Inheritance

The **process of inheriting the properties of the parent class into a child class is called inheritance**. The existing class is called a base class or parent class and the new class is called a subclass or child class or derived class.

In Object-oriented programming, inheritance is an important aspect. The main purpose of inheritance is the **reusability** of code because we can use the existing class to create a new class instead of creating it from scratch.

In inheritance, the child class acquires all the data members, properties, and functions from the parent class. Also, a child class can also provide its specific implementation to the methods of the parent class.

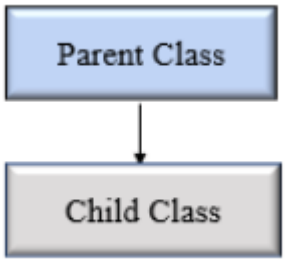
Types of Inheritance

In Python, based upon the number of child and parent classes involved, there are five types of inheritance. The type of inheritance are listed below:

1. Single inheritance
2. Multiple Inheritance
3. Multilevel inheritance
4. Hierarchical Inheritance
5. Hybrid Inheritance

Single Inheritance

In single inheritance, a child class inherits from a single-parent class. Here is one child class and one parent class.



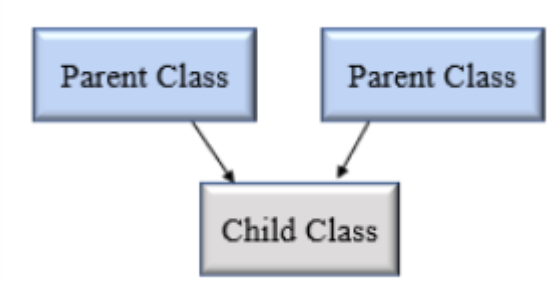
Python Single Inheritance

Example 1:

```
# 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()
```

Multiple Inheritance

In multiple inheritance, one child class can inherit from multiple parent classes. So here is one child class and multiple parent classes.



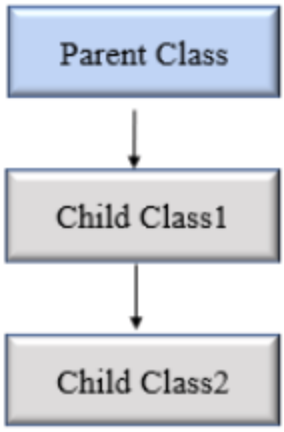
Python Multiple Inheritance

Example 2:

```
# 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)
emp.company_info('Google', 'Atlanta')
emp.Employee_info(12000, 'Machine Learning')
```

Multilevel inheritance

In multilevel inheritance, a class inherits from a child class or derived class. 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.**



Python Multilevel Inheritance

Example 3:

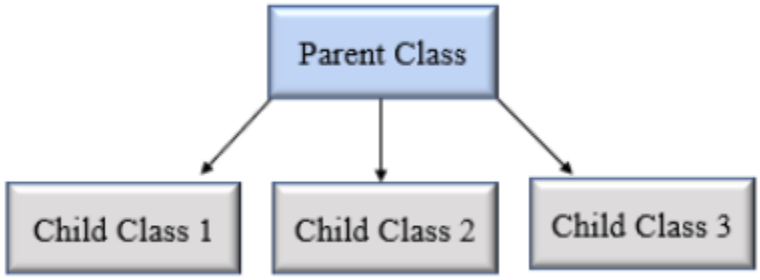
```
# Base class
class Vehicle:
    def Vehicle_info(self):
        print('Inside Vehicle class')

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

# 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()
```

NOTE: In the above example, we can see there are three classes named Vehicle, Car, SportsCar. Vehicle is the superclass, Car is a child of Vehicle, SportsCar is a child of Car. So we can see the chaining of classes.

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.



Python hierarchical inheritance

Example

Let's create 'Vehicle' as a parent class and two child class 'Car' and 'Truck' as a parent class.

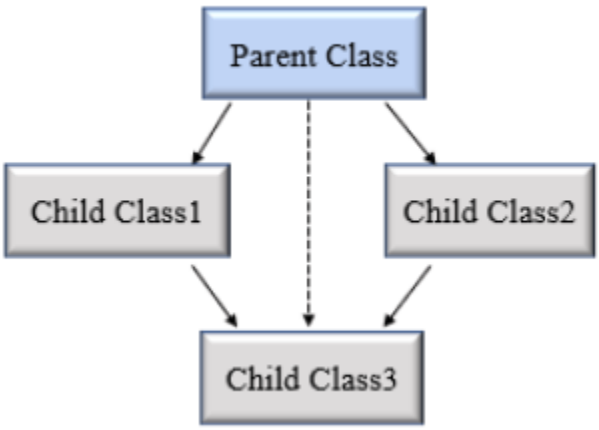
Example 4:

```
class Vehicle:
    def info(self):
        print("This is Vehicle")
```

```
class Car(Vehicle):
    def car_info(self, name):
        print("Car name is:", name)

class Truck(Vehicle):
    def truck_info(self, name):
        print("Truck name is:", name)

obj1 = Car()
obj1.info()
obj1.car_info('BMW')

obj2 = Truck()
obj2.info()
obj2.truck_info('Ford')
```

Hybrid Inheritance

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



Python hybrid inheritance

Example 5:

```
class Vehicle:
    def vehicle_info(self):
        print("Inside Vehicle class")
```

```
class Car(Vehicle):
    def car_info(self):
        print("Inside Car class")

class Truck(Vehicle):
    def truck_info(self):
        print("Inside Truck class")

# Sports Car can inherits properties of Vehicle and Car
class SportsCar(Car, Vehicle):
    def sports_car_info(self):
        print("Inside SportsCar class")

# create object
s_car = SportsCar()

s_car.vehicle_info()
s_car.car_info()
s_car.sports_car_info()
```

Note: In the above example, **hierarchical** and **multiple** inheritance exists. Here we created, parent class `Vehicle` and two child classes named `Car` and `Truck` this is hierarchical inheritance.

Another is `SportsCar` inherit from two parent classes named `Car` and `Vehicle`. This is multiple inheritance.

Python `super()` function

When a class inherits all properties and behavior from the parent class is called inheritance. In such a case, the inherited class is a subclass and the latter class is the parent class.

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.

1. We are not required to remember or specify the parent `class` name to access its methods.
2. We can use the `super()` function in both **single** and **multiple inheritances**.
3. The `super()` function support code **reusability** as there is no need to write the entire function

Example 1:

```
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()
```

Example 2:

```
# Single Inheritance
class One:
    i,j = 10,20

    def __init__(self):
        print("Super classes constr...")
        pass

    def add(self):
        print("Add result is: ", (self.i + self.j))
        pass
```

```
    def sub(self):
        print("Sub result is: ", (self.i - self.j))
        pass


class Two(One):
    a,b = 100,20
    i,j = 12,4

    def __init__(self):
        super().__init__() # calling super classes constr in sub class constr
        print("Sub classes constr...")
        print(self.i , self.j)
        print(super().i, super().j) # Super classes variables
        pass

    def mul(self):
        print("Mul result is: ", (self.a * self.b))
        pass

    def div(self):
        print("Div result is: ", (self.a / self.b))
        pass


obj = Two() # Only the sub classes constr is called
```

Example 3:

```
# Prog to access super classes constr, varible and method
class One:
    i,j = 10,20

    def __init__(self):
        print("Class One constr...")
        pass

    def add(self):
        print("Add result from class One is: ", (self.i + self.j))
        pass

class Two(One):
```

```
    a,b = 100,20

    def __init__(self):
        print("Class Two constr...")
        pass

    def sub(self):
        print("Sub result is: ", (self.a - self.b))
        pass

    def access(self):
        print("Accessing super classes variable type 1: ", super().i)
        print("Accessing super classes variable type 2: ",
super(Two,self).i)

        # Accessing super classes method
        super().add()
        super(Two,self).add()

        # Accessing super classes constr
        super().__init__()
        super(Two, self).__init__()

two = Two()
two.access()
```

---

issubclass()

In Python, we can verify whether a particular class is a subclass of another class. For this purpose, we can use Python built-in function `issubclass()`. This function returns `True` if the given class is the subclass of the specified class. Otherwise, it returns `False`.

Syntax

```
issubclass(class, classinfo)
```

Example 4:

```
# It's important to note that class is considered a subclass of
itself.
class Company:
    def fun1(self):
        print("Inside parent class")

class Employee(Company):
    def fun2(self):
        print("Inside child class.")

class Player:
    def fun3(self):
        print("Inside Player class.")

# Result True
print(issubclass(Employee, Company))

# Result False
print(issubclass(Employee, list))

# Result False
print(issubclass(Player, Company))

# Result True
print(issubclass(Employee, (list, Company)))

# Result True
print(issubclass(Company, (list, Company)))
```

Method Overriding

In inheritance, all members available in the parent class are by default available in the child class. If the child class does not satisfy with parent class implementation, then the child class is allowed to redefine that method by extending additional functions in the child class. This concept is called **method overriding**.

When a child class method has the same name, same parameters, and same return type as a method in its superclass, then the method in the child is said to **override** the method in the parent class.

Example 5:

```
class Vehicle:
    def max_speed(self):
        print("max speed is 100 Km/Hour")

class Car(Vehicle):
    # overridden the implementation of Vehicle class
    def max_speed(self):
        print("max speed is 200 Km/Hour")

# Creating object of Car class
car = Car()
car.max_speed()
```