Poly

By **Great Learning Team** - Jun 15, 2021

python write on book with laptop keyboard background

Introduction

Suppose there is a class called animal, but in that class, a whole lot of forms are there like dog, cat, cow etc. That means a common class animal consists of different forms that exist in various forms and shapes and have different functionalities. This whole thing can be defined by a single term called polymorphism. Polymorphism in python is something that is used to describe the ability to take various forms. The word is derived from two different words poly which means many and morphs meaning forms.

Also Read: Python Tutorial For Beginners

Class and Objects

Class is c
we creat
into it. An object is created that prints the value of x.

Example: A simple example showing class and object in python

```
class A:
  x = 5
object = A()
print(object.x)
```

Output

5

Constructor

Constructor is a type of subroutine in object-oriented programming. Constructor is used to assigning value to data members when an object is created inside a class. The __init__() function is used as a constructor in python almost every time we create an object. In polymorphism, we use this __init__() function almost everywhere.

What is polymorphism in Python?

Polymorphism in python is used for a common function name that can be used for different types. This concept is widely applied in object-oriented based python programming. Like other programming languages say Java, C+, polymorphism is also implemented in python for different purpose commonly Duck Typing, Operator overloading and Method overloading, and Method overriding. This polymorphism process can be achieved in two main ways namely overloading and overriding.

A simple example of polymorphism in python

>>>4+5

9

>>>"4"+"5"

45

>>>"ab"+"cd"

abcd

In the ab
different

In the first example the as the data sent are two integer values, the operator did the addition operation of two numbers.

And in the second example, as the same values are sent as data in the form of string, and the same operator did the concatenation (concatenation is the process by which two strings are joined end-to-end) of the two strings.

In the third example, the data sent are two strings but the operator is same as the above and it did the concatenation of the two strings.

>>>"3"+"ab"

3ab

In this example the addition operator did the concatenation of the two strings though the first value is a integer but here in this specific example it is sent as strings.

So these were the few basic examples of polymorphism in python.

How to use polymorphism?

Overloading

Overloading is divided into two types

- Operator Overloading
- Method Overloading

Operator overloading

Operator overloading is the type of overloading in which an operator can be used in multiple ways beyond its predefined meaning.

>>>print(2*7)

14

>>>print("a"*3)

aaa

So in the above example the multiplication operator did the multiplication of two numbers in the first one, but in the second one as the multiplication of a string and

Practical implementation of operator overloading

Example 1:

```
class Vehicle:
    def __init__(self, fare):
        self.fare = fare

bus= Vehicle(20)
car= Vehicle(30)
total_fare=bus+ car
print(total_fare)
```

Output:

```
Traceback (most recent call last):

 File "G:\python pycharm project\main.py", line 7, in <module>

  total_fare=bus+ car

TypeError: unsupported operand type(s) for +: 'Vehicle' and 'Vehicle'
```

Here in the above example an error has occurred, this is because python has no idea how to add two objects together. Here Vehicle is the object.

Now comes the application of operator overloading here.

Now we will overload the special function __add__ operator.

```
class Vehicle:
    def __init__(self, fare):
        self.fare = fare
    def __add__(self, other)://using the special function __add__
operator
        return self.fare+ other.fare

bus= Vehicle(20)
car= Vehicle(30)
total_fare=bus+ car
print(total_fare)
```

Output:

Overloading the special function magically defines everytime we use plus operator in the object total_fare=bus+ car we are going to add their fares
Example 2: Let us compare the fares of different vehicles in this example

```
class Vehicle:
    def __init__(self, fare):
        self.fare = fare
    def __lt__(self, other):// relational operator  __lt__ is used
here as the special function
        return self.fare< other.fare

bus= Vehicle(10)
car= Vehicle(30)
compare=bus< car
print(compare)
```

Output:

True

__lt__ It is the relational operator which is used as a special function for operator overloading in the above example.

Method Overloading

Method overloading means a class containing multiple methods with the same name but may have different arguments. Basically python does not support method overloading, but there are several ways to achieve method overloading. Though method overloading can be achieved, the last defined methods can only be usable.

Let's take an example for better understanding

Let's consider a class A, within the class we have taken a function show which has a constructor self and arguments whose default values assigned are None and None. Then I have created an object and called the function with the object obj.show, but I didn't pass any argument though it will print None and None cause in the function part we assigned the default values.

Example:

```
class A:
    def show(self, a=None, b=None):
        print(a,b)
```

```
obj=A(
obj.sh
```

Output

None None

Now if I want to pass any other value then I have to call another function obj.show() with an argument in it.

Example:

```
class A:
    def show(self, a=None, b=None):
        print(a,b)

obj=A()
obj.show()
obj.show(4)
```

Output

None None

4 None

Here in the output the None value assigned for a in the function part is replaced with 4. In the function call part 4 is passed as the argument.

Now if we pass two argument during the function call see what will happen in the next example,

Example:

```
class A:
    def show(self, a=None, b=None):
        print(a,b)

obj=A()
obj.show()
obj.show(4)
obj.show(4,5)
```

Output

None None

4 5

Here as we pass two arguments 4 and 5 during function call two different values are assigned for a and b.

So in the above example we have seen how we can use the same method and use different function calls in different ways.

Let's see another example where we have used conditional statements to call different function in different ways

Example:

```
class Area:
    def find_area(self, a=None, b=None):
        if a != None and b != None:
            print("Rectangle:", (a * b))
        elif a != None:
            print("square:", (a * a))
        else:
            print("No figure assigned")

obj1=Area()
obj1.find_area()
obj1.find_area(10)
obj1.find_area(10,20)
```

Output

No figure assigned

square: 100

Rectangle: 200

In the above example if no arguments are passed during function call then it will show no value assigned, if we pass a single argument then it will show the area of square and if 2 values are passed then it will show the area of rectangle.

Inheritance

Before getting into Method overriding we need to know the first Inheritance in python. Inheritance is the process through which a class can be derived from any base class through which the derived class can inherit all the properties from the base class.

Example of inheritance

```
class Bird://base class Bird
    def sound(self):
        print("Birds Sounds")
#child class Dog inherits the base class Animal
class Sparrow(Bird)://child class Sparrow
    def tweet(self):
        print("sparrow tweeting")
d = Sparrow()
d.tweet()
d.sound()
```

Output

Sparrow tweeting
Birds Sound
Method Overriding

Method overriding is the process of modifying a base class from the derived class having the same methods and same number of arguments.

Let's take an example and see how it works. First of all we will create a base class with a method and then create a derived class without any method,

Example:

```
class Vehicle:
    def run(self):
        print("Saves Energy")
class EV(Vehicle):
    pass



ev = EV()
ev.run()
```

So when the function is called, the output will print the base class's method, as the derived class has no method in it.

Output:

Saves Energy

Now in th
the same
method is now overridden by the derived class, only the derived class's method will be printed as the output.

Example:

```
class Vehicle:
    def run(self):
        print("Saves Energy")



class EV(Vehicle):
    def run(self):
        print("Run on Electricity")



ev = EV()
ev.run()
```

Output

Run on Electricity

Super() Function

Now as if the base class's method has been overridden, the method from the base class cannot be called normally. So to call the method of base class we have to use the super function in the overridden method under the derived class.

Example:

```
class Vehicle:
    def run(self):
        print("Saves Energy")
class EV(Vehicle):
    def run(self):
    super().run()//super function is used to call the method of
base class

        print("Run on Electricity")

ev = EV()
ev.run()
```

Outpu

Saves Energy

Run on Electricity

Duck Typing

Duck typing is a concept in polymorphism. The term duck typing is derived from the quote which says something that walks like a duck quacks like a duck and swims like a duck is called a duck no matter what object it is. In simple words it means something that matches its behaviour to anything then it will consider a thing of that category to which it matches.

Also Read: Python Interview Questions

Why is polymorphism needed?

When Talking about object-oriented programming in python the term polymorphism will come for sure. In object-oriented programming, objects need to take different forms. In software development, it's quite important to have this feature. Through polymorphism, a single action can be performed in various ways. This concept is widely used when we talk about loose coupling, dependency injection and interface etc. If you wish to learn more about python, take up the great learning academy's free course on python and power ahead your career.

Great Learning Team

Gre
levera

yourself updated with the fast-changing world of tech and business.