Overloading

Overloading, in the context of programming, refers to the ability of a function or an operator to behave in different ways depending on the parameters that are passed to the function, or the operands that the operator acts on.

Overloading a method fosters reusability. For instance, instead of writing multiple methods that differ only slightly, we can write one method and overload it. Overloading also improves code clarity and eliminates complexity.

Method Overloading

Like other languages do, python does not supports method overloading. We may overload the methods but can only use the latest defined method.

Example	Output

```
def abc(a, b):
   print a+b

def abc(a,b,c):
   print a+b+c

#abc(10,20)
abc(10,20,30)
```
	30

In the above code we have defined two abc method, but we can only use the second product method, as python does not supports method overloading. We may define many method of same name and different argument but we can only use the latest defined method. Calling the other method will produce an error. Like here calling abc(10,20) will produce an error as the latest defined abc method takes three arguments.

Operator Overloading

Python allows us to change the default behavior of an operator depending on the operands that we use. This practice is referred to as "operator overloading".

The functionality of Python operators depends on built-in classes. However, the same operator will behave differently when applied to different types. A good example is the "+" operator. This operator will perform an arithmetic operation when applied on two numbers, will concatenate two strings, and will merge two lists.

Example	Output

a=10
b=20
print a+b
 	
str1="Hello"
str2="Python"
print str1+str2
	30
	HelloPython

Method overriding

Method overriding, in object-oriented programming, is a language feature that allows a subclass or child class to provide a specific implementation of a method that is already provided by one of its super classes or parent classes. The implementation in the subclass overrides (replaces) the implementation in the super class by providing a method that has same name, same parameters or signature, and same return type as the method in the parent class.

The version of a method that is executed will be determined by the object that is used to invoke it. If an object of a parent class is used to invoke the method, then the version in the parent class will be executed, but if an object of the subclass is used to invoke the method, then the version in the child class will be executed.

Example	**Output**
class A:	Hi Python
def a(self):	
print "Hello Python"	
class B(A):	
def a(self):	
print "Hi Python"	
objb=B()	
objb.a()	

In this example we create the object of parent class A and access the method using parent class object

Example	**Output**

class A:
    def a(self):
        print "Hello Python"
class B(A):
    def a(self):
        print "Hi Python"
 	
obja=A()
obja.a()
	Hello Python