {0}.".format(height))
```

```
Enter your name: John
Enter your age: 18
Enter your height: 160.5

Name is John.
Age is 18.
Height is 160.5.
```

ابتدا ۳ متغیر را برای ذخیره داده در برنامه تعریف می‌کنیم (خطوط ۱ و ۲ و ۳). برنامه از کاربر می‌خواهد که نام خود را وارد کند (خط ۱). در خط ۲ شما به عنوان کاربر نام خود را وارد می‌کنید. سپس برنامه از ما سن را سؤال می‌کند (خط ۳). در خط ۶ هم یک خط فاصله به وسیله

تابع print() ایجاد کرده‌ایم تا بین ورودی‌های شما و خروجی فاصله‌ای جهت تفکیک ایجاد شود. حال برنامه را اجرا کرده و با وارد کردن مقادیر مورد نظر نتیجه را مشاهده کنید.

ساختارهای تصمیم

تقریباً همه زبانهای برنامه‌نویسی به شما اجازه اجرای کد را در شرایط مطمئن می‌دهند. حال تصور کنید که یک برنامه دارای ساختار تصمیم گیری نباشد و همه کدها را اجرا کند. این حالت شاید فقط برای چاپ یک پیغام در صفحه مناسب باشد ولی فرض کنید که شما بخواهید اگر مقدار یک متغیر با یک عدد برابر باشد سپس یک پیغام چاپ شود آن وقت با مشکل مواجه خواهید شد. پایتون راه‌های مختلفی برای رفع این نوع مشکلات ارائه می‌دهد. در این بخش با مطالب زیر آشنا خواهید شد:

- دستور if
- دستور if…else
- عملگر سه تایی
- دستور if چندگانه
- دستور if تو در تو
- عملگرهای منطقی

دستور if

می‌توان با استفاده از دستور if و یک شرط خاص که باعث ایجاد یک کد می‌شود یک منطق به برنامه خود اضافه کنید. دستور if ساده‌ترین دستور شرطی است که برنامه می‌گوید اگر شرطی برقرار است کد معینی را انجام بده. ساختار دستور if به صورت زیر است:

```
if (condition):

   code to execute
```

قبل از اجرای دستور if ابتدا شرط بررسی می‌شود. اگر شرط برقرار باشد یعنی درست باشد سپس کد اجرا می‌شود. شرط یک عبارت مقایسه‌ای است. می‌توان از عملگرهای مقایسه‌ای برای تست درست یا اشتباه بودن شرط استفاده کرد. اجازه بدهید که نگاهی به نحوه استفاده از دستور if در داخل برنامه بیندازیم. برنامه زیر پیغام Hello World را اگر مقدار number کمتر از ۱۰ و Goodbye World را اگر مقدار number از ۱۰ بزرگ‌تر باشد در صفحه نمایش می‌دهد:

```
#Declare a variable and set it a value less than 10
number = 5

#If the value of number is less than 10
if (number < 10):
    print("Hello World.")

#Change the value of a number to a value which is greater than 10
```

```
9  number = 15
10
11 #If the value of number is greater than 10
12 if (number > 10):
13     print("Goodbye World.")
```

```
Hello World.
Goodbye World.
```

در خط ۲ یک متغیر با نام number تعریف و مقدار ۵ به آن اختصاص داده شده است. وقتی به اولین دستور if در خط ۲ می‌رسیم برنامه تشخیص می‌دهد که مقدار number از ۱۰ کمتر است یعنی ۵ کوچک‌تر از ۱۰ است.

منطقی است که نتیجه مقایسه درست می‌باشد، بنابراین دستور if دستور را اجرا می‌کند (خط ۶) و پیغام Hello World چاپ می‌شود. حال مقدار number را به ۱۵ تغییر می‌دهیم (خط ۹). وقتی به دومین دستور if در خط ۱۲ می‌رسیم برنامه مقدار number را با ۱۰ مقایسه می‌کند و چون مقدار number یعنی ۱۵ از ۱۰ بزرگ‌تر است برنامه پیغام Goodbye World را چاپ می‌کند (خط ۱۳). به این نکته توجه کنید که دستور if را می‌توان در یک خط نوشت:

```
if (number > 10): print("Goodbye World.")
```

شما می‌توانید چندین دستور را در داخل دستور if بنویسید. کافیست که حواستان به تو رفتگی کدها باشد. نحوه تعریف چند دستور در داخل بدنه if به صورت زیر است:

```
if (condition)
    statement1
    statement2
    .
    .
    .
    statementN
```

این هم یک مثال ساده:

```
x = 15

if (x > 10):
   print("x is greater than 10.")
   print("This is still part of the if statement.")
```

در مثال بالا اگر مقدار x از ۱۰ بزرگ‌تر باشد دو پیغام چاپ می‌شود. حال اگر به عنوان مثال، تو رفتگی خط آخر را حذف کنیم و مقدار x از ۱۰ بزرگ‌تر نباشد مانند کد زیر:

```
x = 5

if (x > 10):
   print("x is greater than 10.")
```

```
print("This is not part of the if statement.")
```

کد بالا در صورتی بهتر خوانده می‌شود که بین دستورات فاصله بگذاریم:

```
x = 5

if (x > 10):
   print("x is greater than 10.")

print("This is not part of the if statement.")
```

می‌بیند که دستور آخر در مثال بالا، جز دستور if نیست. اینجاست که چون ما فرض را بر این گذاشته‌ایم که مقدار x از ۱۰ کوچک‌تر است پس خط This is not part of the if statement چاپ می‌شود. در نتیجه اهمیت وجود تو رفتگی مشخص می‌شود. به عنوان تمرین همیشه حتی اگر فقط یک دستور در بدنه if داشتید برای آن یک تو رفتگی ایجاد کنید. فراموش نکنید که از قلم انداختن یک تو رفتگی باعث به وجود آمدن خطا شده و یافتن آن را سخت می‌کند. مثالی دیگر در مورد دستور if:

```
firstNumber  = input("Enter a number: ")
secondNumber = input("Enter another number: ")

if (firstNumber == secondNumber):
    print("{0} == {1}".format(firstNumber, secondNumber))

if (firstNumber != secondNumber):
    print("{0} != {1}".format(firstNumber, secondNumber))

if (firstNumber < secondNumber):
    print("{0} < {1}".format(firstNumber, secondNumber))

if (firstNumber > secondNumber):
    print("{0} > {1}".format(firstNumber, secondNumber))

if (firstNumber <= secondNumber):
    print("{0} <= {1}".format(firstNumber, secondNumber))

if (firstNumber >= secondNumber):
    print("{0} >= {1}".format(firstNumber, secondNumber))
```

```
Enter a number: 2
Enter another number: 5
2 != 5
2 < 5
2 <= 5
Enter a number: 10
Enter another number: 3
10 != 3
10 > 3
10 >= 3
Enter a number: 5
Enter another number: 5
5 == 5
5 <= 5
5 >= 5
```

ما از عملگرهای مقایسه‌ای در دستور `if` استفاده کرده‌ایم. ابتدا دو عدد که قرار است با هم مقایسه شوند را به عنوان ورودی از کاربر می‌گیریم. اعداد با هم مقایسه می‌شوند و اگر شرط درست بود پیغامی چاپ می‌شود. به این نکته توجه داشته باشید که شرطها مقادیر بولی هستند، یعنی دارای دو مقدار `True` یا `Flase` می‌باشند.

دستور `if...else`

دستور `if` فقط برای اجرای یک حالت خاص به کار می‌رود یعنی اگر حالتی برقرار بود کار خاصی انجام شود. اما زمانی که شما بخواهید اگر شرط خاصی برقرار شد یک دستور و اگر برقرار نبود دستور دیگر اجرا شود باید از دستور `if else` استفاده کنید. ساختار دستور `if else` در زیر آمده است:

```
if (condition):
   code to execute if condition is true
else:
   code to execute if condition is false
```

از کلمه کلیدی `else` نمی‌توان به تنهایی استفاده کرد بلکه حتماً باید با `if` به کار برده شود. اگر فقط یک کد اجرایی در داخل بدنه `if` و بدنه `else` دارید استفاده از آکولاد اختیاری است. کد داخل بلوک `else` فقط در صورتی اجرا می‌شود که شرط داخل دستور `if` نادرست باشد. در زیر نحوه استفاده از دستور `if...else` آمده است:

```
1  number = 5
2
3  #Test the condition
4  if (number < 10):
5      print("The number is less than 10.")
6  else:
7      print("The number is either greater than or equal to 10.")
8
9  #Modify value of number
10 number = 15
11
12 #Repeat the test to yield a different result
13 if (number < 10):
14     print("The number is less than 10.")
15 else:
16     print("The number is either greater than or equal to 10.")
```

```
The number is less than 10.
The number is either greater than or equal to 10.
```

در خط ۱ یک متغیر به نام `number` تعریف کرده‌ایم و در خط ۴ تست می‌کنیم که آیا مقدار متغیر `number` از ۱۰ کمتر است یا نه و چون کمتر است در نتیجه کد داخل بلوک `if` اجرا می‌شود (خط ۷) و اگر مقدار `number` را تغییر دهیم و به مقداری بزرگتر از ۱۰ تغییر دهیم (خط ۱۰)، شرط نادرست می‌شود (خط ۱۳) و کد داخل بلوک `else` اجرا می‌شود (خط ۱۶).

دستور if…elif…else

اگر بخواهید چند شرط را بررسی کنید چکار می‌کنید؟ می‌توانید از چندین دستور if استفاده کنید و بهتر است که این دستورات if را به صورت زیر بنویسید:

```
if (condition):
   code to execute
else:
   if (condition):
      code to execute
   else:
      if (condition):
         code to execute
      else:
         code to execute
```

خواندن کد بالا سخت است. بهتر است دستورات را به صورت تو رفتگی در داخل بلوک else بنویسید. می‌توانید کد بالا را ساده‌تر کنید:

```
if (condition):
   code to execute
elif (condition):
   code to execute
elif (condition):
   code to execute
else:
   code to execute
```

حال که نحوه استفاده از دستور if else را یاد گرفتید باید بدانید که مانند else، elif نیز به دستور if وابسته است. دستور elif وقتی اجرا می‌شود که اولین دستور if اشتباه باشد حال اگر elif اشتباه باشد دستور elif بعدی اجرا می‌شود. و اگر آن نیز اجرا نشود در نهایت دستور else اجرا می‌شود. برنامه زیر نحوه استفاده از دستور elif را نشان می‌دهد:

```
1  print("What's your favorite color?")
2  print("[1] Black")
3  print("[2] White")
4  print("[3] Blue")
5  print("[4] Red")
6  print("[5] Yellown")
7
8  choice = int(input("Enter your choice: "))
9
10 if (choice == 1):
11     print("You might like my black t-shirt.")
12 elif (choice == 2):
13     print("You might be a clean and tidy person.")
14 elif (choice == 3):
15     print("You might be sad today.")
16 elif (choice == 4):
17     print("You might be inlove right now.")
18 elif (choice == 5):
19     print("Lemon might be your favorite fruit.")
20 else:
21     print("Sorry, your favorite color is not in the choices above.")
```

```
What's your favorite color?
[1] Black
[2] White
[3] Blue
[4] Red
[5] Yellow

Enter your choice: 1
You might like my black t-shirt.
What's your favorite color?
[1] Black
[2] White
[3] Blue
[4] Red
[5] Yellow

Enter your choice: 999
Sorry, your favorite color is not in the choices above.
```

خروجی برنامه بالا به متغیر choice وابسته است. بسته به اینکه شما چه چیزی انتخاب می‌کنید پیغامهای مختلفی چاپ می‌شود. اگر عددی که شما تایپ می‌کنید در داخل حالتهای انتخاب نباشد کد مربوط به بلوک else اجرا می‌شود.

دستور if تو در تو

می‌توان از دستور if تو در تو در پایتون استفاده کرد. یک دستور ساده if در داخل دستور if دیگر:

```
if (condition):
    code to execute
    if (condition):
         code to execute
    elif (condition):
        if (condition):
           code to execute
else:
    if (condition):
       code to execute
```

اجازه بدهید که نحوه استفاده از دستور if تو در تو را نشان دهیم:

```
1  age = int(input("Enter your age: "))
2  gender = input("Enter your gender (male/female): ")
3
4  if (age > 12):
5      if (age < 20):
6          if (gender == "male"):
7              print("You are a teenage boy.")
8          else:
9              print("You are a teenage girl.")
10     else:
11         print("You are already an adult.")
12 else:
13     print("You are still too young.")
```

```
Enter your age: 18
Enter your gender: male
```

```
You are a teenage boy.
Enter your age: 12
Enter your gender: female
You are still too young.
```

اجازه بدهید که برنامه را کالبد شکافی کنیم. ابتدا برنامه از شما درباره سنتان سؤال می‌کند (خط ۱). در خط ۲ درباره جنستان از شما سؤال می‌کند. سپس به اولین دستور `if` می‌رسد (خط ۴). در این قسمت اگر سن شما بیشتر از ۱۲ سال باشد برنامه وارد بدنه دستور `if` می‌شود در غیر اینصورت وارد بلوک `else` (خط ۱۲) مربوط به همین دستور `if` می‌شود.

حال فرض کنیم که سن شما بیشتر از ۱۲ سال است و شما وارد بدنه اولین `if` شده‌اید. در بدنه اولین `if` دو دستور `if` دیگر را مشاهده می‌کنید. اگر سن کمتر ۲۰ باشد شما وارد بدنه `if` دوم می‌شوید و اگر نباشد به قسمت `else` متناظر با آن می‌روید (خط ۱۰). دوباره فرض می‌کنیم که سن شما کمتر از ۲۰ باشد، در اینصورت وارد بدنه `if` دوم شده و با یک `if` دیگر مواجه می‌شوید (خط ۶). در اینجا جنسیت شما مورد بررسی قرار می‌گیرد که اگر برابر "`male`" باشد، کدهای داخل بدنه سومین `if` اجرا می‌شود در غیر اینصورت قسمت `else` مربوط به این `if` اجرا می‌شود (خط ۸). پیشنهاد می‌شود که از `if` تو در تو در برنامه کمتر استفاده کنید چون خوانایی برنامه را پایین می‌آورد.

استفاده از عملگرهای منطقی

عملگرهای منطقی به شما اجازه می‌دهند که چندین شرط را با هم ترکیب کنید. این عملگرها حداقل دو شرط را در گیر می‌کنند و در آخر یک مقدار بولی را بر می‌گردانند. در جدول زیر برخی از عملگرهای منطقی آمده است:

عملگر	مثال	تأثیر
`and`	`z = (x > 2) and (y < 10)`	مقدار Z در صورتی `True` است که هر دو شرط دو طرف عملگر مقدارشان `True` باشد. اگر فقط مقدار یکی از شروط `False` باشد مقدار z، `False` خواهد شد.
`or`	`z = (x > 2) or (y < 10)`	مقدار Z در صورتی `True` است که یکی از دو شرط دو طرف عملگر مقدارشان `True` باشد. اگر هر دو شرط مقدارشان `False` باشد مقدار z، `False` خواهد شد.
`not`	`z = not(x > 2)`	مقدار Z در صورتی `True` است که مقدار شرط `False` باشد و در صورتی `False` است که مقدار شرط `True` باشد

به عنوان مثال جمله `z = (x > 2) and (y < 10)` را به این صورت بخوانید: "در صورتی مقدار z برابر `True` است که مقدار x بزرگ‌تر از 2 و مقدار y کوچک‌تر از ۱۰ باشد در غیر اینصورت `False` است". این جمله بدین معناست که برای اینکه مقدار کل دستور `True` باشد

باید مقدار همه شروط True باشد. عملگر منطقی or تأثیر متفاوتی نسبت به عملگر منطقی and دارد. نتیجه عملگر منطقی or برابر True است اگر فقط مقدار یکی از شروط True باشد. و اگر مقدار هیچ یک از شروط True نباشد نتیجه False خواهد شد. می‌توان عملگرهای منطقی and و or را با هم ترکیب کرده و در یک عبارت به کار برد مانند:

```
if ( (x == 1) and ( (y > 3) or z < 10) ) ):
   #do something here
```

در اینجا استفاده از پرانتز مهم است چون از آن در گروه بندی شرطها استفاده می‌کنیم. در اینجا ابتدا عبارت (y > 3) or (z < 10) مورد بررسی قرار می‌گیرد (به علت تقدم عملگرها). سپس نتیجه آن بوسیله عملگر and با نتیجه (x == 1) مقایسه می‌شود. حال بیایید نحوه استفاده از عملگرهای منطقی در برنامه را مورد بررسی قرار دهیم:

```
1  age = int(input("Enter your age: "))
2  gender = input("Enter your gender (male/female): ")
3
4  if (age > 12 and age < 20):
5      if (gender == "male"):
6          print("You are a teenage boy.")
7      else:
8          print("You are a teenage girl.")
9  else:
10     print("You are not a teenager.")
```

```
Enter your age: 18
Enter your gender (male/female): female
You are a teenage girl.
Enter you age: 10
Enter your gender (male/female): male
You are not a teenager.
```

برنامه بالا نحوه استفاده از عملگر منطقی and را نشان می‌دهد (خط ۴). وقتی به دستور if می‌رسید (خط ۴) برنامه سن شما را چک می‌کند. اگر سن شما بزرگتر از ۱۲ و کوچکتر از ۲۰ باشد (سنتان بین ۱۲ و ۲۰ باشد) یعنی مقدار هر دو True باشد سپس کدهای داخل بلوک if اجرا می‌شوند. اگر نتیجه یکی از شروط False باشد کدهای داخل بلوک else اجرا می‌شود. عملگر and عملوند سمت چپ را مورد بررسی قرار می‌دهد. اگر مقدار آن False باشد دیگر عملوند سمت راست را بررسی نمی‌کند و مقدار False را بر می‌گرداند. بر عکس عملگر or عملوند سمت چپ را مورد بررسی قرار می‌دهد و اگر مقدار آن True باشد سپس عملوند سمت راست را نادیده می‌گیرد و مقدار True را بر می‌گرداند.

```
if (x == 2 and y == 3):
   #Some code here

if (x == 2 or y == 3)
   #Some code here
```

نکته مهم اینجاست که شما می‌توانید از عملگرهای and و or به عنوان عملگر بیتی استفاده کنید. تفاوت جزئی این عملگرها وقتی که به عنوان عملگر بیتی به کار می‌روند این است که دو عملوند را بدون در نظر گرفتن مقدار عملوند سمت چپ مورد بررسی قرار می‌دهند. به عنوان مثال حتی اگر مقدار عملوند سمت چپ False باشد عملوند سمت چپ به وسیله عملگر بیتی and ارزیابی می‌شود. اگر شرطها را

در برنامه ترکیب کنید استفاده از عملگرهای منطقی and و or به جای عملگرهای بیتی and و or بهتر خواهد بود. یکی دیگر از عملگرهای منطقی عملگر not است که نتیجه یک عبارت را خنثی یا منفی می‌کند. به مثال زیر توجه کنید:

```
if (not(x == 2))
   print("x is not equal to 2.")
```

اگر نتیجه عبارت x == 2 برابر False باشد عملگر not آن را True می‌کند.

عملگر شرطی

عملگر شرطی در پایتون مانند دستور شرطی if…else عمل می‌کند. در زیر نحوه استفاده از این عملگر آمده است:

```
condition_is_true if condition else condition_is_false
```

عملگر شرطی تنها عملگر سه تایی پایتون است که نیاز به سه عملوند دارد، یک مقدار زمانی که شرط درست باشد، شرط و یک مقدار زمانی که شرط نادرست باشد. اجازه بدهید که نحوه استفاده این عملگر را در داخل برنامه مورد بررسی قرار دهیم:

```
1  pet1 = "puppy"
2  pet2 = "kitten"
3
4  type1 = "dog" if (pet1 == "puppy" ) else "cat"
5  type2 = "cat" if (pet2 == "kitten") else "dog"
6
7  print(type1)
8  print(type2)
```

```
dog
cat
```

برنامه بالا نحوه استفاده از این عملگر شرطی را نشان می‌دهد. خط ۴ به این صورت ترجمه می‌شود که مقدار dog را در متغیر type1 قرار بده اگر مقدار pet1 برابر با puppy بود در غیر این صورت مقدار cat را type1 قرار بده. خط ۵ به این صورت ترجمه می‌شود که مقدار cat را در type2 قرار بده اگر مقدار pet2 برابر با kitten بود در غیر این صورت مقدار dog. حال برنامه بالا را با استفاده از دستور if else می‌نویسیم:

```
if (pet1 == "puppy"):
   type1 = "dog"
else:
   type1 = "cat"
```

هنگامی که چندین دستور در داخل یک بلوک if یا else دارید از عملگر شرطی استفاده نکنید چون خوانایی برنامه را پایین می‌آورد.

نسخه کامل این کتاب را از سایت کتابراه به نشانی زیر دانلود کنید:

روی لینک کلیک کنید و با رعایت حروف بزرگ و کوچک در مرورگر تایپ و کلید Enter را بزنید

https://bit.ly/2nxcgxZ

انتقاد و پیشنهادات خود را به ایمیل زیر ارسال فرمایید:

Younes.ebrahimi.1391@gmail.com

از سایر کتاب های یونس ابراهیمی در لینک زیر دیدن فرمایید:

https://bit.ly/2kKGxYJ

تکرار

ساختارهای تکرار به شما اجازه می‌دهند که یک یا چند دستور کد را تا زمانی که یک شرط برقرار است تکرار کنید. بدون ساختارهای تکرار شما مجبورید همان تعداد کدها را بنویسید که بسیار خسته کننده است. مثلاً شما مجبورید ۱۰ بار جمله "Hello World." را تایپ کنید مانند مثال زیر:

```
print("Hello World!")
print("Hello World!")
print("Hello World!")
print("Hello World!")
print("Hello World!")
print("Hello World!")
print("Hello World!")
print("Hello World!")
print("Hello World!")
print("Hello World!")
```

البته شما می‌توانید با کپی کردن این تعداد کد را راحت بنویسید ولی این کار در کل کیفیت کدنویسی را پایین می‌آورد. راه بهتر برای نوشتن کدهای بالا استفاده از حلقه‌ها است. حلقه‌ها در پایتون عبارتند از:

- `while`
- `for`

حلقه While

ابتدایی‌ترین ساختار تکرار در پایتون حلقه `While` است. ابتدا یک شرط را مورد بررسی قرار می‌دهد و تا زمانیکه شرط برقرار باشد کدهای درون بلوک اجرا می‌شوند. ساختار حلقه `While` به صورت زیر است:

```
while(condition):
    code to loop
```

می‌بینید که ساختار `While` مانند ساختار `if` بسیار ساده است. ابتدا یک شرط را که نتیجه آن یک مقدار بولی است می‌نویسیم اگر نتیجه درست یا `true` باشد سپس کدهای داخل بلوک `While` اجرا می‌شوند. اگر شرط غلط یا `false` باشد وقتی که برنامه به حلقه `While` برسد هیچکدام از کدها را اجرا نمی‌کند. برای متوقف شدن حلقه باید مقادیر داخل حلقه `While` اصلاح شوند.

به یک متغیر شمارنده در داخل بدنه حلقه نیاز داریم. این شمارنده برای آزمایش شرط مورد استفاده قرار می‌گیرد و ادامه یا توقف حلقه به نوعی به آن وابسته است. این شمارنده را در داخل بدنه باید کاهش یا افزایش دهیم. در برنامه زیر نحوه استفاده از حلقه `While` آمده است:

```
counter = 1

while (counter <= 10):
    print("Hello World!")
    counter = counter + 1
```

```
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
```

برنامه بالا ۱۰ بار پیغام Hello World! را چاپ می‌کند. اگر از حلقه در مثال بالا استفاده نمی‌کردیم مجبور بودیم تمام ۱۰ خط را تایپ کنیم. اجازه دهید که نگاهی به کدهای برنامه فوق بیندازیم. ابتدا در خط ۱ یک متغیر تعریف و از آن به عنوان شمارنده حلقه استفاده شده است. سپس به آن مقدار ۱ را اختصاص می‌دهیم چون اگر مقدار نداشته باشد نمی‌توان در شرط از آن استفاده کرد.

در خط ۳ حلقه while را وارد می‌کنیم. در حلقه while ابتدا مقدار اولیه شمارنده با ۱۰ مقایسه می‌شود که آیا از ۱۰ کمتر است یا با آن برابر است. نتیجه هر بار مقایسه ورود به بدنه حلقه While و چاپ پیغام است. همانطور که مشاهده می‌کنید بعد از هر بار مقایسه مقدار شمارنده یک واحد اضافه می‌شود (خط ۵). حلقه تا زمانی تکرار می‌شود که مقدار شمارنده از ۱۰ کمتر باشد.

اگر مقدار شمارنده یک بماند و آن را افزایش ندهیم و یا مقدار شرط هرگز false نشود یک حلقه بینهایت به وجود می‌آید. به این نکته توجه کنید که در شرط بالا به جای علامت > از => استفاده شده است. اگر از علامت > استفاده می‌کردیم کد ما ۹ بار تکرار می‌شد چون مقدار اولیه ۱ است و هنگامی که شرط به ۱۰ برسد false می‌شود چون ۱۰ > ۱۰ نیست. اگر می‌خواهید یک حلقه بی نهایت ایجاد کنید که هیچگاه متوقف نشود باید یک شرط ایجاد کنید که همواره درست (true) باشد:

```
while(True):
   #code to loop
```

این تکنیک در برخی موارد کارایی دارد و آن زمانی است که شما بخواهید با استفاده از دستورات break و return که در آینده توضیح خواهیم داد از حلقه خارج شوید.

حلقه for

یکی دیگر از ساختارهای تکرار حلقه for است. این حلقه عملی شبیه به حلقه while انجام می‌دهد. ساختار حلقه for به صورت زیر است:

```
for iterator_var in sequence:
   code to repeat
```

iterator_var یک متغیر موقتی، in کلمه کلیدی و sequence هم یک سری مانند tuple، list و ... می‌باشد. می‌توان حلقه for را اینگونه ترجمه کرد، که به ازای یا به تعداد آیتم‌های موجود در سری، فلان کارها یا کدها را تکرار کن. در زیر یک مثال از حلقه for آمده است:

```
for i in [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]:
    print("Number ", i)
```

```
Number 1
Number 2
Number 3
Number 4
Number 5
Number 6
Number 7
Number 8
Number 9
Number 10
```

برنامه بالا اعداد ۱ تا ۱۰ را با استفاده از حلقه for می‌شمارد. ابتدا یک متغیر موقتی (i)، سپس کلمه کلیدی in و در آخر یک سری از اعداد که در اینجا یک list می‌باشد، تعریف می‌کنیم. کد اجرا می‌شود. هر بار که حلقه اجرا می‌شود، ابتدا یکی از آیتم‌های list در متغیر i قرار گرفته و در خط بعد چاپ می‌شود. این کار تا چاپ آخرین آیتم ادامه می‌یابد. به جای list در کد بالا می‌توانید از tuple و dictionary هم استفاده کنید:

```
for i in (1, 2, 3, 4, 5, 6, 7, 8, 9, 10):
```

یا

```
for i in {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}:
```

حلقه‌های تو در تو (Nested Loops)

پایتون به شما اجازه می‌دهد که از حلقه‌ها به صورت تو در تو استفاده کنید. اگر یک حلقه در داخل حلقه دیگر قرار بگیرد، به آن حلقه تو در تو گفته می‌شود. در این نوع حلقه‌ها، به ازای اجرای یک بار حلقه بیرونی، حلقه داخلی به طور کامل اجرا می‌شود. در زیر نحوه ایجاد حلقه تو در تو آمده است:

```
for iterator_var in sequence:
    for iterator_var in sequence:
        statements(s)
        statements(s)
```

```
while expression:
    while expression:
        statement(s)
        statement(s)
```

نکته‌ای که در مورد حلقه‌های تو در تو وجود دارد این است که، می‌توان از یک نوع حلقه در داخل نوع دیگر استفاده کرد. مثلاً می‌توان از حلقه for در داخل حلقه while استفاده نمود. در مثال زیر نحوه استفاده از این حلقه‌ها ذکر شده است. فرض کنید که می‌خواهید یک مستطیل با ۳ سطر و ۵ ستون ایجاد کنید:

```
for i in (1, 2, 3, 4):
    for j in (1, 2, 3, 4, 5):
        print("*", end=' ')
    print()
```

```
* * * * *
* * * * *
* * * * *
* * * * *
```

در کد بالا به ازای یک بار اجرای حلقه for اول (خط ۱)، حلقه for دوم (۳-۲) به طور کامل اجرا می‌شود. یعنی وقتی مقدار i برابر عدد ۱ می‌شود، علامت * توسط حلقه دوم ۵ بار چاپ می‌شود، وقتی i برابر ۲ می‌شود، دوباره علامت * پنج بار چاپ می‌شود و .... در کل منظور از دو حلقه for این است که در ۴ سطر علامت * در ۵ ستون چاپ شود یا ۴ سطر ایجاد شود و در هر سطر ۵ بار علامت * چاپ شود. خط ۴ هم برای ایجاد خط جدید است. یعنی وقتی حلقه داخلی به طور کامل اجرا شد، یک خط جدید ایجاد می‌شود و علامت‌های * در خطوط جدید چاپ می‌شوند.

خارج شدن از حلقه با استفاده از continue،break و pass

گاهی اوقات با وجود درست بودن شرط می‌خواهیم حلقه متوقف شود. سؤال اینجاست که چطور این کار را انجام دهید؟ با استفاده از کلمه کلیدی break حلقه را متوقف کرده و با استفاده از کلمه کلیدی continue می‌توان بخشی از حلقه را رد کرد و به مرحله بعد رفت. برنامه زیر نحوه استفاده از break،continue و pass را نشان می‌دهد:

```
print("Demonstrating the use of break\n")

for x in (1, 2, 3, 4, 5, 6, 7, 8, 9, 10):
    if (x == 5):
        break

    print("Number ", x)

print("\nDemonstrating the use of continue\n")

for x in (1, 2, 3, 4, 5, 6, 7, 8, 9, 10):
    if (x == 5):
        continue

    print("Number ", x)

print("\nDemonstrating the use of pass\n")

for x in (1, 2, 3, 4, 5, 6, 7, 8, 9, 10):
    if (x == 5):
        pass
```

```
Demonstrating the use of break

Number  1
Number  2
Number  3
Number  4
```

```
Demonstrating the use of continue

Number  1
Number  2
Number  3
Number  4
Number  6
Number  7
Number  8
Number  9
Number  10
Demonstrating the use of pass
```

در این برنامه از حلقه for برای نشان دادن کاربرد دو کلمه کلیدی فوق استفاده شده است اگر به جای for از حلقه while استفاده می‌شد نتیجه یکسانی به دست می‌آمد. همانطور که در شرط برنامه (خط ۴) آمده است، وقتی که مقدار x به عدد ۵ برسد، سپس دستور break اجرا (خط ۵) و حلقه بلافاصله متوقف می‌شود، حتی اگر شرط 10 > x برقرار باشد. از طرف دیگر در خط ۲۱ حلقه for فقط برای یک تکرار خاص متوقف شده و سپس ادامه می‌یابد. وقتی مقدار x برابر ۵ شود حلقه از ۵ رد شده و مقدار ۵ را چاپ نمی‌کند و بقیه مقادیر چاپ می‌شوند.

ممکن است این سؤال برایتان پیش آمده باشد که کاربرد کلمه pass چیست؟ از این دستور زمانی استفاده می‌کنیم که در شرایطی خاص نیاز به انجام هیچ کاری نباشد! مثلاً برای تعریف یک تابع خالی تا بعداً کدهای آن نوشته شود. یا زمانی که بخواهیم همانند مثلاً بالا، کدهای بدنه یک دستور شرطی و یا حلقه را بعداً بنویسیم، به کار می‌رود. حال شما برای درک بهتر، کلمه pass را از کد بالا حذف کرده و کد را اجرا کنید. مشاهده می‌کنید که به شما پیغام خطا نمایش داده می‌شود و از شما می‌خواهد که بدنه دستور if و for را مشخص کنید ولی اگر کلمه pass را دوباره بنویسید، این خطا نادیده گرفته و کد اجرا می‌شود.

تابع

توابع به شما اجازه می‌دهند که یک رفتار یا وظیفه را تعریف کنید و مجموعه‌ای از کدها هستند که در هر جای برنامه می‌توان از آنها استفاده کرد. توابع دارای آرگومانهایی هستند که وظیفه تابع را مشخص می‌کنند. می‌توان را می توان در داخل کلاس و یا تابع دیگر تعریف کرد. وقتی که شما در برنامه یک تابع را صدا می‌زنید برنامه به قسمت تعریف تابع رفته و کدهای آن را اجرا می‌کند.

> پارامترها همان چیزهایی هستند که تابع منتظر دریافت آنها است.
>
> آرگومان‌ها مقادیری هستند که به پارامترها ارسال می‌شوند.

گاهی اوقات دو کلمه پارامتر و آرگومان به یک منظور به کار می‌روند. ساده‌ترین ساختار یک تابع به صورت زیر است:

```
def functionname(Parameter List):
   code to execute
```

به برنامه ساده زیر توجه کنید. در این برنامه از یک تابع برای چاپ یک پیغام در صفحه نمایش استفاده شده است:

```
def printmessage():
   print("Hello World!")

printmessage()
```

```
Hello World!
```

در خطوط ۲-۱ یک تابع تعریف کرده‌ایم. در تعریف تابع بالا کلمه کلیدی def آمده است که نشان دهنده تعریف تابع است. نام تابع ما printmessage() است. به این نکته توجه کنید که در پایتون و طبق قرارداد نام توابع به صورت حروف کوچک نوشته می شود و اگر در نامگذاری تابع از دو یا چند کلمه استفاده شود، بهتر است که آنها را با علامت _ از هم جدا کنید. این روش نامگذاری قراردادی است و می‌توان از این روش استفاده نکرد، اما پیشنهاد می‌شود که از این روش برای تشخیص توابع استفاده کنید. بهتر است در نامگذاری توابع از کلماتی استفاده شود که کار آن تابع را مشخص می‌کند مثلاً نام‌هایی مانند GoToBed یا OpenDoor. دو پرانتزی که بعد از نام می‌آید نشان دهنده آن است که نام متعلق به یک تابع است. در این مثال در داخل پرانتزها هیچ چیزی نوشته نشده چون پارامتری ندارد. در درسهای آینده در مورد توابع بیشتر توضیح می‌دهیم.

بعد از پرانتزها علامت: قرار می‌دهیم و کدهایی را که می‌خواهیم اجرا شوند را به صورت تو رفتگی بعد از علامت: و در خط زیر می‌نویسیم. در خط ۴ تابع printmessage() را صدا می‌زنیم. برای صدا زدن یک تابع کافیست نام آن را نوشته و بعد از نام پرانتزها را قرار دهیم.

به این نکته توجه کنید که برای اجرای کدهای تابع، هنگام فراخوانی تابع، حتماً باید بعد از نام تابع، علامت‌های پرانتز را هم قرار دهید.

اگر تابع دارای پارامتر باشد باید شما آرگومانها را به ترتیب در داخل پرانتزها قرار دهید. در این مورد نیز در درسهای آینده توضیح بیشتری می‌دهیم. با صدا زدن یک تابع کدهای داخل بدنه آن اجرا می‌شوند. برای اجرای تابع printmessage() برنامه به محل تعریف تابع printmessage() می‌رود. مثلاً وقتی ما تابع printmessage() را در خط ۴ صدا می‌زنیم برنامه از خط ۴ به خط ۱، یعنی جایی که تابع تعریف شده می‌رود و کدهای بدنه آن را اجرا می‌کند.

مقدار برگشتی از یک تابع

توابع می‌توانند مقدار برگشتی از هر نوع داده‌ای داشته باشند. این مقادیر می‌توانند در محاسبات یا به دست آوردن یک داده مورد استفاده قرار بگیرند. در زندگی روزمره فرض کنید که کارمند شما یک تابع است و شما او را صدا می‌زنید و از او می‌خواهید که کار یک سند را به پایان برساند. سپس از او می‌خواهید که بعد از اتمام کارش سند را به شما تحویل دهد. سند همان مقدار برگشتی تابع است. نکته مهم در مورد یک تابع، مقدار برگشتی و نحوه استفاده شما از آن است. برگشت یک مقدار از یک تابع آسان است. کافیست در تعریف تابع به روش زیر عمل کنید:

```
def functionname()
```

```
    return value
```

در داخل بدنه تابع کلمه کلیدی return و بعد از آن یک مقدار یا عبارتی که نتیجه آن یک مقدار است را می‌نویسیم. مثال زیر یک تابع که دارای مقدار برگشتی است را نشان می‌دهد.

```
1  def calculatesum():
2      firstNumber = 10
3      secondNumber = 5
4      sum = firstNumber + secondNumber
5
6      return sum
7
8  result = calculatesum()
9
10 print("Sum is {0}.".format(result))
```

```
Sum is 15.
```

همانطور که مشاهده می‌کنید، در خطوط ۶-۱ یک تابع تعریف کرده‌ایم. در خطوط ۲ و ۳ دو متغیر تعریف و مقدار دهی شده‌اند. توجه کنید که این متغیرها، متغیرهای محلی هستند. و این بدان معنی است که این متغیرها در سایر توابع، قابل دسترسی نیستند و فقط در تابعی که در آن تعریف شده‌اند قابل استفاده هستند. در خط ۴ جمع دو متغیر در متغیر sum قرار می‌گیرد. در خط ۶ مقدار برگشتی sum توسط دستور return فراخوانی می‌شود. در خط ۸ یک متغیر به نام result تعریف کرده و تابع calculatesum() را فراخوانی می‌کنیم.

تابع calculatesum() مقدار ۱۵ را بر می‌گرداند که در داخل متغیر result ذخیره می‌شود. در خط ۱۰ مقدار ذخیره شده در متغیر result چاپ می‌شود. تابعی که در این مثال ذکر شد تابع کاربردی و مفیدی نیست. با وجودیکه کدهای زیادی در تابع بالا نوشته شده ولی همیشه مقدار برگشتی ۱۵ است، در حالیکه می‌توانستیم به راحتی یک متغیر تعریف کرده و مقدار ۱۵ را به آن اختصاص دهیم. این تابع در صورتی کارآمد است که پارامترهایی به آن اضافه شود که در درسهای آینده توضیح خواهیم داد. هنگامی که می‌خواهیم در داخل یک تابع از دستور if استفاده کنیم باید تمام کدها دارای مقدار برگشتی باشند. برای درک بهتر این مطلب به مثال زیر توجه کنید:

```
1  def getnumber():
2      number = int(input("Enter a number greater than 10: "))
3      if number > 10:
4          return number
5      else:
6          return 0
7
8  result = getnumber()
9
10 print("Result = {0}.".format(result))
```

```
Enter a number greater than 10: 11
Result = 11
Enter a number greater than 10: 9
Result = 0
```

در خطوط ۶-۱ یک تابع با نام getnumber() تعریف شده است که از کاربر یک عدد بزرگتر از ۱۰ را می‌خواهد. اگر عدد وارد شده توسط کاربر درست نباشد تابع مقدار صفر را بر می‌گرداند. و اگر قسمت else دستور if و یا دستور return را از آن حذف کنیم در هنگام اجرای برنامه با پیغام خطا مواجه می‌شویم.

چون اگر شرط دستور if نادرست باشد (کاربر مقداری کمتر از ۱۰ را وارد کند) برنامه به قسمت else می‌رود تا مقدار صفر را بر گرداند و چون قسمت else حذف شده است برنامه با خطا مواجه می‌شود و همچنین اگر دستور return حذف شود چون برنامه نیاز به مقدار برگشتی دارد پیغام خطا می‌دهد. و آخرین مطلبی که در این درس می‌خواهیم به شما آموزش دهیم این است که شما می‌توانید از یک تابع که مقدار برگشتی ندارد خارج شوید. استفاده از return باعث خروج از بدنه تابع و اجرای کدهای بعد از آن می‌شود:

```
1  def testreturnexit():
2      print("Line 1 inside the method testreturnexit()")
3      print("Line 2 inside the method testreturnexit()")
4
5      return
6
7      #The following lines will not execute
8      print("Line 3 inside the method testreturnexit()")
9      print("Line 4 inside the method testreturnexit()")
10
11 testreturnexit()
12 print("Hello World!")
```

```
Line 1 inside the method testreturnexit()
Line 2 inside the method testreturnexit()
Hello World!
```

در برنامه بالا نحوه خروج از تابع با استفاده از کلمه کلیدی return و نادیده گرفتن همه کدهای بعد از این کلمه کلیدی نشان داده شده است. در کد بالا انتظار ما این است که با فراخوانی تابع در خط ۱۱، همه کدهای بدنه تابع (۹-۲) اجرا شوند. ولی با فراخوانی تابع خطوط ۲ و ۳ چاپ می‌شوند، چون هنگامی که برنامه به خط ۵ می‌رسد، از بدنه تابع خارج می‌شود. سپس مفسر به خط ۱۲ رفته و رشته Hello World را چاپ می‌کند.

پارامترها و آرگومان‌ها

پارامترها داده‌های خامی هستند که تابع آنها را پردازش می‌کند و سپس اطلاعاتی را که به دنبال آن هستید، در اختیار شما قرار می‌دهد. فرض کنید پارامترها مانند اطلاعاتی هستند که شما به یک کارمند می‌دهید که بر طبق آنها کارش را به پایان برساند. یک تابع می‌تواند هر تعداد پارامتر داشته باشد. هر پارامتر می‌تواند از انواع مختلف داده باشد. در زیر یک تابع با N پارامتر نشان داده شده است:

```
def functionname(param1, param2, ... paramN):
   code to execute
```

پارامترها بعد از نام تابع و بین پرانتزها قرار می‌گیرند. بر اساس کاری که تابع انجام می‌دهد می‌توان تعداد پارامترهای زیادی به تابع اضافه کرد. بعد از فراخوانی یک تابع باید آرگومانهای آن را نیز تأمین کنید. آرگومان‌ها مقادیری هستند که به پارامترها اختصاص داده می‌شوند. اجازه بدهید که یک مثال بزنیم:

```
1  def calculatesum(number1, number2):
2    return number1 + number2
3
4  num1 = int(input("Enter the first number: "))
5  num2 = int(input("Enter the second number: "))
6
7  print("Sum = {0}".format(calculatesum(num1, num2)))
```

```
Enter the first number: 10
Enter the second number: 5
Sum = 15
```

در برنامه بالا یک تابع به نام `calculatesum()` (خطوط ۲-۱) تعریف شده است و می‌خواهیم مقدار دو عدد را با این تابع حمع کنیم. در بدنه تابع دستور `return` نتیجه جمع دو عدد را بر می‌گرداند. در خطوط ۴ و ۵ برنامه از کاربر دو مقدار را درخواست می‌کند و آنها را داخل متغیرها قرار می‌دهد. حال تابع را که آرگومانهای آن را آماده کرده‌ایم فراخوانی می‌کنیم. مقدار `num1` به پارامتر اول و مقدار `num2` به پارامتر دوم ارسال می‌شود. حال اگر مکان دو مقدار را هنگام ارسال به تابع تغییر دهیم (یعنی مقدار `num2` به پارامتر اول و مقدار `num1` به پارامتر دوم ارسال شود) هیچ تغییری در نتیجه تابع ندارد چون جمع خاصیت جابه جایی دارد.

فقط به یاد داشته باشید که باید تعداد آرگومانها هنگام فراخوانی تابع دقیقاً با تعداد پارامترها تعریف شده در تابع مطابقت داشته باشد. بعد از ارسال مقادیر ۱۰ و ۵ به پارامترها، پارامترها آنها را دریافت می‌کنند. به این نکته نیز توجه کنید که نام پارامترها طبق قرارداد به شیوه کوهان شتری یا `camelCasing` (حرف اول دومین کلمه بزرگ نوشته می‌شود) نوشته می‌شود. در داخل بدنه تابع (خط ۲) دو مقدار با هم جمع می‌شوند و نتیجه به تابع فراخوان (تابعی که تابع `()calculatesum` را فراخوانی می‌کند) ارسال می‌شود. در درس آینده از یک متغیر برای ذخیره نتیجه محاسبات استفاده می‌کنیم ولی در اینجا مشاهده می‌کنید که می‌توان به سادگی نتیجه جمع را نشان داد (خط ۷). در خط ۷ تابع `()calculatesum` را فراخوانی می‌کنیم و دو مقدار صحیح به آن ارسال می‌کنیم. دو عدد صحیح در داخل تابع با هم جمع شده و نتیجه آنها برگردانده می‌شود. مقدار برگشت داده شده از تابع به وسیله تابع `()print` نمایش داده می‌شود. و نکته آخر اینکه یک تابع را می‌توان به عنوان آرگومان به تابع دیگر ارسال کرد. به کد زیر توجه کنید:

```
1  def functionA(myFunction):
2    return myFunction()
3
4  def functionB():
5      return "Hello World!"
6
7  print("{0}".format(functionA(functionB)))
```

```
Hello World!
```

در کد بالا ما دو تابع تعریف کرده‌ایم. که تابع اول یعنی functionA قرار است که یک آرگومان از نوع تابع دریافت کند. برای این منظور بعد از تعریف پارامتر در خط ۱ در خط ۲ و بعد از نام پارامتر حتماً باید علامت‌های پرانتز را بنویسید. این بدین معنی است که مقدار برگشتی از تابع functionA یک تابع است. در نتیجه هنگامی که در خط ۷ ما functionB را به عنوان آرگومان به تابع functionA می‌دهیم، کلمه functionA در خط ۲ جایگزین کلمه myFunction شده و در نتیجه این خط همانند فراخوانی تابع functionB عمل می‌کند (یعنی فراخوانی یک تابع همراه با پرانتزهای آن).

آرگومان‌های کلمه کلیدی (Keyword Arguments)

یکی دیگر از راه‌های ارسال آرگومانها استفاده از نام آنهاست. استفاده از نام آرگومانها شما را از به یاد آوری و رعایت ترتیب پارامترها هنگام ارسال آرگومان‌ها راحت می‌کند. در عوض شما باید نام پارامترهای تابع را به خاطر بسپارید. استفاده از نام آرگومانها خوانایی برنامه را بالا می‌برد چون شما می‌توانید ببینید که چه مقادیری به چه پارامترهایی اختصاص داده شده است. در زیر نحوه استفاده از آرگومان‌های کلمه کلیدی، وقتی که تابع فراخوانی می‌شود نشان داده شده است:

```
functionToCall( paramName1 = value, paramName2 = value, ... paramNameN = value)
```

حال به مثال زیر توجه کنید:

```
1  def tellinformation(jack, andy, mark):
2      print("Jack's family is {0}.".format(jack))
3      print("Andy's family is {0}.".format(andy))
4      print("Mark's family is {0}.".format(mark))
5
6  tellinformation(jack = "Scalia", andy = "Brown", mark = "OverMars")
7
8  #Print a newline
9  print()
10
11 tellinformation(andy = "Brown", mark = "OverMars", jack = "Scalia")
12
13 print()
14
15 tellinformation(mark = "OverMars", jack = "Scalia", andy = "Brown")
```

```
Jack's family is Scalia.
Andy's family is Brown.
Mark's family is OverMars.

Jack's family is Scalia.
Andy's family is Brown.
Mark's family is OverMars.

Jack's family is Scalia.
Andy's family is Brown.
Mark's family is OverMars.
```

خروجی نشان می‌دهد که حتی اگر ما ترتیب آرگومانها در سه بار فراخوانی تابع را تغییر دهیم مقادیر مناسب به پارامترهای مربوطه‌شان اختصاص داده می‌شود. همچنین می‌توان از آرگومان‌های کلمه کلیدی و آرگومانهای ثابت (مقداری) به طور همزمان استفاده کرد به شرطی که آرگومانهای ثابت قبل از آرگومان‌های کلمه کلیدی قرار بگیرند:

```
#The following codes are currect

tellinformation("Scalia", andy = "Brown", mark = "OverMars")

tellinformation("Scalia", mark = "OverMars", andy = "Brown")

#The following codes are wrong and will lead to errors

tellinformation(jack = "Scalia", andy = "Brown", "OverMars")

tellinformation(jack = "Scalia", "OverMars", andy = "Brown")
```

همانطور که مشاهده می‌کنید ابتدا باید آرگومانهای ثابت هنگام فراخوانی تابع ذکر شوند. در اولین و دومین فراخوانی در کد بالا، مقدار "Scalia" را به عنوان اولین آرگومان به اولین پارامتر تابع یعنی jack اختصاص می‌دهیم. سومین و چهارمین خط کد بالا اشتباه هستند چون آرگومان‌های کلمه کلیدی، قبل از آرگومانهای ثابت قرار گرفته‌اند. قرار گرفتن آرگومان‌های کلمه کلیدی بعد از آرگومانهای ثابت از بروز خطا جلوگیری می‌کند.

آرگومان‌های متغیر

با استفاده از دستورات خاص *args و **kwargs می‌توان تعداد دلخواهی از آرگومان‌ها را به تابع ارسال کرد. همانطور که در درس‌های قبل ذکر شد، هنگام فراخوانی تابع باید به تعداد پارامترهایی که در داخل پرانتز تعریف شده‌اند، آرگومان به تابع ارسال کرد. گاهی اوقات در برنامه‌نویسی ممکن است بخواهید که در هر بار فراخوانی تابع تعداد دلخواهی آرگومان به آن ارسال کنید. این کار با استفاده از * و ** ممکن است. به کد زیر توجه کنید:

```
def varargumentS(*args):
    total = 0
    for number in args:
        total = total + number
    return total

print("1 + 2 + 3 = {0}".format(varargumentS(1, 2, 3)))
print("1 + 2 + 3 + 4 = {0}".format(varargumentS(1, 2, 3, 4)))
print("1 + 2 + 3 + 4 + 5 = {0}".format(varargumentS(1, 2, 3, 4, 5)))
```

```
6 = 3 + 2 + 1
10 = 4 + 3 + 2 + 1
15 = 5 + 4 + 3 + 2 + 1
```

ابتدا به این نکته توجه کنید که نامهای args و kwargs اختیاری هستند و هم نام دیگری می‌تواند به جای آنها به کار رود. تنها چیزی که مهم است تعداد علامت * می‌باشد که در ادامه کاربرد آنها را توضیح می‌دهیم.

همانطور که در کد بالا مشاهده می‌کنید، با قرار دادن یک علامت ستاره قبل از نام پارامتر، می‌توان هر بار که تابع را فراخوانی کرد، تعداد دلخواهی از آرگومانها را به آن ارسال کرد. وجود یک علامت * باعث می‌شود که آرگومانها در یک متغیر از جنس tuple ذخیره شوند و در نتیجه می‌توان با یک دستور for مقادیر آنها را با هم جمع کرد. این نوع پارامتر را می‌توان با پارامترهای ثابت هم به کار برد. به مثال زیر توجه کنید:

```
def varargumentes(number, *args):
    print("number = ", number)
    print("args = ", args)

varargumentes(1, 2, 3)
```

```
number =  1
args =  (2, 3)
```

در کد بالا اولین آرگومان به اولین پارامتر (یعنی ۱ به number) و بقیه آرگومانها به args اختصاص داده می‌شوند. وقتی از چندین پارامتر در یک تابع استفاده می‌کنید فقط یکی از آنها باید دارای * بوده و همچنین از لحاظ مکانی باید آخرین پارامتر باشد. اگر این پارامتر (پارامتری که دارای علامت * است) در آخر پارامترهای دیگر قرار نگیرد و یا از چندین پارامتر علامت دار استفاده کنید با خطا مواجه می‌شوید. به مثالهای اشتباه و درست زیر توجه کنید:

```
def somefunction(*args, *args) #ERROR

def somefunction(*args, param1, param2) #ERROR

def somefunction(param1, param2, *args) #Correct
```

البته می‌توان پارامتر ستاره دار را در ابتدای پارامترهای دیگر قرار داد ولی پارامترهای بعد از این پارامتر یا باید دارای مقدار پیشفرض باشند

```
def varargumentes(*args , number = 10):
    print("args = {0}".format(args))
    print("number = {0}".format(number))

varargumentes(1, 3, 5)
```

```
args = (1, 3, 5)
number = 10
```

و یا هنگام فراخوانی تابع، باید پارامترهای بعد از این پارامتر را با استفاده از اسمشان مقداردهی کرد. به کد زیر توجه کنید:

```
def varargumentes(*args , number):
    print("args   = {0}".format(args))
    print("number = {0}".format(number))

varargumentes(1, 3, 5, number = 10)
```

```
args   = (1, 3, 5)
number = 10
```

حال فرض کنید که می‌خواهید چند list یا tuple به پارامتر ستاره دار ارسال کنید. به کد زیر توجه نمایید:

```
def varargumentes(number, *args):
    print("number = {0}".format(number))
    print("args   = {0}".format(args))

varargumentes(1, *(2,3,4), *(5,6,7,8))
```

```
number = 1
args   = (2, 3, 4, 5, 6, 7, 8)
```

همانطور که در کد بالا مشاهده می‌کنید، کافیست که آرگومان‌ها به صورت `list` یا `tuple` ارسال شوند و قبل از آنها علامت * را قرار دهیم.

خط آخر کد بالا را به صورت زیر هم می‌توان نوشت:

```
varargumentes(1, *[2,3,4], *[5,6,7,8])
```

**kwargs هم شبیه *args عمل می‌کند با این تفاوت که هنگام فراخوانی تابع باید آرگومان‌ها را به صورت کلید/ مقدار به آن ارسال کرد تا به صورت `dictionary` ذخیره کند:

```
def varargumentes(**kwargs):
    print(kwargs)

varargumentes(person1 = "Jack", person2 = "Joe", person3 = "Smith")
```

```
{'person1': 'Jack', 'person2': 'Joe', 'person3': 'Smith'}
```

محدوده متغیر

متغیرها در پایتون دارای محدوده (`scope`) هستند. محدوده یک متغیر به شما می‌گوید که در کجای برنامه می‌توان از متغیر استفاده کرد و یا متغیر قابل دسترسی است. به عنوان مثال متغیری که در داخل یک تابع تعریف می‌شود فقط در داخل بدنه تابع قابل دسترسی است. می‌توان دو متغیر با نام یکسان در دو تابع مختلف تعریف کرد. برنامه زیر این ادعا را اثبات می‌کند:

```
def firstfunction():
    number = 5
    print("number inside method firstMethod() = {0}".format(number))

def secondfunction():
    number = 10
    print("number inside method secondMethod() = {0}".format(number))

firstFunction()
secondFunction()
```

```
number inside method firstFunction() = 5
number inside method secondFunction() = 10
```

مشاهده می‌کنید که حتی اگر ما دو متغیر با نام یکسان تعریف کنیم که دارای محدوده‌های متفاوتی هستند، می‌توان به هر کدام از آنها مقادیر مختلفی اختصاص داد. متغیر تعریف شده در داخل تابع `firstfunction()` هیچ ارتباطی به متغیر داخل تابع

secondfunction() ندارد. همانطور که ذکر شد، متغیری که در داخل بدنه یک تابع تعریف شود در خارج از تابع قابل دسترسی نیست.

به مثال زیر توجه کنید:

```
def myfunction():
    number = 10

print(number)
```

در تابع بالا یک متغیر به نام number تعریف شده است. اگر بخواهیم در خارج از تابع یعنی خط آخر مقدار این متغیر را چاپ کنیم با پیغام خطا مواجه می‌شویم. چون این متغیر فقط در داخل تابع قابل دسترسی است. به این متغیرها محلی یا Local گفته می‌شود. یک نوع دیگر از متغیرها، عمومی یا global هستند. این متغیرها در خارج از تابع تعریف می‌شوند و در داخل بدنه تابع قابل دسترسی هستند. به مثال زیر توجه کنید:

```
number = 10

def myfunction():
    print(number)

myFunction()
```

```
10
```

در کد بالا یک متغیر به نام number با مقدار ۱۰ در خارج از تابع تعریف شده است. از تابع myfunction() خواسته‌ایم که مقدار این متغیر را در هنگام فراخوانی چاپ کند. این اتفاق می افتد، چون متغیرهای خارج از تابع در داخل تابع قابل دسترسی هستند. حال اگر بخواهیم از یک متغیر محلی به صورت عمومی و خارج از تابع استفاده کنیم باید چکار کنیم؟ راهکار، استفاده از کلمه کلیدی global است. به مثال زیر توجه کنید:

```
1  def myfunction():
2      global number
3      number = 10
4      print(number)
5
6  number = 15
7  print(number)
8
9  myFunction()
```

```
15
10
```

در کد بالا یک متغیر تعریف کرده‌ایم و قبل از آن کلمه کلیدی global را نوشته‌ایم. این کلمه به برنامه می‌فهماند که قرار است از فلان متغیر در خارج از تابع استفاده شود. همانطور که مشاهده می‌کنید با وجود کلمه global می‌توان به متغیر number در خارج از تابع دسترسی داشت. ما در اینجا فقط مقدار number را خارج از تابع تغییر داده‌ایم. ولی شما ممکن است که بخواهید از آن، استفاده‌های دیگری بکنید. در خط ۷ با چاپ مقدار متغیر number عدد 15 حاصل می‌شود، چون در خط قبل آن را تغییر داده‌ایم. ولی مقدار همین متغیر در داخل

تابع همان عدد ۱۰ است. نکته آخر این است که در خطی که کلمه global به کار رفته است نمی‌توان عمل انتساب را انجام داد. یعنی خط زیر اشتباه است:

```
global number = 10
```

پارامترهای پیشفرض

پارامترهای پیشفرض همانگونه که از اسمشان پیداست دارای مقادیر پیشفرضی هستند و می‌توان به آنها آرگومان ارسال کرد یا نه. اگر به اینگونه پارامترها، آرگومانی ارسال نشود از مقادیر پیشفرض استفاده می‌کنند. به مثال زیر توجه کنید:

```
1  def printmessage(message = "Welcome to Python Tutorials!"):
2    print(message)
3
4  printmessage()
5  printmessage("Learn Python Today!")
```

```
Welcome to Python Tutorials!
Learn Python Today!
```

تابع printmessage() (خطوط ۲-۱) یک پارامتر اختیاری دارد. برای تعریف یک پارامتر اختیاری می‌توان به آسانی و با استفاده از علامت = یک مقدار را به یک پارامتر اختصاص داد (مثال بالا خط ۱). دو بار تابع را فراخوانی می‌کنیم. در اولین فراخوانی (خط ۴) ما آرگومانی به تابع ارسال نمی‌کنیم بنابراین تابع از مقدار پیشفرض (Welcome to Python Tutorials!) استفاده می‌کند. در دومین فراخوانی (خط ۵) یک پیغام (آرگومان) به تابع ارسال می‌کنیم که جایگزین مقدار پیشفرض پارامتر می‌شود. اگر از چندین پارامتر در تابع استفاده می‌کنید همه پارامترهای اختیاری باید در آخر بقیه پارامترها ذکر شوند. به مثالهای زیر توجه کنید:

```
def somefunction(default1 = 10, default2 = 20, require1, require2) #ERROR

def somefunction(require1, default1 = 10, require2, default2 = 20) #ERROR

def somefunction(require1, require2, default1 = 10, default2 = 20) #Correct
```

وقتی توابع با چندین پارامتر اختیاری فراخوانی می‌شوند باید به پارامترهایی که از لحاظ مکانی در آخر بقیه پارامترها نیستند مقدار اختصاص داد. به یاد داشته باشید که نمی‌توان برای نادیده گرفتن یک پارامتر به صورت زیر عمل کرد:

```
def somefunction(required1, default1 = 10,  default2 = 20):
   #Some Code

somefunction(10, , 100) #Error
```

بازگشت (Recursion)

بازگشت فرایندی است که در آن تابع مدام خود را فراخوانی می‌کند تا زمانی که به یک مقدار مورد نظر برسد. بازگشت یک مبحث پیچیده در برنامه‌نویسی است و تسلط به آن کار راحتی نیست. به این نکته هم توجه کنید که بازگشت باید در یک نقطه متوقف شود وگرنه برای

بی نهایت بار، تابع، خود را فراخوانی می‌کند. در این درس یک مثال ساده از بازگشت را برای شما توضیح می‌دهیم. فاکتوریل یک عدد صحیح مثبت (n!) شامل حاصل ضرب همه اعداد مثبت صحیح کوچکتر یا مساوی آن می‌باشد. به فاکتوریل عدد ۵ توجه کنید.

```
5! = 5 * 4 * 3 * 2 * 1 = 120
```

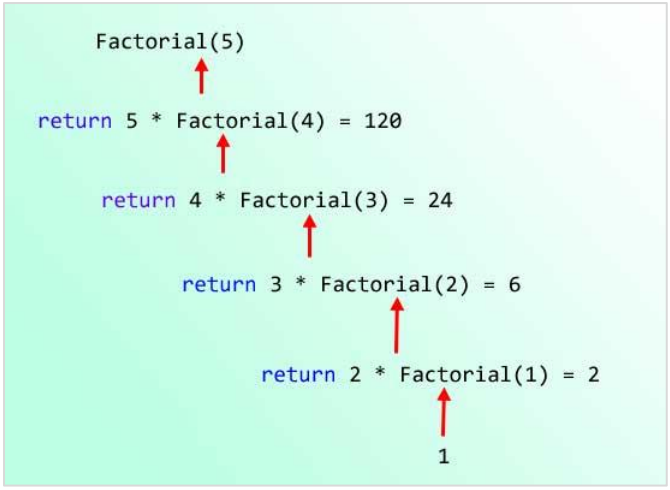
بنابراین برای ساخت یک تابع بازگشتی باید به فکر توقف آن هم باشیم. بر اساس توضیح بازگشت، فاکتوریل فقط برای اعداد مثبت صحیح است. کوچک‌ترین عدد صحیح مثبت ۱ است. در نتیجه از این مقدار برای متوقف کردن بازگشت استفاده می‌کنیم.

```
1  def factorial(number):
2    if (number == 1):
3      return 1
4    return number * Factorial(number - 1)
5
6  print(factorial(5))
```

```
120
```

تابع مقدار بزرگی را بر می‌گرداند چون محاسبه فاکتوریل می‌تواند خیلی بزرگ باشد. تابع یک آرگومان که یک عدد است و می‌تواند در محاسبه مورد استفاده قرار گیرد را می‌پذیرد. در داخل تابع یک دستور if می‌نویسیم و در خط ۲ می گوییم که اگر آرگومان ارسال شده برابر ۱ باشد سپس مقدار ۱ را برگردان در غیر اینصورت به خط بعد برو. این شرط باعث توقف تکرارها نیز می‌شود.

در خط ۴ مقدار جاری متغیر number در عددی یک واحد کمتر از خودش (number - 1) ضرب می‌شود. در این خط تابع Factorial خود را فراخوانی می‌کند و آرگومان آن در این خط همان number - 1 است. مثلاً اگر مقدار جاری number 10 باشد یعنی اگر ما بخواهیم فاکتوریل عدد ۱۰ را به دست بیاوریم آرگومان تابع Factorial در اولین ضرب ۹ خواهد بود. فرایند ضرب تا زمانی ادامه می‌یابد که آرگومان ارسال شده با عدد ۱ برابر نشود. شکل زیر فاکتوریل عدد ۵ را نشان می‌دهد.



کد بالا را به وسیله یک حلقه while نیز می‌توان نوشت.

```
num = 5
factorial = 1
```

```
while num > 1:
  factorial = factorial * num
  num = num - 1

print(factorial)
```

این کد از کد معادل بازگشتی آن آسان‌تر است. از بازگشت در زمینه‌های خاصی در علوم کامپیوتر استفاده می‌شود. استفاده از بازگشت حافظه زیادی اشغال می‌کند پس اگر سرعت برای شما مهم است از آن استفاده نکنید.

توابع داخلی

به تابعی که در داخل تابع دیگر تعریف شده باشد، تابع داخلی گفته می‌شود:

```
def outer():
    def inner():
        #Some code
```

در کد بالا تابع inner() یک تابع داخلی است. برای اینکه یک تابع داخلی را به وسیله تابع خارجی آن برگشت دهیم باید بعد از کلمه کلیدی return، نام تابع داخلی را بدون پرانتز بنویسیم:

```
def outer():
    def inner():
        #Some code

    return inner
```

به مثالی ساده در مورد توابع داخلی توجه کنید:

```
1  def make_adder(x):
2      def addfive():
3          return x + 5
4      return addfive
5
6  result = make_adder(10)
7  print(result())
```

```
15
```

قبل از توضیح کد بالا به این نکته توجه کنید که توابع داخلی به متغیرهای تابع خارجی دسترسی دارد. در کد بالا یک تابع داخلی به نام addfive() تعریف شده است که وظیفه آن اضافه کردن عدد ۵ به مقداری است که با تابع خارجی ارسال می‌شود. در خط ۶ وقتی تابع make_adder() را فراخوانی می‌کنیم و مقدار آن را در داخل یک متغیر به نام result می‌ریزیم، این متغیر در اصل مقدار برگشتی از تابع make_adder() یعنی تابع addfive() است. در نتیجه برای چاپ نهایی این مقدار برگشتی باید در جلوی نام result در خط ۷ علامت پرانتز بگذاریم تا به نوعی تابع addfive() فراخوانی شود. پس وقتی مقدار ۱۰ را به تابع make_adder() ارسال می‌کنیم، این عدد در داخل تابع addfive() با ۵ جمع شده و مقدار ۱۵ برگشت داده می‌شود. توابع داخلی در مبحث Decorator ها کاربرد دارند، که در درس‌های آینده توضیح می‌دهیم.

Decorator

در درس‌های قبلی در مورد ارسال مقدار به تابع، ارسال تابع به تابع و همچنین نحوه ایجاد توابع داخلی توضیح دادیم. حال می‌خواهیم این سه مبحث را در این درس و در قالب مبحث جدیدی به نام Decorator به کار ببریم. فرض کنید می‌خواهیم یک تابع را ایجاد کنیم که هر عددی که به آن دادیم را به توان ۲ برساند و نتیجه را برگشت دهد. روش تعریف همچین تابعی به صورت زیر است:

```
def powevenvalue(number):
    return number * number
```

اما اگر بخواهیم فقط اعداد زوج را به توان دو برساند به دو روش می‌توانیم این کار را انجام دهیم. یا باید بدنه تابع را دستکاری کنیم:

```
def powevenvalue(number):
    if ((number % 2) == 0):
        return number * number
    else:
       number
```

و اگر نخواهید بدنه تابع را دستکاری کنید می‌توانید از روش دوم استفاده کنید و آن استفاده از یک تابع داخلی است. بدین صورت که شما هر تغییری را که می‌خواهید در تابع اصلی اعمال کنید به یک تابع داخلی می‌دهید. کد بالا را به صورت زیر اصلاح می‌کنیم:

```
1  def getfunction(function):
2
3      def checkvalue(num):
4          if ((num % 2) == 0):
5              return function(num)
6          else:
7              return num
8
9      return checkvalue
10
11 def powevenvalue(number):
12     return number * number
13
14
15 result = getfunction(powevenvalue)
16
17 print(result(10))
```

```
100
```

در کد بالا با تابع اصلی کاری نداریم (خطوط ۱۲-۱۱). یک تابع تعریف می‌کنیم، که یک تابع دریافت می‌کند (خطوط ۹-۱). در داخل این تابع یک تابع دیگر تعریف می‌کنیم (خطوط ۷-۳)، که همان تابع داخلی بوده و همان تغییراتی را که قرار است در تابع اصلی بدهیم، از این تابع می‌خواهیم. این تابع یک پارامتر قبول می‌کند که همان عددی است که قرار است به توان برسد. در داخل بدنه این تابع در خط ۴ چک می کنیم که اگر باقیمانده تقسیم عدد گرفته شده بر ۲ برابر با ۰ بود، عدد را به تابع اصلی بدهد تا آن را به توان ۲ برساند در غیر اینصورت خود عدد را نشان دهد (خط ۷). در خط ۹ هم همین تابع داخلی را به وسیله تابع خارجی برگشت می‌دهیم.

در خط ۱۵، تابع خارجی را صدا می‌زنیم. این تابع، یک تابع را به عنوان پارامتر دریافت می‌کند، و ما هم تابع اصلی (powevenvalue()) را به آن ارسال می‌کنیم و نتیجه را در یک متغیر با نام result می‌ریزیم. از آنجاییکه خروجی تابع خارجی یعنی تابع getfunction()، تابع داخلی checkvalue() است، در نتیجه، متغیر result همان تابع checkvalue() است و کافیست در خط ۱۷ یک پرانتزها را در جلوی این متغیر قرار داده و یک عدد به آن بدهیم تا چک کند که آیا زوج است یا فرد؟

اگر بخواهید همان تابع اصلی خود را صدا زده و خروجی مورد نظر را دریافت کنید، کافیست که از مفهوم Decorator استفاده کنید. خطوط ۱۷-۱۳ کد بالا را حذف کرده و خط ۱۱ کد زیر را قبل از تابع اصلی بنویسید. خط ۱۱ همان کار خطوط ۱۵ و ۱۷ کد بالا را انجام می‌دهد و مفهوم Decorator هم همین است. در این خط علامت @ و سپس نام تابع خارجی را نوشته‌ایم:

```
1  def getfunction(function):
2
3      def checkvalue(num):
4          if ((num % 2) == 0):
5              return function(num)
6          else:
7              return num
8
9      return checkvalue
10
11 @getfunction
12 def powevenvalue(number):
13     return number * number
14
15 powevenvalue(10):
```

```
100
```

همانطور که در کد بالا مشاهده می‌کنید می‌توان تابع اصلی را صدا کرد و یک عدد به آن داد، تا مجذورش را محاسبه کند.

عبارات لامبدا (Lambda expressions)

عبارات لامبدا در اصل توابع یک خطی هستند که در برخی از زبان‌ها به عنوان توابع بی نام شناخته می‌شوند. گاهی اوقات اتفاق می افتد که در برنامه نمی‌خواهید یک تابع را جهت انجام یک کار تعریف کنید. در این صورت می‌توان از عبارات لامبدا استفاده کرد. نحوه استفاده از لامبدا به صورت زیر است:

```
lambda parameter(s): manipulate(parameter)
```

به کد زیر توجه کنید:

```
showmessage  = lambda message: print(message)

showmessage ("Hello World!")
```

```
Hello World!
```

لامبدا می‌تواند، هیچ پارامتری نگیرد:

```
showmessage  = lambda : print("Hello World!")

showmessage ()
```

```
Hello World!
```

هنگام فراخوانی تابع باید تعداد آرگومان‌ها با تعداد پارامترها برابر باشد. مثلاً برنامه زیر با خطا مواجه می‌شود:

```
showmessage  = lambda message1, message2: print(message1, message2)

showmessage ("Hello World!")
```

اگر یک عبارت لامبدا دارای دو یا تعداد بیشتری پارامتر باشد باید آنها را در داخل پرانتز قرار دهید:

```
mutilParameters= lambda param1, param2 : (param1, param2)
```

به مثال زیر توجه کنید:

```
showmessage  = lambda message: (print(message), print("Some more message"))

showmessage ("Hello World!")
```

```
Hello World!
Some more message
```

عبارات لامبدا نمی‌توانند دارای کلمه `return` باشند. می‌توان گفت که دستورات لامبدا در حالت عادی برگردانده می‌شوند و نیازی به این کلمه نیست:

```
GetSquare = lambda number : number * number
print(GetSquare(5))
```

```
25
```

توابع از پیش تعریف شده (Built-in Function)

در درس‌های قبل در مورد چگونگی تعریف تابع و ارسال آرگومان به آن و ... بحث کردیم. پایتون علاوه بر توابعی که توسط کاربر تعریف می شوند دارای توابع دیگری نیز هست که به آنها توابع از پیش تعریف شده می گویند. طراحان زبان برنامه‌نویسی پایتون برای سادگی کار برنامه نویسان، این توابع را نوشته و به همراه پایتون ارائه می‌دهند. تعداد این توابع به همراه نسخه‌های جدید پایتون افزایش می‌یابد. این توابع دارای کابردهای مختلفی از جمله برگرداندن قدر مطلق یک عدد، تبدیل انواع داده به هم، ایجاد لیست و ... به کار می‌روند. در نسخه ۳٫۶ پایتون که آخرین نسخه این زبان تا کنون است، ۶۸ تابع از پیش تعریف شده وجود دارد که در جدول زیر به برخی از آنها اشاره شده است:

تابع	کاربرد

`abs()`	قدر مطلق یک عدد را بر می‌گرداند.
`bin()`	یک مقدار صحیح را به یک رشته باینری تبدیل می‌کند.
`chr()`	معادل کاراکتر یا رشته‌ای یک عدد را بر می‌گرداند.
`float()`	مقدرا یک رشته یا عدد صحیح را به نوع اعشاری تبدیل می‌کند.
`format()`	یک مقدار را قالب بندی می‌کند. این تابع بیشتر به همراه تابع `print()` جهت قالب بندی خروجی به کار می‌رود.
`input()`	برای خواندن و برگرداندن یک خط رشته به کار می‌رود.
`int()`	یک مقدار را به نوع صحیح تبدیل می‌کند.
`list()`	برای ایجاد یک لیست به کار می‌رود.
`pow()`	یک عدد را به توان عدد دیگر می‌رساند.
`print()`	یک مفدار را چاپ می‌کند.
`range()`	یک محدود از اعداد صحیح ایجاد می‌کند. مثلاً اعداد بین ۱ تا ۱۰.
`type()`	نوع یک مقدار را بر می‌گرداند. مثلاً اگر عدد `10.2` را به این تابع بدهیم، عبارت `float` را بر می‌گرداند که نشان دهنده نوع عدد است.

در مثال زیر هم در عمل نحوه کار با این توابع نشان داده شده است:

```
print(abs(-3))
print(bin(5))
print(chr(97))
print(float(10))
print(pow(2, 2))
```

```
3
0b101
a
10.0
4
```

همانطور که احتمالاً از خروجی کدهای بالا متوجه شده‌اید، مثلاً تابع pow() دو آرگومان می‌گیرد که اولی عدد و دومی توان می‌باشد. در مثلاً بالا عدد ۲ را به توان دو رسانده‌ایم، که در خروجی عدد ۴ نشان دهده شده است. یا مثلاً تابع abs() قدر مطلق ۳- را که عدد ۳ می باشد را برگشت داده است. تابع از پیش تعریف شده range() یک محدوده از اعداد را ایجاد می‌کند. به کد زیر توجه کنید:

```
for number in (1, 2, 3, 4, 5):
    print(number)
```

در کد بالا مقادیر یک مجموعه از اعداد را چاپ کرده‌ایم. برای ایجاد همین محدوده از اعداد با استفاده از تابع range() می‌توان به صورت زیر عمل کرد:

```
for number in range(1,6):
    print(number)
```

خروجی دو کد بالا، شبیه هم می‌باشد. ممکن است که این سؤال برایتان پیش بیاید که چرا در کد بالا ۶ را در داخل تابع نوشته‌ایم. آرگومان دوم تابع range() جز خروجی نست. یعنی اعداد ۱ تا ۵ چاپ می‌شوند. اگر به جای ۶ عدد ۱۰ را بنویسید. اعداد ۱ تا ۹ چاپ می‌شوند. از تابع type() هم برای تشخیص نوع داده استفاده می‌شود:

```
intVar        = 10
floatVar      = 12.5
boolVar       = True
StringVar     = "Hello World!"
listVar       = [1,5,8]
tupleVar      = ("Python","Programming","begginer")
dictionaryVar = {'Name': 'jack', 'family': 'Scalia', 'Age': 7}

print(type(intVar))
print(type(floatVar))
print(type(boolVar))
print(type(StringVar))
print(type(listVar))
print(type(tupleVar))
print(type(dictionaryVar))
```

```
<class 'int'>
<class 'float'>
<class 'bool'>
<class 'str'>
<class 'list'>
<class 'tuple'>
<class 'dict'>
```

لیست کامل توابع از پیش تعریف شده پایتون در لینک زیر آمده است:

```
https://docs.python.org/2/library/functions.html
```

توابعی خاص (Special Methods)

در پایتون توابعی خاصی (Special Methods) وجود دارند که از آنها برای مقاصد متفاوتی می‌توان استفاده کرد. این توابع به صورت توکار همراه با پایتون نصب شده و هنگام برنامه نویسی و در حالت‌های خاصی فراخوانی می‌شوند. با استفاده از توابعی خاص، کلاسی که

به وسیله شما تعریف می‌شود، می‌تواند همانند مجموعه‌ها، دیکشنری‌ها، توابع، پیمایشگرها و حتی همانند اعداد عمل کند. مشخصه اصلی این توابع این است که قبل و بعد از نام آنها، دو علامت زیر خط (__) یا همان underscore قرار می‌گیرد. قبل از آموزش یک مثال ساده می‌زنیم:

```
x = 5

print(dir(x))
```

```
['__abs__',  '__add__',  '__and__',  '__bool__',  '__ceil__',  '__class__',  '__delattr__',
'__dir__',  '__divmod__',  '__doc__',  '__eq__',  '__float__',  '__floor__',  '__floordiv__',
'__format__',   '__ge__',   '__getattribute__',   '__getnewargs__',   '__gt__',   '__hash__',
'__index__',   '__init__',   '__init_subclass__',   '__int__',   '__invert__',   '__le__',
'__lshift__',  '__lt__',  '__mod__',  '__mul__',  '__ne__',  '__neg__',  '__new__',  '__or__',
'__pos__',  '__pow__',  '__radd__',  '__rand__',  '__rdivmod__',  '__reduce__',  '__reduce_ex__',
'__repr__',  '__rfloordiv__',  '__rlshift__',  '__rmod__',  '__rmul__',  '__ror__',  '__round__',
'__rpow__',   '__rrshift__',   '__rshift__',   '__rsub__',   '__rtruediv__',   '__rxor__',
'__setattr__',  '__sizeof__',  '__str__',  '__sub__',  '__subclasshook__',  '__truediv__',
'__trunc__',  '__xor__',  'bit_length',  'conjugate',  'denominator',  'from_bytes',  'imag',
'numerator', 'real', 'to_bytes']
```

در مثال بالا یک متغیر از نوع کلاس int تعریف کرده و سپس با استفاده از تابع dir()، لیست توابع و خاصیت‌های آن را به دست آورده‌ایم. اکثر اسامی بالا، نام توابعی کلاس int بوده و اکثر آنها به عملگرهای پایتون مرتبط هستند. یکی از آنها متد __add__() می باشد. همانطور که می دانید برای جمع دو عدد از علامت + استفاده می‌شود:

```
x = 5
print(x + 10)
```

```
15
```

اما چیزی که در پشت صحنه و به صورت خودکار توسط پایتون اتفاق می افتد فراخوانی متد خاص __add__() برای جمع این دو مقدار است:

```
x = 5
print(x.__add__(10))
```

```
15
```

کدهای بالا را به صورت زیر هم می‌توان نوشت:

```
x = 5
print(int.__add__(x, 10))
```

```
15
```

این مثال ساده به ما می‌فهماند که وقتی مثلاً ما از عملگر + استفاده می‌کنیم، در اصل پایتون با فراخوانی متد ()__add__ عملیات مورد نظر را انجام می‌دهد. در عبارت x.__add__(10)، پایتون می‌فهمد که x از نوع int است. در نتیجه با فراخوانی متد ()__add__ از کلاس int و ارسال x و عدد ۱۰ به آن عمل جمع را انجام می‌دهد. یعنی در واقع دو عبارت x + 10 و x.__add__(10) توسط مفسر پایتون به int.__add__(x, 10) ترجمه می‌شوند. در پایتون توابعی خاص دیگری وجود دارند که در درس‌های بعدی با برخی از آنها آشنا می‌شویم.