Inspiring Excellence

Course Title: Programming Language II

Course Code: CSE 111

Semester: Summer 2020

Topic: Method and Constructor Overloading

Table of Contents

METHOD OVERLOADING

When a function or an operator behaves in different ways which are based on the parameters they are passed to the function, or the operands that the operator acts on is known as Overloading. Python cannot have two methods consisting of same names like other languages and by default, python doesn't support method overloading. We might try coding it but python executes only the latest defined method. For example:

Code	Output

```python
# First product method.
# Takes two argument and print them
# product
def potpies(x,y):
    p= x * y
    return p

# Second product method
# Takes three argument and print them
# product
def potpies(x,y,z):
    p= x * y *z
    return p
# Uncommenting the below line shows an error
#print(potpies(1,2))

# This line will call the second product method
print(potpies(1,2,3))
```

```
> 6
```

Since Python doesn't support overloading, the second method is only used. We may define many methods of the same name and different argument but python will use the latest defined method. And calling other methods will produce an error. We can use a different way to overcome the above problem.

However, there are some ways following which method overloading can be done in python:

Code	Output

```python
def add(datatype, *args): # Function to take multiple arguments

    if datatype =='int': # for datatype "int" ans 0
        answer = 0
    if datatype =='str': # for datatype "str" ans ''
        answer =''
    for x in args: # Traverse through the arguments
        # This will do addition if the
        # arguments are int. Or concatenation
        # if the arguments are str
        answer = answer + x

    print(answer)
# Integer
add('int', 3, 7)

# String
add('str', 'Keep ', 'Rocking')
```

```
>10
>Keep Rocking
```

In the above example, using the arguments we can make the same function work in different ways i.e. as per the arguments.

Let's take another example:

Code	Output

`# Lets determine size of a pizza:`
 	
`class size_pizza:`
`# area method`
`    def size(self, a = None, b = None):`
`        if a != None and b != None:`
`            return a * b`
`        elif a != None:`
`            return a * a`
`        else:`
`            return 0`
`# now we create an object named pizza`
`pizza = size_pizza()`
 	
`# zero argument`
`print(" Size of pizza :", pizza.size())`
`# one argument`
`print("Size of pizza :", pizza.size(3))`
`# two argument`
`print("Size of pizza :", pizza.size(3, 5))`
	>Size of pizza : 0
	>Size of pizza : 9
	>Size of pizza : 15

In the above example, we are overloading the size method where the conditions are:
if there is no argument then it returns 0.

- If one argument, it returns the square of the value.
- Lastly, for two arguments it only returns the product of the two values.

CONSTRUCTOR OVERLOADING

In python, to overload a constructor we have to write a single constructor that handles all cases, using either default arguments or type or capability tests.

Code	Output

`class Kitty:`
`    def __init__(self, lullaby=None):`
`        if lullaby is None:`
`            print ("Soft kitty, Warm kitty")`
`        else:`
`            print ("Little ball of fur", lullaby)`
`l1=Kitty()`
 	
`l2=Kitty("Happy kitty, Sleepy kitty, phur phur phur ")`
	>Soft kitty, Warm kitty
	>Little ball of fur Happy kitty, Sleepy kitty, phur phur phur

In the above example, we are declaring one constructor which has one or more conditions. Here, the mentioned parameter of the constructor is handling the works both as a parameterized constructor and non-parameterized constructor.

Again, in the below example through passing several arguments in the main constructor we can do constructor overloading

Code	Output

`class Employee:`
`  def __init__(self,e_name=None, e_id=None):`
`  if e_name != None and e_id!= None:`
`  print(f"The employee's name is {e_name} with ID {e_id}")`
`  elif e_name != None:`
`  print("The employee's name: "+e_name)`
`  else:`
`  print("Empty nothing to display ")`
`emp1=Employee()`
`emp1=Employee("Besant")`
`emp1=Employee("Besant",23132)`
	>Empty nothing to display
	>The employee's name: Besant
	>The employee's name is Besant with ID 23132