Module 3:
Functions

By Ninad Gaikwad

Reference - “Core Python Programming”
Dr. R. Nageshwara Rao
Dreamtech Press

Overview

- Perform the required task
- Can be reused as and when required
- Provide modularity to the programming
- Code maintenance becomes easy
- Code debugging becomes easy
- Reduce length of the program
- Difference between a function and a method:
  - When a function is written inside a class it becomes a method

9.1 Function Introduction

- Defining a function:
    a. Using def keyword
    b. Eg def sum(a, b):
            return a+b
- Calling a function:
    a. Eg sum(10, 15)
- Returning results from a function:
    a. Return output from a function
    b. Eg return a+b
- Returning multiple values from a function
    a. Eg def sum_sub(a, b):
            c=a+b
            d=a-b
            return c, d
- Functions are first class objects
    a. Functions are treated as perfect objects
    b. Possible to assign a function to a variable
    c. Possible to define a function inside another function
    d. Possible to pass a function as a parameter to another function
    e. Possible that function can return another function
- Pass by object reference
    a. Parameters are passed by object reference

9.2 Arguments of function

1. Formal and actual arguments
   a. Positional arguments
      i. Passed to a function in correct positional order
      ii. Number and position of arguments is maintained
   b. Keyword arguments
      i. Identify parameters by their name
      ii. Eg def grocery(item, price): # function definition
      grocery(price= 88.00, item= ‘oil’) # function call
   c. Default arguments
      i. Mention some default value which is assigned if the value is not specified
      ii. Eg def grocery(item, price=90): # function definition
      grocery(item= ‘oil’) # function call
   d. Variable length arguments
      i. Used when the number of arguments in the function call is unknown
      Eg def add(fargs, *args): # fargs represents formal arguments *args stores all other arguments in a tuple
      for i in arg: # gives the individual arguments stored in args
      ii. Keyword variable length arguments or **kwargs are arguments that are variable in length and in the form of key value pair
      Eg def display(fargs, **kwargs): # variable length key value pair
      for x, y in kwargs.items():

9.2 Arguments of function

2. Local and Global variables
   a. Global keyword

      Eg

```
a=1
def myfunction():
    global a
    print(‘global a = ’, a) # displays global value of a
    a=2
    print(‘modified a=’, a) # displays local value of a

myfunction()
print(‘global a = ’, a) # displays global value of a
```

3. Passing a group of elements to a function
   b. Passing list, array, dictionary to functions

9.3 Recursive and anonymous functions

- Recursive functions

  Eg Finding factorial of first 10 numbers

  ```
  def factorial(n):
      if n==0:
          result=1
      else:
          result =n*factorial(n-1)
      return result

  for i in range(1,11):
      print(‘factorial of {} is {}’.format(i, factorial(i)))
  ```

- Anonymous (Lambda) functions
  1. Functions without a name and starting with keyword lambda
  2. lambda argument_list: expression

     Eg
     ```
     f = lambda x: x*x
     value = f(5)
     ```
  3. Lambdas contain only one expression and they return the result implicitly

9.3 Recursive and anonymous functions

d. Lambdas with filter() function:

- Filters out elements of a sequence depending on result of function
- filter(function, sequence)

```
Eg lst= [10, 23, 30, 45, 60]
lst1 = list(filter(lambda x: (x%2==0), lst))
print(lst1)
# will print the even values from the above list
```

e. Lambda with map() function

- Acts on each element of the sequence and the modified elements are returned to be stored in another sequence
- map(function, sequence)

```
Eg lst=[1, 2, 3, 4]
lst1 = list(map(lambda x: x*x, lst))
print(lst1)
# will print the list containing squares of all the numbers in the list
```

f. Lambda with reduce() function

- Reduces the sequence of elements to a single value by processing the elements
- reduce(function, sequence)

```
Eg lst = [1, 2, 3, 4]
reduce(lambda x, y: x*y, lst)
# will give the product of all the elements
```

9.4 Function decorators

- Decorators are functions that take functions as parameters and return functions as results
- Decorators modify the results of a function, they are useful to perform additional processing
- Eg

```
def decor(fun):
    def inner():
        value = fun()
        return value+2
    return inner

def decor1(fun):
    def inner():
        value = fun()
        return value*2
    return inner

# function to which decorator is to be applied
def num():
    return 10

result_fun = decor(decor1(num))
print(result_fun)
```

9.4 Function decorators

- Eg

```
def decor(fun):
        def inner():
                value = fun()
                return value+2
        return inner

def decor1(fun):
        def inner():
                value = fun()
                return value*2
        return inner

# function to which decorator is to be applied
@decor
@decor1
def num():
        return 10

print(num())
```

9.5 Generators

- Return sequence of values
- Uses yield statement
- Eg

```
def mygen(x, y):
        while x<=y:
                yield x
                x+=1

g = mygen(5, 10)
for i in g:
        print(i, end=’ ‘)
```

- Convert output to list

```
lst = list(g)
```

- Retrieve elements of list

```
next(g)
```

9.7 Creating your own modules in python

9.7 Creating your own modules in python

- Save file as ‘filename.py’ file
- While accessing the file write ‘import filename’ or ‘from filename import *’

9.8 Special variable __name__

- Stores information about whether the program is executed individually or as a module
- Eg

```
# one.py
def display():
    print(“Hello python”)

if __name__ == ‘__main__’:
    display()
    print(“This code runs as a program”)
else:
    print(“This code runs as a module”)

# two.py
import one
one.display()

## run one.py and two.py
```