Module 05. Functions

September 26, 2018

1 Functions

1. Piece of reusable code
2. Solves particular task
3. Call function instead of writing code yourself

1.1 Python in-built functions

type()
print()
sum()
max()
len()

help() –> Opens up documentation

```
In [ ]: help(round)
```

```
In [ ]: help(print)
```

1.2 Methods: Functions that belong to objects

```
In [ ]: name = "Jane" # --> str object

        name.upper()
```

```
In [ ]: height = 1.73 # --> float object

        height.is_integer()
```

```
In [ ]: l = ["a","'b","c"] # --> list object

        l.index("a")
```

Everything = object Object have methods associated, depending on type.

```
In [ ]: fam = ["liz", 1.73, "emma", 1.68, "mom", 1.71, "dad", 1.89]
        sister = "liz"
```

```
In [ ]: sister.replace("z", "sa")
```

```
In [ ]: fam.replace("mom", "mommy")
```

```
In [ ]: sister.index("z")
```

```
In [ ]: fam.index("mom")
```

1.3 Defining Functions

```
In [ ]: def func():
            print("Im in a function")

        func()
```

1.3.1 Passing parameters to functions

Pass By Value

```
In [ ]: def func(var):
            var = [1, 2, 3, 4]   # Assigning new list object to var
            print("Values inside the function: ", var)

        mylist = [10, 20, 30]
        func(mylist)
        print("Values outside the function: ", mylist)
```

Pass By Reference

```
In [ ]: def func(var):
            var.append(50);   # Accessing the same object instance
            print("Values inside the function: ", var)

        mylist = [10, 20, 30]
        func(mylist)
        # func(mylist[:])
        print("Values outside the function: ", mylist)
```

1.3.2 Function Arguments

Required arguments

```
In [ ]: def printme(s):   # This function takes exactly 1 argument irrespective of datatype
            print(s)

        printme("abc")
        printme(10)
        printme()
```

Keyword arguments

```
In [ ]: def printinfo(name, age):
            print("Name: ", name, end="\t")
            print("Age: ", age)

        # printinfo("Jia",25)
        # printinfo(25) #  TypeError: printinfo() missing 1 required positional argument: 'age
        # printinfo(50, "mikki")
        # printinfo(age=50, name="miki")
        # printinfo("miki", age=50)
        # printinfo(50, name ="miki")
```

Default arguments

```
In [ ]: def printinfo(name, age=35):
            print("Name: ", name, end="\t")
            print("Age ", age)

        # printinfo("Jia",25)
        # printinfo("Jia")
        # printinfo() # printinfo() missing 1 required positional argument: 'name'
        # printinfo(50, "mikki")
        # printinfo(age=50, name="miki")
```

Variable-length arguments

```
In [ ]: def printinfo(arg1, *var):

            print("arg1 Output is: ",arg1)
            print("vartuple output is: ",var)

            for v in var:
                print(v)

        # printinfo(10)

        printinfo(70, 60, 50, 80, 'abc')

        # printinfo()  # atleast one argument is complusory
```

1.3.3 Return Statement

```
In [ ]: def add(arg1, arg2):
            total = arg1 + arg2
            print("Inside the function : ", total)
            if total > 50:
                return total
```

```
        total1 = add(10, 20)
        print("Outside the function : ", total1)
```

1.3.4 Scope of variables

```
In [ ]: def outer():
        #     global a
            a=20
            def inner():
        #         global a
                a=30
                print("INNER a = ",a)
            inner()
            print("OUTER a = ",a)

        def func():
            print("func a = ",a)

        a=10
        outer()
        print("MAIN a = ",a)
        func()
```

```
In [2]: def func1():
            a =10
            b =20
            print("func1",a,b,y)

        def func2():
            global b,y
            a =50
            b =60
            y = 90
            print("func2",a,b,y)

        func2()

        func1()

        print("main ",b)
```

```
func2 50 60 90
func1 10 20 90
main  60
```

1.3.5 Closures

```
In [8]: def outer():
            def inner(n):
                print("n = ",n)
                if n>5:
                    return n+5
            return inner(a) # calls function internally

        a=10
        o = outer()
        print(o,type(o))
```

```
n =  10
15 <class 'int'>
```

```
In [14]: def outer():
             def inner(n):
                 print("n = ",n)
             return inner # return function object

         a=10
         o = outer()
         print(o,type(o))

         x = o(a)
         print(x, type(x))
```

```
<function outer.<locals>.inner at 0x063AE468> <class 'function'>
n =  10
10 <class 'int'>
```

```
In [ ]: def func(a):
            if a == 10:
                return "abc"
```

```
In [ ]: def fibo(n = 15):
            l = [0,1]
            for i in range(2,n+1):
                l.append(l[i-2]+l[i-1])

            def innerfibo(x = 2):
                return [i for i in l if i % x==0]

            return innerfibo
```

```
f = fibo(50)
print(f(3))
print(f())
print(f(5))

f = fibo(20)
print(f(3))
print(f())
print(f(5))
```

1.3.6 Decorators

```
In [ ]: def outer(func):
            print('outer function')
            def inner():
                print('inner function')
                return func()
            return inner

        @outer
        def display():
            print('display function executed')


        display()

        # a=outer(display)
        # print('---')
        # a()
```

1.4 Lambda Operator

```
In [16]: f = lambda x, y = 3 : x + y

         print(f, type(f))

         print(f(2))
```

```
<function <lambda> at 0x063AE7C8> <class 'function'>
5
```

```
In [19]: # def outer(n):
         #     def inner(x):
         #         return x + n
         #     return inner

         # def outer(n):
```

```
         #     return lambda x : x+n

         f = lambda n : lambda x : x + n

         a = f(10)
         b = a(10)
         # c = b(7) # error
```

```
7
12
10
14
```

```
In [22]: def func(n):
             return lambda x: [i for i in range(n) if i % x == 0 ]

         a = func(50)
         print(a(5))
```

```
[0, 5, 10, 15, 20, 25, 30, 35, 40, 45]
```

```
In [ ]: def fibo(n = 15):
            l = [0,1]
            for i in range(2,n+1):
                l.append(l[i-2]+l[i-1])

            return lambda x : [i for i in l if i % x==0]

        f = fibo()
        print(f(2))
```

Some more examples of lambda functions

```
In [25]: l = ['car','house','bike',"aeroplane"]
         l.sort(key = len) # key attribute takes object of functions
         print(l)
```

```
['car', 'bike', 'house', 'aeroplane']
```

```
In [28]: l = [1,'abc',3.4,False]
         l.sort(key = lambda x: x.__class__.__name__)
         print(l)
```

```
[False, 3.4, 1, 'abc']
```

Map function

```
In [ ]: l = ['car','house','bike']

        for i,j in zip(list(map(len,l)),l): # len is built in so can be passed directly
            print(j,i)
```

```
In [ ]: l = ((101,"n1"),(102,"n2"),(103,"n3"),(104,"n4"))
        l1 = ((101,200),(103,500),(101,300),(102,600),(101,100))

        m = map(lambda x : (x[0], sum([i[1] for i in l1 if i[0] == x[0]])) , l)
        #print(dict(m))
        d = dict(m)
        print(sorted(d.items(),key=lambda x : x[1]))
```

Filter function

```
In [ ]: f = filter(lambda x:x<0, range(-5,5))

        print(list(f))
```

```
In [ ]: f = filter(lambda x : x[1]>0,d.items())
        print(list(f))
```

```
In [ ]: l = [104,102]
        f = filter(lambda x : x[0] not in l, d.items())
        print(list(f))
```

Reduce function

```
In [ ]: from functools import reduce

        print(reduce(lambda x,y: x+y, range(10)))
```

```
In [ ]: reduce(lambda x, y: x+y, range(10), 5)
```

```
In [ ]: n = 4
        print(reduce(lambda x,y: x*y, range(1,n+1)))
```

```
In [ ]: l = ['car','house','bike',3.4]

        reduce(lambda x,y : str(x)+str(y),l)
```

```
In [ ]: reduce(lambda x, _: x+[x[-1]+x[-2]], range(8), [0, 1])
```

```
In [ ]: from functools import *
        fib = lambda n = 15 : reduce(lambda x, _: x+[x[-1]+x[-2]], range(n - 2), [0, 1])
        fib(10)
```

Prime number

```
In [ ]: x = int(input())
        l = [i for i in range(2, (x//2) + 1) if x % i == 0 ]
        print("{} is a prime number".format(x) if len(l) == 0 else l)
```

1.4.1 Storing functions into modules and importing them