



جامعة الانبار
كلية علوم الحاسوب وتكنولوجيا المعلومات
قسم أنظمة شبكات الحاسوب

برمجة كيانية OOP
المرحلة الثانية
الفصل الدراسي الأول والثاني

مدرس المادة
م.د. سميه عبدالله حمد

Object Oriented Programming (OOP)

Inheritance in Python

One of the core concepts in object-oriented programming (OOP) languages is inheritance. It is a mechanism that allows you to create a hierarchy of classes that share a set of properties and methods by deriving a class from another class. Inheritance is the capability of one class to derive or inherit the properties from another class.

Benefits of inheritance are:

- It represents real-world relationships well.
- It provides the **reusability** of a code. We don't have to write the same code again and again. Also, it allows us to add more features to a class without modifying it.
- It is transitive in nature, which means that if class B inherits from another class A, then all the subclasses of B would automatically inherit from class A.
- Inheritance offers a simple, understandable model structure.
- Less development and maintenance expenses result from an inheritance.

Python Inheritance Syntax

Class BaseClass:

{Body}

Class DerivedClass(BaseClass):

{Body}

Object Oriented Programming (OOP)

Creating a Parent Class

Creating a **Person** class with **Display** methods.

```
# A Python program to demonstrate inheritance

class Person(object):

    # Constructor

    def __init__(self, name, id):

        self.name = name

        self.id = id

    # To check if this person is an employee

    def Display(self):

        print(self.name, self.id)

# Driver code

emp = Person("Satyam", 102) # An Object of Person

emp.Display()
```

Output:

Satyam 102

Creating a Child Class

Here **Emp** is another class which is going to inherit the properties of the **Person** class(base class).

Object Oriented Programming (OOP)

```
class Emp(Person):  
  
    def Print(self):  
  
        print("Emp class called")  
  
Emp_details = Emp("Mayank", 103)  
  
# calling parent class function  
  
Emp_details.Display()  
  
# Calling child class function  
  
Emp_details.Print()
```

Output:

Mayank 103

Emp class called

Object Oriented Programming (OOP)

Example of Inheritance in Python

```
#A Python program to demonstrate inheritance

#Base or Super class. Note object in bracket.
#(Generally, object is made ancestor of all classes)
#In Python 3.x "class Person" is
#equivalent to "class Person(object)"

class Person(object):
    # Constructor
    def __init__(self, name):
        self.name = name
    # To get name
    def getName(self):
        return self.name
    # To check if this person is an employee
    def isEmployee(self):
        return False
# Inherited or Subclass (Note Person in bracket)
class Employee(Person):
    # Here we return true
    def isEmployee(self):
        return True

# Driver code
emp = Person("Geek1") # An Object of Person
print(emp.getName(), emp.isEmployee())
emp = Employee("Geek2") # An Object of Employee
print(emp.getName(), emp.isEmployee())
```

Output :

Geek1 False
Geek2 True

Object Oriented Programming (OOP)

What is object class?

In Python (from version 3. x), the object is the root of all classes.

Subclassing (Calling constructor of parent class)

A child class needs to identify which class is its parent class. This can be done by mentioning the parent class name in the definition of the child class.

Eg: class **subclass_name** (**superclass_name**):

```
# Python code to demonstrate how parent constructors
# are called.

# parent class
class Person(object):

    # __init__ is known as the constructor
    def __init__(self, name, idnumber):

        self.name = name

        self.idnumber = idnumber

    def display(self):

        print(self.name)

        print(self.idnumber)

# child class
class Employee(Person):

    def __init__(self, name, idnumber, salary, post):
```

Object Oriented Programming (OOP)

```
self.salary = salary  
  
self.post = post  
  
# invoking the __init__ of the parent class  
  
Person.__init__(self, name, idnumber)  
  
# creation of an object variable or an instance  
  
a = Employee('Rahul', 886012, 200000, "Intern")  
  
# calling a function of the class Person using its instance  
  
a.display()
```

Output:

Rahul

886012

‘a’ is the instance created for the class Person. It invokes the `__init__()` of the referred class. You can see ‘object’ written in the declaration of the class Person. In Python, every class inherits from a built-in basic class called ‘object’. The constructor i.e. the ‘`__init__`’ function of a class is invoked when we create an object variable or an instance of the class. The variables defined within `__init__()` are called the instance variables or objects. Hence, ‘name’ and ‘idnumber’ are the objects of the class Person. Similarly, ‘salary’ and ‘post’ are the objects of the class Employee. Since the class Employee inherits from class Person, ‘name’ and ‘idnumber’ are also the objects of class Employee.

Object Oriented Programming (OOP)

Python program to demonstrate error if we forget to invoke `__init__()` of the parent

If you forget to invoke the `__init__()` of the parent class then its instance variables would not be available to the child class.

The following code produces an error for the same reason.

```
class A:

    def __init__(self, n='Rahul'):

        self.name = n

class B(A):

    def __init__(self, roll):

        self.roll = roll

object = B(23)

print(object.name)
```

Output :

Traceback (most recent call last):

File `"/home/de4570cca20263ac2c4149f435dba22c.py"`,
line 12, in

```
    print (object.name)
```

AttributeError: 'B' object has no attribute 'name'