12. Python - Functions - Part 2

Table of Contents

12. Python - Functions - Part 2

1. Formal and actual arguments

- ✓ When a function is defined it may have some parameters.
- ✓ These parameters receive the values.
  - Parameters are called as a 'formal arguments'
  - When we call the function, we should pass values or data to the function.
    - These values are called as '**actual arguments**'

Program Name: Formal and actual arguments
demo1.py

```
def add(a, b):
        c = a + b
        print(c)

# call the function

x = 10
y = 15
add(x, y)
```

output

```
25
```

- ✓ **a** and **b** called as formal arguments
- ✓ **x** and **y** called actual arguments

2. Types of arguments

In python there are 4 types of actual arguments are existing,

1. positional arguments
2. keyword arguments
3. default arguments
4. variable length arguments (*args)

2.1. Positional arguments

- ✓ These are the arguments passed to a function in correct positional order.
- ✓ The number of arguments and position of arguments should be matched, otherwise we will get error.

Program Name: Positional arguments
demo2.py

```
def sub(x, y):
        print(x-y)

# calling function

sub(20, 10)
```

output

```
10
```

Make a note

- ✓ If we change the number of arguments, then we will get error.
- ✓ This function accepts two arguments then if we are trying to provide three values then we will get error.

Program Name: Error: Positional arguments
demo3.py

```
def sub(x, y):
        print(x-y)

# calling function

sub(10, 20, 30)
```

output

```
TypeError: sub() takes 2 positional arguments but 3 were given
```

2.2. Keyword arguments

- ✓ Keyword arguments are arguments that recognize the parameters by the name of the parameters.
- ✓ We can use an identifier (name of the variable) to provide values to the function parameters.

Program Name: keyword arguments
demo4.py

```
def cart(product, price):
        print("Product is :" , product)
        print("cost is :" , price)

cart(product = "bangles", price = 200000)
```

output

```
Product is: bangles
cost is: 200000
```

Program Name: keyword arguments
demo5.py

```
def cart(product, price):
        print("Product is :" , product)
        print("cost is :" , price)

cart(product = "handbag ", price = 100000)
```

output

```
Product is: handbag
cost is: 100000
```

Program Name: keyword arguments
demo6.py

```
def cart(product, price):
        print("Product is:" , product)
        print("cost is:" , price)

cart(price = 1200, product = "shirt")
```

output

```
Product is: shirt
cost is: 1200
```

2.3. Default arguments

- ✓ During function definition we can provide default values to function parameters
- ✓ While creating a function, we can provide values to the function parameters.

Program Name: Default arguments
demo7.py

```
def cart(product, price = 40.0):
        print("Product is :" , product)
        print("cost is :" , price)

# calling function

cart(product = "pen")
```

output

```
Product is: pen
Cost is : 40.0
```

Program Name: Default arguments
demo8.py

```
def cart(product, price = 40.0):
        print("product is :", product)
        print("cost is :", price)

cart(product = "handbag", price = 10000)
```

output

```
Product is: handbag
Cost is : 10000
```

Program Name: Default arguments
demo9.py

```
def cart(product, price = 40.0):
        print("Product is:", product)
        print("cost is:", price)

cart(price = 500, product = "shirt")
```

output

```
Product is: shirt
Cost is : 500
```

Make a note

- ✓ If we are not passing any value, then only default value will be considered.

2.4. * args or variable length arguments

- ✓ *args also called as variable length arguments.
- ✓ Sometimes our requirement can be like, need to provide more values to function parameter to process the result. Here we can use *args concept.
- ✓ It represents single star symbol before the argument name.
- ✓ Internally the provided values will be stored in a tuple.

Few points

- ✓ If we define one parameterised function then during function calling we need to pass one value.
- ✓ If we define two parameterised function then during function calling we need to pass two values, if we pass more or less than two values then we will get error.

Syntax

```
def nameofthefunction(*x):
        body of the function
```

- ***x** is variable length argument

- ✓ Now we can pass any number of values to this ***x**.
- ✓ Internally the provided values will be represented in tuple.

Program Name: Variable length argument
demo10.py

```
def m(x):
    print(x)

m(10)
```

output

```
10
```

Program Name: Variable length argument
demo11.py

```
def m(x):
    print(x)

m(10, 20)
```

output

```
TypeError: m() takes 1 positional argument but 2 were given
```

Program Name: Variable length argument
demo12.py

```
def m(*x):
    print(x)

m(10)
```

output

```
(10)
```

Program Name: Variable length argument
demo13.py

```
def m(*x):
    print(x)

m(10, 20)
```

output

```
(10, 20)
```

Program Name: Variable length argument
demo14.py

```
def m(*x):
    print(x)

m(10, 20, 30)
```

output

```
(10, 20, 30)
```

3. Anonymous functions or Lambdas

- ✓ Generally we can create normal function by using def keyword
- ✓ lambda is a keyword in python.
- ✓ By using lambda keyword we can create lambda function.
- ✓ Lambda function also called as anonymous function.
- ✓ A function without a name is called as anonymous function.
- ✓ Lambda function will process the input and return the result.

Lambda function syntax

```
lambda argument_list: expression
```

Advantage

- ✓ By using Lambda Functions, we can write very concise code so that readability of the program will be improved.

Program Name: demo16.py

```
s = lambda a: a*a

x = s(4)
print(x)
```

output

```
16
```

- ✓ Here because of lambda keyword it creates anonymous function.

A simple difference between normal and lambda functions

Program Name: To find square by using a normal function
demo17.py

```
def square(t):
        return t*t

s=square(2)
print(s)
```

output

```
4
```

Program Name: To find sum of two values by using normal function
demo18.py

```
def add(x, y):
        return x + y

b = add(2, 3)
print(b)
```

output

```
5
```

Program Name: To find sum of two values by using anonymous function
demo19.py

```
add = lambda x, y: x+y

result = add(1, 2)
print("The sum of value is:", result)
```

output

```
3
```

Make notes

- ✓ Lambda Function internally returns expression value and we no need to write return statement explicitly.

Where lambda function fits exactly?

- ✓ Sometimes we can pass function as argument to another function. In such cases lambda functions are best choice.
- ✓ We can use lambda functions very commonly with filter(),map() and reduce() functions, these functions expect function as argument.

Lambda functions

- ✓ map(p1, p2) function
- ✓ filter(p1, p2) function
- ✓ reduce(p1, p2) function

3.1. map(p1, p2) function

- ✓ map(fun, iterable) is a predefined function in python.
- ✓ This function takes iterable and apply logic on every item and returns new iterable.

Syntax

```
map(function, sequence)
```

Program Name: Find square by using map function
demo20.py

```
without_gst_cost = [100, 200, 300, 400]
with_gst_cost = map(lambda x: x+10, without_gst_cost)

x = list(with_gst_cost)
print("Without GST items costs: ", without_gst_cost)
print("With GST items costs: ", x)
```

output

```
Without GST items costs:  [100, 200, 300, 400]
With GST items costs:  [110, 210, 310, 410]
```

3.2. filter(p1, p2) function

- ✓ filter(fun, iterable) is a predefined function in python.
- ✓ This function takes iterable and apply filtering logic on every item and returns new iterable.

Syntax

```
filter(function, sequence)
```

Program Name: example by using filter function
demo21.py

```
items_cost = [999, 888, 1100, 1200, 1300, 777]
gt_thousand = filter(lambda x : x>1000, items_cost)

x = list(gt_thousand)
print("Eligible for discount:", x)
```

output

```
Eligible for discount: [1100, 1200, 1300]
```

3.3. reduce(p1, p2) function

- ✓ reduce(fun, iterable) is a predefined function existing in functools module.
- ✓ This function apply a function of two arguments cumulatively to the items of sequence, from left to right.
- ✓ Finally this function reduce the sequence to a single value.

Syntax

```
reduce(function, sequence)
```

Program Name: reduce function
demo22.py

```
from functools import reduce

items_cost = [111, 222, 333, 444]
total_cost = reduce(lambda x, y : x+y, items_cost)
print(total_cost)
```

output

```
1110
```

- ✓ For example, reduce(lambda x, y: x+y, [1, 2, 3, 4, 5]) calculates ((((1+2)+3)+4)+5)