Module 1: Introduction to Python

Python is a high-level programming language with extensive libraries available to perform various data analysis tasks. The following tutorial contains examples of using various data types, functions, and library modules available in the standard Python library. The notebook can be downloaded from http://www.cse.msu.edu/~ptan/dmbook/tutorials/tutorial1/tutorial1.ipynb (http://www.cse.msu.edu/~ptan/dmbook/tutorials/tutorial1/tutorial1.ipynb). Read the step-by-step instructions below carefully. To execute the code, click on each cell below and press the SHIFT-ENTER keys simultaneously.

We begin with some basic information about Python:

1. Python is an interpreted language, unlike other high-level programming languages such as C or C++. You only need to submit your Python program to an interpreter for execution, without having to explicitly compile and link the code first.
2. Python is a dynamically typed language, which means variable names are bound to their respective types during execution time. You do not have to explicitly declare the type of a variable before using it in the code unlike Java, C++, and other statically-typed languages.
3. Instead of using braces '{' and '}', Python uses whitespace indentation to group together related statements in loops or other control-flow statements.
4. Python uses the hash character ('#') to precede single-line comments. Triple-quoted strings (''') are commonly used to denote multi-line comments (even though it is not part of the standard Python language) or docstring of functions.
5. Python uses pass by reference (instead of pass by value) when assigning a variable to another (e.g., a = b) or when passing an object as input argument to a function. Thus, any modification to the assigned variable or to the input argument within the function will affect the original object.
6. Python uses `None` to denote a null object (e.g., a = None). You do not have to terminate each statement with a terminating character (such as a semicolon) unlike other languages.
7. You may access the variables or functions defined in another Python program file using the `import` command. This is analogous to the `import` command in Java or the `#include` command in C or C++.

1.1 Elementary Data Types

The standard Python library provides support for various elementary data types, including including integers, booleans, floating points, and strings. A summary of the data types is shown in the table below.

	Data Type	Example
Number	Integer	x = 4
	Long integer	x = 15L
	Floating point	x = 3.142
	Boolean	x = True
Text	Character	x = 'c'
	String	x = "this" or x = 'this'

In [1]:
```
x = 4               # integer
print(x, type(x))

y = True            # boolean (True, False)
print(y, type(y))

z = 3.7             # floating point
print(z, type(z))

s = "This is a string"    # string
print(s, type(s))
```

```
4 <class 'int'>
True <class 'bool'>
3.7 <class 'float'>
This is a string <class 'str'>
```

The following are some of the arithmetic operations available for manipulating integers and floating point numbers

In [2]:
```
x = 4            # integer
x1 = x + 4       # addition
x2 = x * 3       # multiplication
x += 2           # equivalent to x = x + 2
x3 = x
x *= 3           # equivalent to x = x * 3
x4 = x
x5 = x % 4       # modulo (remainder) operator

z = 3.7          # floating point number
z1 = z - 2       # subtraction
z2 = z / 3       # division
z3 = z // 3      # integer division
z4 = z ** 2      # square of z
z5 = z4 ** 0.5   # square root
z6 = pow(z,2)    # equivalent to square of z
z7 = round(z)    # rounding z to its nearest integer
z8 = int(z)      # type casting float to int

print(x,x1,x2,x3,x4,x5)
print(z,z1,z2,z3,z4)
print(z5,z6,z7,z8)
```

```
18 8 12 6 18 2
3.7 1.7000000000000002 1.2333333333333334 1.0 13.690000000000001
3.7 13.690000000000001 4 3
```

The following are some of the functions provided by the math module for integers and floating point numbers

In [3]:
```
import math

x = 4
print(math.sqrt(x))      # sqrt(4) = 2
print(math.pow(x,2))     # 4**2 = 16
print(math.exp(x))       # exp(4) = 54.6
print(math.log(x,2))     # log based 2  (default is natural logarithm)
print(math.fabs(-4))     # absolute value
print(math.factorial(x)) # 4! = 4 x 3 x 2 x 1 = 24

z = 0.2
print(math.ceil(z))      # ceiling function
print(math.floor(z))     # floor function
print(math.trunc(z))     # truncate function

z = 3*math.pi            # math.pi = 3.141592653589793
print(math.sin(z))       # sine function
print(math.tanh(z))      # arctan function

x = math.nan             # not a number
print(math.isnan(x))

x = math.inf             # infinity
print(math.isinf(x))
```

```
2.0
16.0
54.598150033144236
2.0
4.0
24
1
0
0
3.6739403974420594e-16
0.9999999869751758
True
True
```

The following are some of the logical operations available for booleans

In [4]:
```
y1 = True
y2 = False

print(y1 and y2)       # logical AND
print(y1 or y2)        # logical OR
print(y1 and not y2)   # logical NOT
```

```
False
True
True
```

The following are some of the operations and functions for manipulating strings

In [5]:
```
s1 = "This"

print(s1[1:])                    # print last three characters
print(len(s1))                               # get the string length
print("Length of string is " + str(len(s1))) # type casting int to str
print(s1.upper())                            # convert to upper case
print(s1.lower())                            # convert to lower case

s2 = "This is a string"
words = s2.split(' ')             # split the string into words
print(words[0])
print(s2.replace('a','another'))  # replace "a" with "another"
print(s2.replace('is','at'))      # replace "is" with "at"
print(s2.find("a"))               # find the position of "a" in s2
print(s1 in s2)                   # check if s1 is a substring of s2

print(s1 == 'This')               # equality comparison
print(s1 < 'That')                # inequality comparison
print(s2 + " too")                # string concatenation
print((s1 + " ")* 3)              # replicate the string 3 times
```

```
his
4
Length of string is 4
THIS
this
This
This is another string
That at a string
8
True
True
False
This is a string too
This This This
```

1.2 Compound Data Types

The following examples show how to create and manipulate a list object

In [6]:
```
intlist = [1, 3, 5, 7, 9]
print(type(intlist))
print(intlist)
intlist2 = list(range(0,10,2))   # range[startvalue, endvalue, stepsize]
print(intlist2)

print(intlist[2])                # get the third element of the list
print(intlist[:2])               # get the first two elements
print(intlist[2:])               # get the last three elements of the list
print(len(intlist))              # get the number of elements in the list
print(sum(intlist))              # sums up elements of the list

intlist.append(11)               # insert 11 to end of the list
print(intlist)
print(intlist.pop())             # remove last element of the list
print(intlist)
print(intlist + [11,13,15])      # concatenate two lists
print(intlist * 3)               # replicate the list
intlist.insert(2,4)              # insert item 4 at index 2
print(intlist)
intlist.sort(reverse=True)       # sort elements in descending order
print(intlist)
```

```
<class 'list'>
[1, 3, 5, 7, 9]
[0, 2, 4, 6, 8]
5
[1, 3]
[5, 7, 9]
5
25
[1, 3, 5, 7, 9, 11]
11
[1, 3, 5, 7, 9]
[1, 3, 5, 7, 9, 11, 13, 15]
[1, 3, 5, 7, 9, 1, 3, 5, 7, 9, 1, 3, 5, 7, 9]
[1, 3, 4, 5, 7, 9]
[9, 7, 5, 4, 3, 1]
```

In [7]:
```
mylist = ['this', 'is', 'a', 'list']
print(mylist)
print(type(mylist))

print("list" in mylist)          # check whether "list" is in mylist
print(mylist[2])                 # show the 3rd element of the list
print(mylist[:2])                # show the first two elements of the list
print(mylist[2:])                # show the last two elements of the list
mylist.append("too")             # insert element to end of the list

separator = " "
print(separator.join(mylist))    # merge all elements of the list into a strin
g

mylist.remove("is")              # remove element from list
print(mylist)
```

```
['this', 'is', 'a', 'list']
<class 'list'>
True
a
['this', 'is']
['a', 'list']
this is a list too
['this', 'a', 'list', 'too']
```

The following examples show how to create and manipulate a dictionary object

In [8]:
```
abbrev = {}
abbrev['MI'] = "Michigan"
abbrev['MN'] = "Minnesota"
abbrev['TX'] = "Texas"
abbrev['CA'] = "California"

print(abbrev)
print(abbrev.keys())            # get the keys of the dictionary
print(abbrev.values())          # get the values of the dictionary
print(len(abbrev))              # get number of key-value pairs

print(abbrev.get('MI'))
print("FL" in abbrev)
print("CA" in abbrev)

keys = ['apples', 'oranges', 'bananas', 'cherries']
values = [3, 4, 2, 10]
fruits = dict(zip(keys, values))
print(fruits)
print(sorted(fruits))     # sort keys of dictionary

from operator import itemgetter
print(sorted(fruits.items(), key=itemgetter(0)))    # sort by key of dictionar
y
print(sorted(fruits.items(), key=itemgetter(1)))    # sort by value of diction
ary
```

```
{'MI': 'Michigan', 'MN': 'Minnesota', 'TX': 'Texas', 'CA': 'California'}
dict_keys(['MI', 'MN', 'TX', 'CA'])
dict_values(['Michigan', 'Minnesota', 'Texas', 'California'])
4
Michigan
False
True
{'apples': 3, 'oranges': 4, 'bananas': 2, 'cherries': 10}
['apples', 'bananas', 'cherries', 'oranges']
[('apples', 3), ('bananas', 2), ('cherries', 10), ('oranges', 4)]
[('bananas', 2), ('apples', 3), ('oranges', 4), ('cherries', 10)]
```

The following examples show how to create and manipulate a tuple object. Unlike a list, a tuple object is immutable, i.e., they cannot be modified after creation.

In [9]:
```
MItuple = ('MI', 'Michigan', 'Lansing')
CAtuple = ('CA', 'California', 'Sacramento')
TXtuple = ('TX', 'Texas', 'Austin')

print(MItuple)
print(MItuple[1:])

states = [MItuple, CAtuple, TXtuple]    # this will create a list of tuples
print(states)
print(states[2])
print(states[2][:])
print(states[2][1:])

states.sort(key=lambda state: state[2])  # sort the states by their capital ci
ties
print(states)
```

```
('MI', 'Michigan', 'Lansing')
('Michigan', 'Lansing')
[('MI', 'Michigan', 'Lansing'), ('CA', 'California', 'Sacramento'), ('TX', 'T
exas', 'Austin')]
('TX', 'Texas', 'Austin')
('TX', 'Texas', 'Austin')
('Texas', 'Austin')
[('TX', 'Texas', 'Austin'), ('MI', 'Michigan', 'Lansing'), ('CA', 'Californi
a', 'Sacramento')]
```

1.3 Control Flow Statements

Similar to other programming languages, the control flow statements in Python include if, for, and while statements. Examples on how to use these statements are shown below.

In [10]:
```
# using if-else statement

x = 10

if x % 2 == 0:
    print("x =", x, "is even")
else:
    print("x =", x, "is odd")

if x > 0:
    print("x =", x, "is positive")
elif x < 0:
    print("x =", x, "is negative")
else:
    print("x =", x, "is neither positive nor negative")
```

```
x = 10 is even
x = 10 is positive
```

In [11]:
```
# using for loop with a list

mylist = ['this', 'is', 'a', 'list']
for word in mylist:
    print(word.replace("is", "at"))

mylist2 = [len(word) for word in mylist]   # number of characters in each word
print(mylist2)

# using for loop with list of tuples

states = [('MI', 'Michigan', 'Lansing'),('CA', 'California', 'Sacramento'),
          ('TX', 'Texas', 'Austin')]

sorted_capitals = [state[2] for state in states]
sorted_capitals.sort()
print(sorted_capitals)

# using for loop with dictionary

fruits = {'apples': 3, 'oranges': 4, 'bananas': 2, 'cherries': 10}
fruitnames = [k for (k,v) in fruits.items()]
print(fruitnames)
```

```
that
at
a
latt
[4, 2, 1, 4]
['Austin', 'Lansing', 'Sacramento']
['apples', 'oranges', 'bananas', 'cherries']
```

In [12]:
```
# using while loop

mylist = list(range(-10,10))
print(mylist)

i = 0
while (mylist[i] < 0):
    i = i + 1

print("First non-negative number:", mylist[i])
```

```
[-10, -9, -8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
First non-negative number: 0
```

1.4 User-Defined Functions

You can create your own functions in Python, which can be named or unnamed. Unnamed functions are defined using the lambda keyword as shown in the previous example for sorting a list of tuples.

In [13]:
```
myfunc = lambda x: 3*x**2 - 2*x + 3      # example of an unnamed quadratic f
ction

print(myfunc(2))
```

```
11
```

In [14]:
```
import math

# The following function will discard missing values from a list
def discard(inlist, sortFlag=False):    # default value for sortFlag is False
    outlist = []
    for item in inlist:
        if not math.isnan(item):
            outlist.append(item)

    if sortFlag:
        outlist.sort()
    return outlist

mylist = [12, math.nan, 23, -11, 45, math.nan, 71]

print(discard(mylist,True))
```

```
[-11, 12, 23, 45, 71]
```

1.5 File I/O

You can read and write data from a list or other objects to a file.

In [15]:
```
states = [('MI', 'Michigan', 'Lansing'),('CA', 'California', 'Sacramento'),
          ('TX', 'Texas', 'Austin'), ('MN', 'Minnesota', 'St Paul')]

with open('states.txt', 'w') as f:
    f.write('\n'.join('%s,%s,%s' % state for state in states))

with open('states.txt', 'r') as f:
    for line in f:
        fields = line.split(sep=',')    # split each line into its respective
 fields
        print('State=',fields[1],'(',fields[0],')','Capital:', fields[2])
```

```
State= Michigan ( MI ) Capital: Lansing

State= California ( CA ) Capital: Sacramento

State= Texas ( TX ) Capital: Austin

State= Minnesota ( MN ) Capital: St Paul
```