Functions

Functions

- Functions in Python
  - Functions vs Methods
  - Parameters vs Arguments
- How To Define a Function: User-Defined Functions (UDFs)
  - The return statement
  - How To Call a Function
  - How To Add Docstrings to a Python Function
  - Function Arguments in Python
  - Global vs Local Variables
- Anonymous Functions in Python
- Using main() as a Function

Functions

- You use functions in programming to bundle a set of instructions that you want to use repeatedly or
- Because of their complexity, it is better to have a self-contained block of code also called as a sub-program which can be called when needed.
- That means that a function is a block of code written to carry out a specified task.
- To carry out that specific task, the function might or might not need multiple inputs.
- When the task is carried out, the function can or can not return one or more values.

Functions

- There are three types of functions in Python:

1. Built-in functions, such as
   1. help() to ask for help,
   2. min() to get the minimum value,
   3. print() to print an object to the terminal.
2. User-Defined Functions (UDFs), which are functions that users create to perform a specific task.
3. Anonymous functions, which are also called lambda functions because they are not declared with the standard def keyword.

Defining a function

- Function blocks begin with the keyword **def** followed by the function name and parentheses :   ( )
- Any input parameters or arguments should be placed within these parentheses.
- The code block within every function starts with a colon (:) and is indented.
- The statement return [expression] exits a function, optionally passing back an expression to the caller.
- A return statement with no arguments is the same as return None.

Syntax

```
def functionname( parameters ):
      function_docstring
      function_suite
      return [expression]
```

Without arguments and no return type

'''Program having a function to display “Sum:” with no arguments without return type'''

```
def process():
	print("Sum:")
process()
```

Function definition

Function call

```
Sum:15
```

With arguments and no return type

```
'''Program to find sum of 5 numbers (use
arguments without return type)'''
def process(a,b,c,d,e):
    s= a+b+c+d+e
    print("Sum:",s)
x=process(3,3,3,3,3)
```

```
Sum:15
```

With arguments and no return type

```
'''Program to find sum of 5 numbers (use
arguments without return type)'''
def process(a,b,c,d,e):
    s= a+b+c+d+e
    print("Sum:",s)
x=process(3,3,3,3,3)
print(x)
```

```
Sum:15
None
```

With arguments with return type

```
'''Program to find sum of 5 numbers (use
arguments with return type)'''

def process(a,b,c,d,e):
    s= a+b+c+d+e
    return s
x=process(3,3,3,3,3)
print(x)
```

Sum:15

Example

```
import math
def process(a,b,c,d,e) :
    aa=math.sin(a)
    bb=math.cos(b)
    cc=math.ceil(c)
    dd=math.floor(d)
    ee=math.sqrt(e)
    print("sin:",aa)
    print("cos:",bb)
    print("ceil:",cc)
    print("floor:",dd)
    print("Square root:“,ee)
x=process(30,45,60.01,60.999,625)
```

```
sin: -0.9880316240928618
cos: 0.5253219888177297
ceil: 61
floor: 60
Square root: 25.0
```

Ex 2

```
(-0.9880316240928618, 0.5253219888177297, 61, 60, 25.0)
```

```
import math
def process(a,b,c,d,e) :
    aa=math.sin(a)
    bb=math.cos(b)
    cc=math.ceil(c)
    dd=math.floor(d)
    ee=math.sqrt(e)
    return aa,bb,cc,dd,ee
x=process(30,45,60.01,60.1,625)
print(x)
#Returns a tuple
```

Python - Tuple

- A tuple is a sequence of immutable Python objects.
- Tuples are sequences, just like lists.
- The differences between tuples and lists are, the tuples cannot be changed unlike lists and
- tuples use parentheses, whereas lists use square brackets.
- Creating a tuple is as simple as putting different comma-separated values. Optionally you can put these comma-separated values between parentheses also. For example –
- tup1 = ('physics', 'chemistry', 1997, 2000);
- tup2 = (1, 2, 3, 4, 5);
- tup3 = "a", "b", "c", "d";

Python - Tuple

- The empty tuple is written as two parentheses containing nothing –
- tup1 = ()
- To write a tuple containing a single value you have to include a comma, even though there is only one value –
- tup1 = (50**,**)
- Like string indices, tuple indices start at 0, and they can be sliced, concatenated, and so on.

Python - Tuple

- Accessing Values in Tuples:
- To access values in tuple, use the square brackets for slicing along with the index or indices to obtain value available at that index. For example –
- #!/usr/bin/python
- tup1 = ('physics', 'chemistry', 1997, 2000);
- tup2 = (1, 2, 3, 4, 5, 6, 7);
- print ("tup1[0]: ", tup1[0])
- print ("tup2[1:5]: ", tup2[1:5])

Python - Tuple

- >>> tup = (1,2)
- >>> tup1 = (5,6)
- >>> tup3 = tup + tup1
- >>> print(tup3)
- (1, 2, 5, 6)
- >>> tup2 = 100,200
- >>> print(tup2)
- (100, 200)
- >>> tup4 = 'a','b'
- >>> print(tup4)
- ('a', 'b')
- >>> tup5 = "A" , "B"
- >>> print(tup5)
- ('A', 'B')

Python - Tuple

- Updating Tuples
- Tuples are immutable which means you cannot update or change the values of tuple elements. You are able to take portions of existing tuples to create new tuples as the following example demonstrates –
- tup1 = (12, 34.56)
- tup2 = ('abc', 'xyz')
- # Following action is not valid for tuples
- # tup1[0] = 100
- # So let's create a new tuple as follows
- tup3 = tup1 + tup2
- print (tup3)
- (12, 34.56, 'abc', 'xyz')

Example 3

```
import math

def process(a,b) :
    aa=math.sin(a)
    bb=math.cos(b)
    cc=math.ceil(aa)
    dd=math.floor(bb)
    print("sin:",aa)
    print("cos:",bb)
    return cc,dd

x=process(90,120)
print(x)
```

```
sin: 0.8939966636005579
cos: 0.8141809705265618
(1, 0)
```

Pass by reference

- Pass by reference vs value
- Parameters (arguments) in the Python language can be passed by reference.
- It means if you change what a parameter refers to within a function, the change also reflects back in the calling function.
- For example –

Pass by reference

- # Function definition is here
- def changeme(mylist):
- # "This changes a passed list into this function"
-     mylist.append(1);
-     print ("Values inside the function: ", mylist)
-     return
- # Now you can call changeme function
- mylist = [10,20,30];
- changeme(mylist);
- print ("Values outside the function: ", mylist)