PythonBasics

March 21, 2017

1 Python Basics

This section provides a brief overview of:

- Data types in python
- Basic data structures - lists, tuples, and dictionaries
- Conditional statements and loops

1.1 Data Types and Variables

In this section I am going to focus on integer, floating points, Boolean data types and associated arithmetic operations. 5 is an example of an integer data type whereas 5.0, 6.2 are examples of floating point data types. All standard arithmetic operators are defined in Python.

Note that python also has a string data type which is not covered in this tutorial.

```
In [1]: 3 + 2

Out[1]: 5

In [2]: 3 - 1

Out[2]: 2

In [3]: 3 / 2

Out[3]: 1.5

In [4]: 3 // 2 # Integer division

Out[4]: 1

In [5]: 3 ** 2 # 3 to the power of 2

Out[5]: 9

In [6]: 3 % 2 # This is the remainder operator

Out[6]: 1
```

Python treats every line as a statement to be executed except the line beginning with a #. # indicates the line contains comments to the right of # sign which are not to be executed.

Python has the following comparison operators on all relevant data types: * == equal to * != not equal to * < less than * <= less than or equal to * > greater than * >= greater than or equal to

Boolean is another data type which can take two values – true or false.

```
In [7]: 3 == 2

Out[7]: False

In [8]: 3 != 2

Out[8]: True

In [9]: (5 > 3) and (2 < 1)

Out[9]: False

In [10]: (5 > 3) or (2 < 1)

Out[10]: True
```

In Python, there is no need to declare variables before assigning them.

```
In [11]: x = 3 # Assigns 3 to an integer variable x
         print(x)

3


In [12]: y = 5.2 # Assigns 5.2 to a floating point variable y
         print("y = ", y)

y =  5.2


In [13]: x + y

Out[13]: 8.2

In [14]: float(x) # converts to floating point

Out[14]: 3.0

In [15]: int(y) # converts to integer

Out[15]: 5

In [16]: type(x) #prints the variable type

Out[16]: int
```

1.2 Lists, Tuples, and Dictionaries

You can use lists to store a sequence of values. Note than in Python list indexes start at 0 (similar to C or C++). Different ways of creating lists are shown below

```
In [17]: a = [] # creates an empty list

In [18]: a = [8, 9, 10] # creates a list of three elements

In [19]: a[1] # Prints the second element

Out[19]: 9

In [20]: len(a) #prints the length of the list

Out[20]: 3

In [21]: a.append(20) #Adds 20 to the end of the list
         a

Out[21]: [8, 9, 10, 20]

In [22]: a.pop() # Deletes the last element
         a

Out[22]: [8, 9, 10]

In [23]: a.reverse() # reverses the list
         a

Out[23]: [10, 9, 8]

In [24]: a.sort() #sorts the list
         a

Out[24]: [8, 9, 10]

In [25]: b = [11, 12, 13, 14]

In [26]: a = a + b #lists can be concatenated
         a

Out[26]: [8, 9, 10, 11, 12, 13, 14]

In [27]: c = a[2:5] #creates a sublist from third to fifth element
         c

Out[27]: [10, 11, 12]

In [28]: c.remove(11) # removes 11
         c
```

```
Out[28]: [10, 12]

In [29]: 20 in b # Returns a boolean value for the check if element 20 is in list b

Out[29]: False

In [30]: 11 in b

Out[30]: True
```

list(range(start, stop, stepsize)) can be used to create a list comprising of a sequence of numbers. Note that the number corresponsing to stop is not a part of the list.

```
In [31]: list(range(10,20,2))

Out[31]: [10, 12, 14, 16, 18]
```

Lists can be two dimensional.

```
In [32]: twodlist = [[1, 2, 3],[11, 12, 13],[21, 22, 23]]
         twodlist[1]

Out[32]: [11, 12, 13]

In [33]: twodlist[1][1]

Out[33]: 12
```

Along similar lines we can create, three, or four dimensional lists and so on. You can create lists of non-integer objects also.

```
In [34]: cities= ["Nashville", "Austin", "Pittsburgh"]
         cities

Out[34]: ['Nashville', 'Austin', 'Pittsburgh']
```

Lists can contain objects of different type.

```
In [35]: d = [5, 6, "seven", 8, 9]
         d

Out[35]: [5, 6, 'seven', 8, 9]

In [36]: del d[2] #removes the third element
         d

Out[36]: [5, 6, 8, 9]
```

Tuples are like lists but with two main differences:

- You cannot change or delete elements of a tuple once created. Tuples are immutable objects.
- Tuples use parenthesis whereas lists use square brackets.

Tuples are faster than lists. However, the speed gains might not be significant for our applications. Therefore, stick to lists whenever possible.

```
In [37]: e = (3, 8, 11, 7) # creates a tuple
         e

Out[37]: (3, 8, 11, 7)

In [38]: e[2] # we use square brackets to access elements of tuples

Out[38]: 11
```

Dictionaries can be used to create maps or unordered collection of objects in Python. In dictionaries, keys are associated with values. The keys are separated from their values using colon. Each key-value item is separated using comma.

```
In [39]: pop = {'Nashville': 200, 'Austin': 500, 'Pittsburgh': 300}

In [40]: pop.keys()

Out[40]: dict_keys(['Pittsburgh', 'Nashville', 'Austin'])

In [41]: pop.values()

Out[41]: dict_values([300, 200, 500])
```

The keys and values can be converted to lists using list() function.

```
In [42]: list(pop.keys())

Out[42]: ['Pittsburgh', 'Nashville', 'Austin']

In [43]: pop['Nashville']

Out[43]: 200

In [44]: 'Austin' in pop

Out[44]: True
```

1.3 Conditional Statements and Loops

If loops can be used to execute codes based on conditions.

Note that indentations are important for defining code blocks. Normal indentation is one tab or four whitespaces. The code will not compile properly and give errors if it is not indented properly.

```
In [45]: x = 10
         print(x)
         if x > 5:
             print("x is larger than 5")
         elif x < 5:
             print("x is smaller than 5")
         else:
             print("x is equal to 5")
```

```
10
x is larger than 5
```

While loops execute a section of code until a condition is met. The following code prints 1 through 10.

```
In [46]: y=0
         while y <= x:
             print(y)
             y=y+1

0
1
2
3
4
5
6
7
8
9
10
```

For loops can be used to iterate over elements in lists, tuples, or dictionaries.

```
In [47]: f = [1, 5, 9, 10]
         for i in f:
             print(i*2)

2
10
18
20
```

```
In [48]: for i in range(20, 30):
             if (i % 2 == 0):
                 print(i)

20
22
24
26
28
```

In dictionaries, the default is to iterate through the keys.

```
In [49]: for i in pop:
             print(i)

Pittsburgh
Nashville
Austin


In [50]: for i in pop.values():
             print(i)

300
200
500


In [51]: for i,j in pop.items():
             print(i,j)

Pittsburgh 300
Nashville 200
Austin 500
```

For loops and conditional expressions can be used to create lists conveniently.s

```
In [52]: [i for i in range(3)]

Out[52]: [0, 1, 2]

In [53]: [i*2 for i in [10, 20, 25]]

Out[53]: [20, 40, 50]

In [54]: [[i,j] for i in [1, 2, 3] for j in [10, 11, 12]]

Out[54]: [[1, 10],
          [1, 11],
          [1, 12],
          [2, 10],
          [2, 11],
          [2, 12],
          [3, 10],
          [3, 11],
          [3, 12]]
```