Research article Available online www.ijsrr.org ISSN: 2279–0543

International Journal of Scientific Research and Reviews

A Study onthe Effect of Method Overriding In Python

A Sujitha Amalin Nancy*

Department of Computer Science,AJK College of Arts and Science.Navakkarai,
Coimbatore- 641105, T.N, INDIA,
Email: sujithaamalinnancy@gmail.com. Mob No- 7397506777

ABSTRACT

In object oriented programming the fundamental uses of generalization is the reusability of code and better structuring of the objects description. This paper is about the effect of method overriding in python which enables developers to combine concepts and increase the reusability of the resulting software.

***Corresponding author**

Mrs. A SujithaAmalin Nancy

Assistant Professor,

Department of Computer Science,

AJK College of Arts and Science.

Navakkarai, Coimbatore-641 105, T.N, INDIA,

Email: sujithaamalinnancy@gmail.com. Mob No- 7397506777

1.INTRODUCTION

Python supports different programming due to its multi-paradigm programming approach .creating object is the easiest and effective approach to solve programming problem. This is known as object oriented programming. Attributes and behavior are the two characteristics .example color; name is the attributes where as dancing and singing are the behaviors.

Python follows the DRY(don't repeat yourself) concept for an effective programming. In python the process of using details from a new class without modifying existing class was called inheritance .while encapsulation hides the private details of a class from other objects. Whereas the concept of using common operation in different ways for different data input is called polymorphism

2. OBJECT ORIENTED PROGRAMMING CONCEPTS IN PYTHON

2.1 Class

Class is always known as the blue print for the object in object oriented programmingconcepts.Class contains all the details about the object. For example if pigeon is the object, we can specify its features like color,size etc in the class.

```
class pigeon:
species = "bird"
```

Here, we use ***class*** keyword to define an empty class **pigeo*n*.** From class, we construct instance which is a specific object created from a particular class.

A class by itself is of no use unless there is some functionality associated with it. Functionalities are defined by setting attributes, which act as containers for data and functions related to those attributes. Those functions are called methods[1].

2.2 Object

An object (instance) is an instantiation of a class .when class is defined,only the description for the object is defined .therefore, no memory or storage is allocated

You can create multiple different objects that are of the same class(have the same variables and functions defined). However, each object contains independent copies of the variables defined in the class. For instance, if we were to define another object with the "MyClass" class and then change the string in the variable above[2]

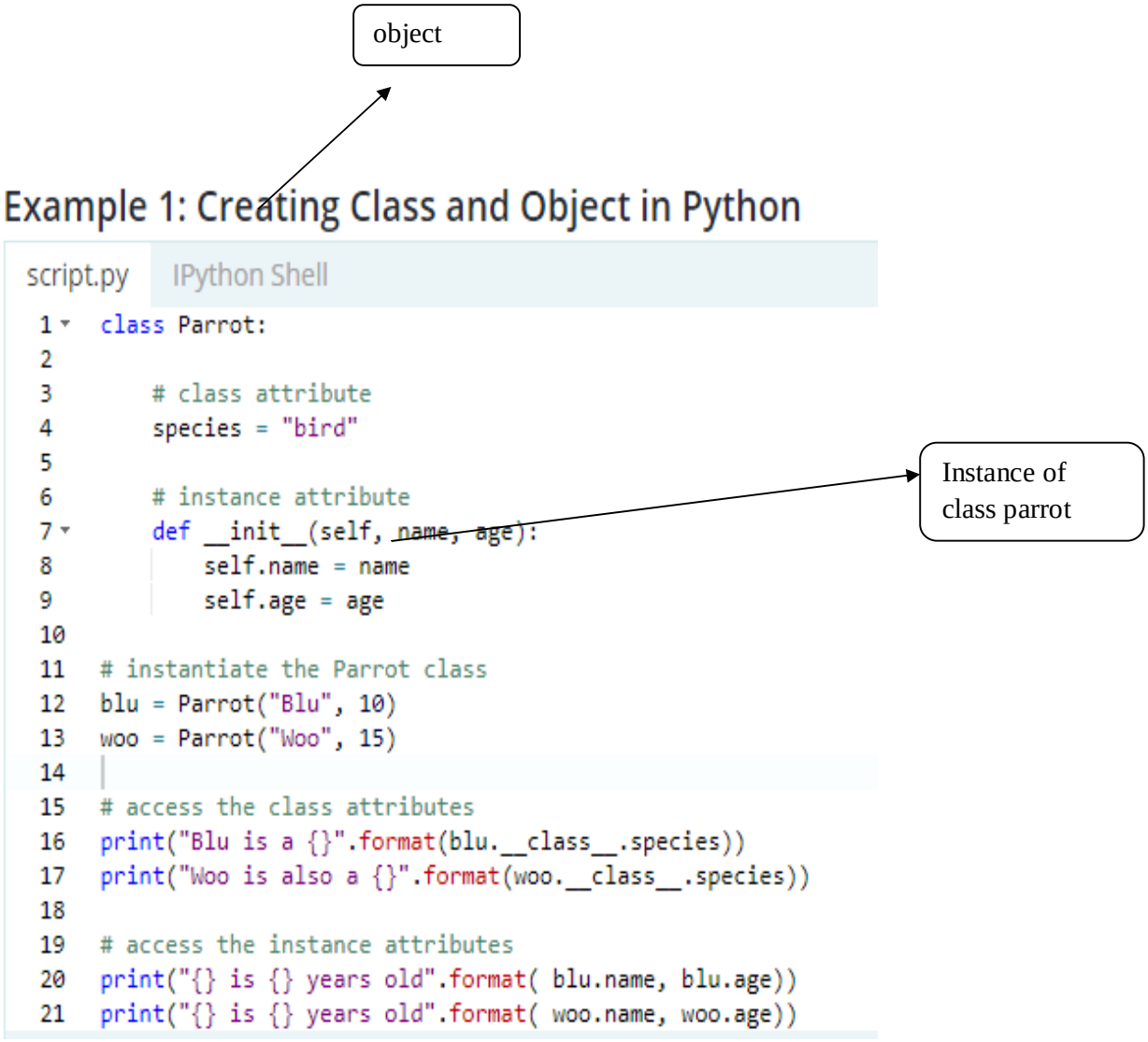

Figure 1.class and objects in python

We create instances of the Parrot class. Here, *blu* and woo are references (value) to our new objects.Then, we access the class attribute using *__class_.species*. Class attributes are same for all instances of a class. Similarly, we access the instance attributes using *blu.name* and *blu.age*. However, instance attributes are different for every instance of a class.

3.INHERITANCE

Inheritance is a powerful tool in object oriented programming.Using inheritance we can define a new class with little or no modification to an existing class. The new class is called derived class (or child class)and the one from which we inherits the class is called base class(or parent class)

Figure 2. Inheritance In Python

Syntax for inheritance in python:

classBaseClass:

Body of base class

classDerivedClass(BaseClass):

Body of derived class

Here, derived class inherits the features from base class, adding new features to it. This results into reusability of code.

Example for inheritance in python

classPolygon:

def __init__(self,no_of_sides):

self.n=no_of_sides

self.sides=[0for i inrange(no_of_sides)]

definputSides(self):

self.sides=[float(input("Enter side "+str(i+1)+" : "))for i in range(self.n)]

defdispSides(self):

for i in range(self.n):

print("Side",i+1,"is",self.sides[i])

This class has data attributes to store the number of sides, *n* and magnitude of each side as a list, *sides*.Method *inputSides()* takes in magnitude of each side and similarly, *dispSides()* will display these properly.

A triangle is a polygon with 3 sides. So, we can created a class called *Triangle* which inherits from *Polygon*. This makes all the attributes available in class *Polygon* readily available in *Triangle*. We don't need to define them again (code re-usability). *Triangle* is defined as follows.

```
classTriangle(Polygon):
def __init__(self):
Polygon.__init__(self,3)
deffindArea(self):
a, b, c =self.sides
# calculate the semi-perimeter
    s =(a + b + c)/2
area=(s*(s-a)*(s-b)*(s-c))**0.5
print('The area of the triangle is %0.2f'%area)
```

However, class Triangle has a new method *findArea()* to find and print the area of the triangle. Here is a sample run.

```
>>> t =Triangle()
>>>t.inputSides()
Enter side 1:3
Enter side 2:5
Enter side 3:4
>>>t.dispSides()
Side1is3.0
Side2is5.0
Side3is4.0
>>>t.findArea()
The area of the triangle is6.00
```

We can see that, even though we did not define methods like *inputSides()* or *dispSides()*for class *Triangle*, we were able to use them.

If an attribute is not found in the class, search continues to the base class. This repeats recursively, if the base class is itself derived from other classes.

4.METHOD OVERRIDING IN PYTHON

In the above example, notice that *__init__()* method was defined in both classes, *Triangle* as well *Polygon*. When this happens, the method in the derived class overridesthat in the base class.

This is to say, *__init__() in Triangle* gets preference over the same in *Polygon*.Generally when overriding a base method, we tend to extend the definition rather than simply replace it. The same is being done by calling the method in base class from the one in derived class[3]
(calling *Polygon.__init__()* from *__init__() in Triangle).*

A better option would be to use the built-in function *super().So, super().__init__(3)* is equivalent to *Polygon.__init__(self,3)* and is preferred. You can learn more about the super() function in Python.Two built-in functions *isinstance()* and *issubclass()* are used to check inheritances.

Function *isinstance()* returns *True* if the object is an instance of the class or other classes derived from it. Each and every class in Python inherits from the base class *object*

```
>>>isinstance(t,Triangle)
True
>>>isinstance(t,Polygon)
True
>>>isinstance(t,int)
False
>>>isinstance(t,object)
True
```

Similarly, issubclass() is used to check for class inheritance.

```
>>>issubclass(Polygon,Triangle)
False
>>>issubclass(Triangle,Polygon)
True
>>>issubclass(bool,int)
True
```

5.REFERENCES

1. https://www.hackerearth.com/practice/python/object-oriented-programming/classes-and-objects-i/tutorial/
2. https://www.learnpython.org/en/Classes_and_Objects
3. https://www.programiz.com/python-programming/multiple-inheritance