Function Overloading in Python

Definition:

Function overloading is a feature where multiple functions with the same name can have different implementations based on the number or type of arguments passed. Unlike languages like C++ and Java, Python **does not support traditional function overloading** because functions in Python are dynamically typed, and the latest definition of a function overrides the previous ones.

Behaviour in Python:

If multiple functions with the same name are defined, only the last definition remains valid, and previous definitions are overridden.

Eg:

```
def fun1():
    print("P")
def fun1():
    print("J")
def fun1():
    print("Z")
fun1()
```

- The first two fun1 definitions are overridden.
- Only the last definition (print("Z")) remains.

```
def fun1(a):
    print("1")
def fun1(a,b):
    print("2")
def fun1(a,b,c):
    print("3")
fun1(10)
```

- Similar to the first part, the last definition of fun1(a,b,c) overrides the previous ones.
- When calling fun1(10), Python expects fun1 to take **three** arguments (a, b, c).
- Since fun1(10) provides only **one** argument, Python raises a **TypeError**

Method Overriding in Python

Definition:

Method overriding is a feature of object-oriented programming where a subclass provides a new implementation of a method that is already defined in its superclass. The overridden method in the subclass must have **the same name and parameters** as in the parent class.

How Method Overriding Works:

1. When a method is called on an object of the subclass, Python first checks if the method exists in the subclass.
2. If found, the subclass's method is executed instead of the superclass's method.
3. If not found, Python looks for the method in the superclass.

Eg:

```
# Base class A
class A:
    def display(self):
        print("Inside A")  # Method in class A


# Class B inherits from A and overrides display()
class B(A):
    def display(self):
        print("Inside B")  # Method overridden in class B


# Class C inherits from B and overrides display()
class C(B):
    def display(self):
        print("Inside C")  # Method overridden in class C


# Class D inherits from C
class D(C):
    def dispD(self):
        A.display(self)  # Calls A's display() method
        B.display(self)  # Calls B's display() method
        C.display(self)  # Calls C's display() method
# Creating an object of class D
d1 = D()
d1.dispD()  # Calls dispD() which invokes display() from A, B, and C
```