جامعة الانبار
كلية علوم الحاسوب وتكنولوجيا المعلومات
قسم أنظمة شبكات الحاسوب

برمجة كيانية OOP
المرحلة الثانية
الفصل الدراسي الأول والثاني

مدرس المادة
م.د. سميه عبدالله حمد

Function Overloading in Python

In python, function overloading is defined as the ability of the function to behave in different ways depend on the number of parameters passed to it like zero, one, two which will depend on how function is defined. Overloading function provides code reusability, removes complexity and improves code clarity to the users who will use or work on it. Function overloading in python can be of two types one is overloading built-in functions and overloading the custom or user-defined functions in python. We will have a look into both of them in the below sections. In general, not every programming language supports function overloading but in this case, python supports functional overloading.

Syntax of Function Overloading in Python

The syntax and example is as follows:

Syntax:

In python, we can define a method in such a way that it can be called in different ways using different numbers of parameters. We will see a function overloading example which can take zero or one argument and its syntax as below:

Example:

```
class World:
    def hello(self, name=None):
        if name is not None:
            print (“Hello “, name)
        else:
            print(“Hello”)
obj = World        # calling function without any argument
obj.hello(“srinivas”) # calling function with an argument.
```

In the above syntax example, we have created a class World with a method/function hello where we set the first argument is None so that we can call the function with or without an argument. We have created an obj of the class World and using this obj we will call its method using zero or one argument. In order to see how function overloading is working, we will call the function with zero parameters as obj.hello() and with one parameter as obj.hello("srinivas") and the output of the above program is as below. The above example is having up to one variable but it is not limited to it we can have a number of parameters.

Output:
Hello

How Function Overloading Works in Python?

Let us see how overloading of functions works in python using an example as below. Let us have a function to calculate the area of square and rectangle.

Code:

```
def area(l, b):
  c = l*b
  return c
def area(size):
  c = size*size
  return c
area(4)
area(5,6)
```

Output:

```
Traceback (most recent call last):
  File "C:/Users/PC/AppData/Local/Programs/Python/Python37/x.py", line 8, in <module>
    area(5,6)
TypeError: area() takes 1 positional argument but 2 were given
```

In python, when we define two functions with the same name than the function which we defined later only a valid function in python. So when we execute area(4) it executes properly but when we execute area(5,6) gives an error saying function area() takes exactly one argument. So by default function overloading is not there in python but it can be achieved using decorators, defining functions by setting parameters default values to None.

Let us have an example of function overloading by defining a function with default parameter values as None.

Code:

```
class Compute:
    def area(self, x=None, y=None):
        if x!=None and y !=None:
            return x*y
        elif x!=None:
            return x*x
        else:
```

```
        return 0
obj = Compute()
print(obj.area())
print(obj.area(6))
print(obj.area(2,8))
```

Output:

0

36

16

In the above example, we have defined a class Compute with a function named the area where we have default parameter values as None so that the function can be called either with zero, one, and two parameters. If we have one argument then the area function will return zero as its output, if it has one parameter then the area function returns the square of the parameter, and if it has two parameters then the area function will return the product of two parameters as output. We have created an obj for the class Compute by which we can access the function area() with different parameters. Here we called obj.area() which gives output as 0, obj.area(6) gives the output as 36, and obj.area(2,8) gives output as 16 which is the product of 2 and 8.

Now we will see built-in function overloading with an example of overloading the len() function as below:

Built-in Function Overloading

Code:

```
class Purchase:

    def __init__(self, basket, consumer):

        self.basket= list(basket)

        self.consumer= consumer

    def __len__(self):

        return 10

purchase = Purchase(['pencil','book'],'python')

print(len(purchase))
```

In the above example, we defined a class Purchase where it has constructor __init__ with parameters basket which is a list and consumer is a string variable and assigning to the self. We have a function __len__ which is overriding the built-in len() function of python. We have created an object purchase of class Purchase with parameters that will be initialized. When we execute the print statement in which we are calling len(purchase) will return 10 as we overloaded the built-in len() function of python. If it calls built-in len() function it will throw an error as an object doesn't have any len() function.

Output:

The above program when function overloading successes:

10

The output of the above program, if we didn't perform function overloading is an error:

TypeError: object of type 'purchase' doesn't have len().

Example of Function Overloading in Python

Let us have an example of a class student having a function hello with a parameter name having default value as None as below:

Example #1

Code:

```
class Student:
    def sayHello(self, name=None):
        if name is not None:
            self.name = name
            print("Hey, " , name)
        else:
            print("Hey")
stu = Student()
stu.sayHello()
stu.sayHello('dasu')
```

In the above example, we have a class Student with a function sayHello() where default parameter value is set to None so that the function can be called with either zero, one parameter but not limited to them. We have created an stu of class Student by which we can access the function sayHello(). While calling stu.sayHello() output of it will be “hey”, and while calling stu.sayHello(“dasu”) output of it will be “hey dasu”.

Output:

Hey

Hey, dasu