OOPS - Inheritance

August 23, 2022

Inheritance In Python

What is meant by Inheritance ?

Simply, Inheritance is when a class uses code constructed within another class.

Inheritance enables us to define a class that takes all the functionality from a parent class and allows us to add more

In terms of biology, we can think of a child inheriting certain traits from their parents.

Parent Class :

It is the class being inherited from, also called a base class.

Child Class :

It is the class that inherits from another class, also called a derived class.

Benefits of Inheritance :

Code re usability, here are many possible methods to let you reuse the same functions and properties in your code.

It is transitive in nature, which means if class B inherits from another class A, then all subclasses of B would automatically inherit from class A.

Less Development : Changes just need to be made in the base class, all derived classes will automatically follow.

Modular Codebase: Increases modularity i.e. breaking down codebase into modules which makes it easier to understand.

[1]:
```
#Suppose you have the following Person class:Parent Class
class Person:
    def __init__(self, name):
        self.name = name

    def greet(self):
        return f"Hi, it's {self.name}"
```

[2]:
```
#Now, you want to define the Employee that is similar to the Person class:
class Employee:
    def __init__(self, name, job_title):
```

```
        self.name = name
        self.job_title = job_title

    def greet(self):
        return f"Hi, it's {self.name}"
```

[3]:
```
#The following redefines the Employee class that inherits from the Person class:
↪Child Class

class Employee(Person):
    '''
    The Person class is the parent class, the base class, or the super class of␣
↪the Employee class
    The Employee class is a child class, a derived class, or a subclass of the␣
↪Person class.
    '''
    def __init__(self, name, job_title):
        self.name = name
        self.job_title = job_title
```

[4]:
```
employee = Employee('John', 'Python Developer') #object
print(employee.greet())
```

```
Hi, it's John
```

[5]:
```
print(dir(Employee))
```

```
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__',
'__format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__',
'__init_subclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__',
'__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__',
'__str__', '__subclasshook__', '__weakref__', 'greet']
```

[6]:
```
#To check if an object is an instance of a class, you use the isinstance()␣
↪method
print(isinstance(employee, Person))
```

```
True
```

[8]:
```
#To check if a class is a subclass of another class, you use the issubclass()␣
↪function.
print(issubclass(Employee, Person))
```

```
True
```

Types Inheritance

Depending upon the number of classes(i.e. Child and parent classes). There are five types of inheritance in Python.

Single Inheritance

Multiple Inheritance

Multilevel Inheritance

Hierarchical Inheritance

Hybrid Inheritance

Single Inheritance

Single Inheritance is the simplest form of inheritance where a single child class is derived from a single parent class

Due to its candid nature, it is also known as simple inheritance.

[10]:
```
#Single Inheritance

# Base class
class Vehicle:
    def Vehicle_info(self):
        print('Inside Vehicle class')

# Child class
class Car(Vehicle):
    def car_info(self):
        print('Inside Car class')

# Create object of Car
car = Car()

# access Vehicle's info using car object
car.Vehicle_info()
car.car_info()
```

```
Inside Vehicle class
Inside Car class
```

Multiple Inheritance

In multiple inheritance, one child class can inherit from multiple parent classes

So here is one child class and multiple parent classes.

[22]:
```
# Parent class 1
class Person:
    def person_info(self, name, age):
        print('Inside Person class')
        print('Name:', name, 'Age:', age)

# Parent class 2
class Company:
    def company_info(self, company_name, location):
```

```
        print('Inside Company class')
        print('Name:', company_name, 'location:', location)

# Child class
class Employee(Person, Company):
    def Employee_info(self, salary, skill):
        print('Inside Employee class')
        print('Salary:', salary, 'Skill:', skill)

# Create object of Employee
emp = Employee()

# access data
emp.person_info('Jessa', 28)
print()
emp.company_info('Google', 'Atlanta')
print()
emp.Employee_info(12000, 'Machine Learning')
```

```
Inside Person class
Name: Jessa Age: 28

Inside Company class
Name: Google location: Atlanta

Inside Employee class
Salary: 12000 Skill: Machine Learning
```

[8]:
```
#However, if two parents have the same "named" methods,
#the child class performs the method of the first parent in order of reference

class parent1:                          # first parent class
    def func1(self):
        print("Hello Parent1")

class parent2:                          # second parent class
    def func2(self):
        print("Hello Parent2")

class parent3:                          # third parent class
    def func2(self):                    # the function name is same as parent2
        print("Hello Parent3")

class child(parent1, parent3, parent2):     # child class
    def func3(self):                        # we include the parent classes
        print("Hello Child")                # as an argument comma separated
```

```
# Driver Code
test = child()          # object created
test.func1()            # parent1 method called via child
test.func2()            # parent2 method called via child instead of parent3
test.func3()            # child method called

# to find the order of classes visited by the child class, we use __mro__␣
 ↪(Method Resolution Order)on the child class
print(child.__mro__)
```

```
Hello Parent1
Hello Parent3
Hello Child
(<class '__main__.child'>, <class '__main__.parent1'>, <class
'__main__.parent3'>, <class '__main__.parent2'>, <class 'object'>)
```

Multilevel inheritance

We can also inherit from a derived class. This is called multilevel inheritance.

Suppose three classes A, B, C. A is the superclass, B is the child class of A, C is the child class of B.

In other words, we can say a chain of classes is called multilevel inheritance.

[9]:
```
#Multi level Inheritence
# Base class
class Vehicle:
    def Vehicle_info(self):
        print('Inside Vehicle class')

# Child class for Vechile , Parent Class for SportsCar
class Car(Vehicle):
    def car_info(self):
        print('Inside Car class')

# Child class
class SportsCar(Car):
    def sports_car_info(self):
        print('Inside SportsCar class')

# Create object of SportsCar
s_car = SportsCar()

# access Vehicle's and Car info using SportsCar object
s_car.Vehicle_info()
s_car.car_info()
s_car.sports_car_info()
```

```
Inside Vehicle class
```

```
Inside Car class
Inside SportsCar class
```

Hierarchical Inheritance

In Hierarchical inheritance, more than one child class is derived from a single parent class

In other words, we can say one parent class and multiple child classes.

Hierarchical Inheritance is the right opposite of multiple inheritance

[25]:
```python
#Hierarchical Inheritance

#Base Class
class Vehicle:
    def info(self):
        print("This is Vehicle")

#Child Class 1
class Car(Vehicle):
    def car_info(self, name):
        print("Car name is:", name)

#Child Class 2
class Truck(Vehicle):
    def truck_info(self, name):
        print("Truck name is:", name)

obj1 = Car()
obj1.info() #info is method of parent class
obj1.car_info('BMW')
print()
obj2 = Truck()
obj2.info() #info is method of parent class
obj2.truck_info('Ford')
```

```
This is Vehicle
Car name is: BMW

This is Vehicle
Truck name is: Ford
```

Hybrid Inheritance

When inheritance is consists of multiple types or a combination of different inheritance is called hybrid inheritance.

Here we can have many to many relations between parent classes and child classes with multiple levels.

[31]:
```
#Base Class
class Family:
    def show_family(self):
        print("This is our family:")

# Father class inherited from Family
class Father(Family):
    fathername = ""
    def show_father(self):
        print(self.fathername)

# Mother class inherited from Family
class Mother(Family):
    mothername = ""
    def show_mother(self):
        print(self.mothername)     #-->Hierarchical Inheritance

# Son class inherited from Father and Mother classes
class Son(Father, Mother):         #-->Multiple Inheritance
    def show_parent(self):
        print("Father :", self.fathername)
        print("Mother :", self.mothername)


s = Son() # Object of Son class
s.fathername = "Mark"
s.mothername = "Sonia"
s.show_family()
s.show_parent()
```

```
This is our family:
Father : Mark
Mother : Sonia
```

Super function

In child class, we can refer to parent class by using the super() function.

The super function returns a temporary object of the parent class that allows us to call a parent class method inside a child class method.

Benefits of using the super ( ) function.

We are not required to remember or specify the parent class name to access its methods.

We can use the super ( ) function in both single and multiple inheritances.

The super ( ) function support code reusability as there is no need to write the entire function

[32]:
```
class Company:
    def company_name(self):
```

```
        return 'Google'

class Employee(Company):
    def info(self):
        # Calling the superclass method using super()function
        c_name = super().company_name()
        print("Jessa works at", c_name)

# Creating object of child class
emp = Employee()
emp.info()
```

```
Jessa works at Google
```

Disadvantages of Inheritance in Python :

Decreases the Execution Speed: loading multiple classes because they are interdependent on each other

Tightly Coupled Classes: this means that even though parent classes can be executed independently, child classes cannot be executed without defining their parent classes.