Method Overloading
&
Method Overriding

A R U N A R U N I S T O

In **Python**, you cannot achieve traditional **method overloading** directly as you would in languages like **Java** or **C++**. In those languages, **method overloading** involves having multiple methods with the same name but different signatures within the same class.

```
#method overloading
#compile time polymorphism
class Demo:
    def add(self, a, b):
        return a+b
    def add(self, a, b, c):
        return a+b+c


obj = Demo()
obj.add(1, 2, 3) #6
obj.add(1, 2) #it will raise an error
```

in Python, methods are just attributes of a class, and when you define multiple method-

s with the same name, only the latest definition is retained.

In the above code, only the second definition of the **add** method with **three parameters** is retained, and the first one is overwritten. This is why encountering an error when try to call **add** with only **two arguments.**

If you want to achieve similar functionality in Python, you can use default arguments like this:

```
#method overloading in python
class Demo:
    def add(self, a, b, c=None):
        if c!=None:
            return a+b+c
        return a+b

obj = Demo()
obj.add(1, 2, 3) #6
obj.add(1, 2) #3
```

```
#method overriding Python
#runtime polymorphism
class Demo:
    def add(self, a, b):
        return a+b

class sub(Demo):
    def add(self, a, b):
        return a+b

obj = sub()
obj.add(1, 2) #3
obj1 = Demo()
obj1.add(1, 2) #3
```

In Python, **method overriding** allows a subclass to provide a specific implementation of a method that is already defined in its superclass. When you call the overridden method on an instance of the subclass, the subclass's implementation is executed instead of the superclass's implementation. This is a form of **runtime polymorphism**.

In the above code, both **obj** and **obj1** have an add method. When you call **add** on **obj**, it executes the overridden method defined in sub, while when you call **add** on **obj1**, it executes the method defined in Demo. This demonstrates **runtime polymorphism** in Python.