OOPs Python - 2

Class Methods
In Object-oriented programming, Inside a Class, we can define the following three types of methods.

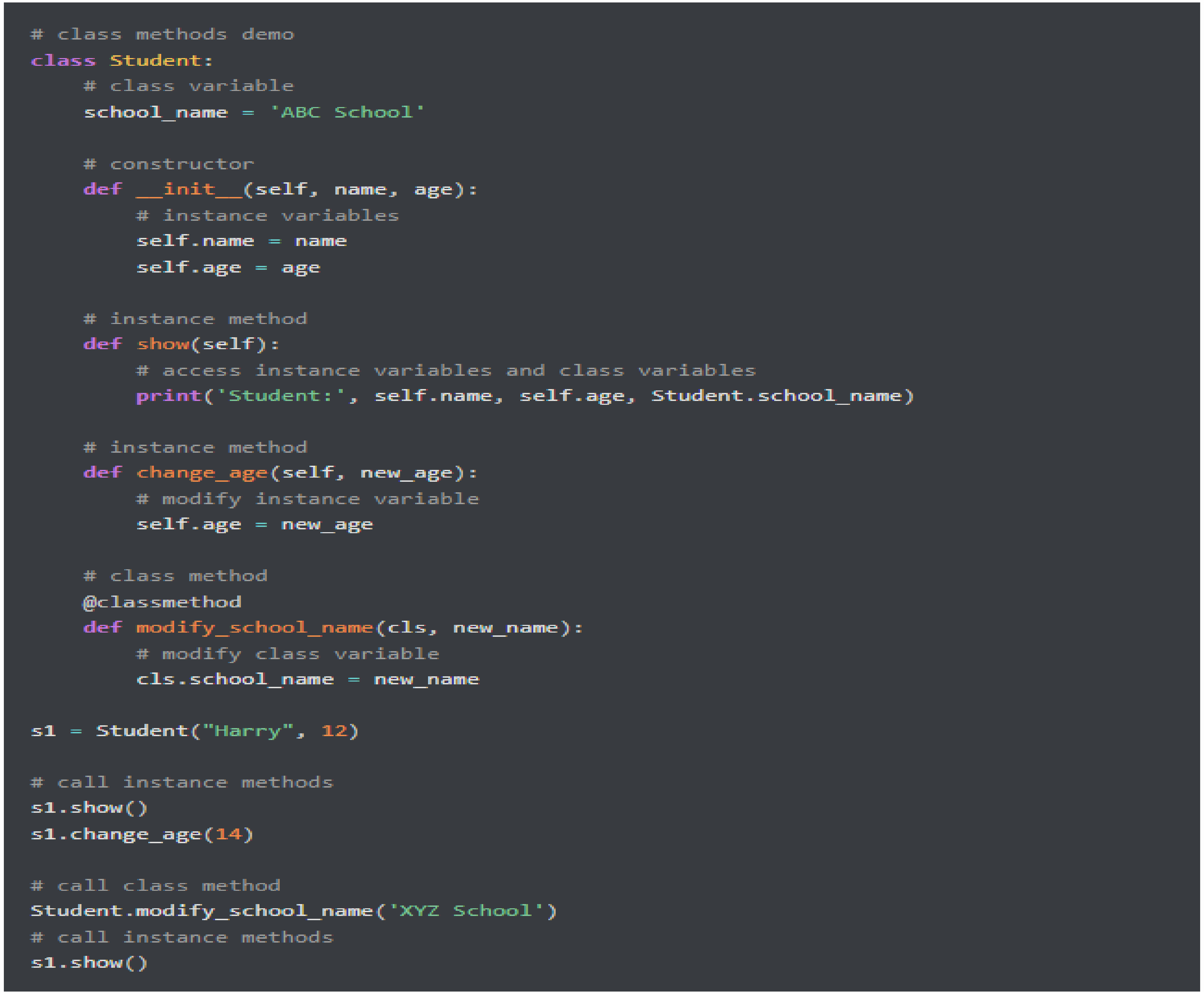
Instance method: Used to access or modify the object state. If we use instance variables inside a method, such methods are **called instance methods.**

Class method: Used to access or modify the class state. In method implementation, if we use only class variables, then such type of methods we should declare as a class method.

Static method: It is a general utility method that performs a task in isolation. Inside this method, we don’t use instance or class variable because this static method doesn’t have access to the class attributes.

Instance methods work on the instance level (object level).

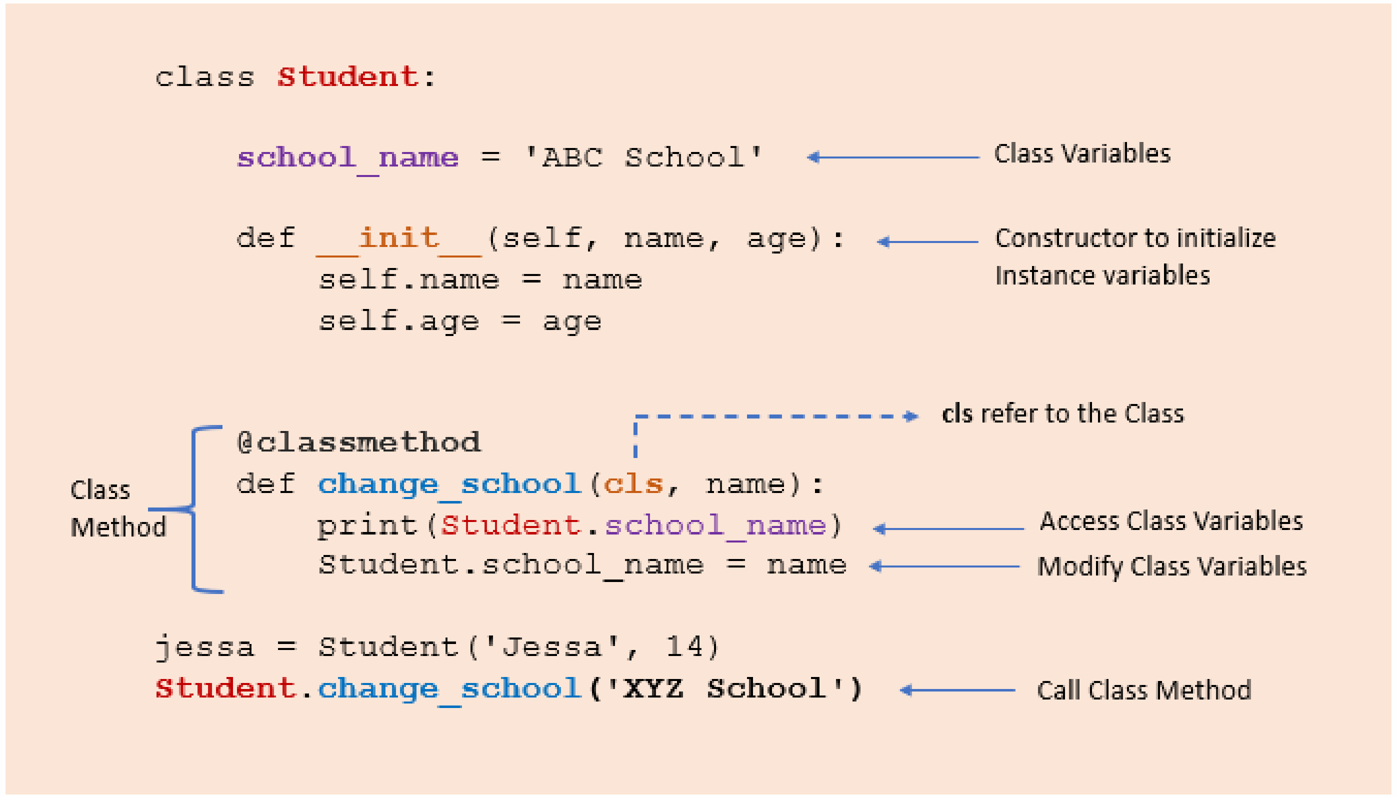
A class method is bound to the class and not the object of the class.



```python
# class methods demo
class Student:
    # class variable
    school_name = 'ABC School'

    # constructor
    def __init__(self, name, age):
        # instance variables
        self.name = name
        self.age = age

    # instance method
    def show(self):
        # access instance variables and class variables
        print('Student:', self.name, self.age, Student.school_name)

    # instance method
    def change_age(self, new_age):
        # modify instance variable
        self.age = new_age

    # class method
    @classmethod
    def modify_school_name(cls, new_name):
        # modify class variable
        cls.school_name = new_name

s1 = Student("Harry", 12)

# call instance methods
s1.show()
s1.change_age(14)

# call class method
Student.modify_school_name('XYZ School')
# call instance methods
s1.show()
```

We should follow specific rules while we are deciding a name for the class in Python.

Rule-1: Class names should follow the UpperCaseCamelCase convention
Rule-2: If a class is callable (Calling the class from somewhere), in that case, we can give a class name like a function.
Rule-3: Python's built-in classes are typically lowercase words



```
class Student:

    school_name = 'ABC School'

    def __init__(self, name, age):
        self.name = name
        self.age = age


    @classmethod
    def change_school(cls, name):
        print(Student.school_name)
        Student.school_name = name

jessa = Student('Jessa', 14)
Student.change_school('XYZ School')
```

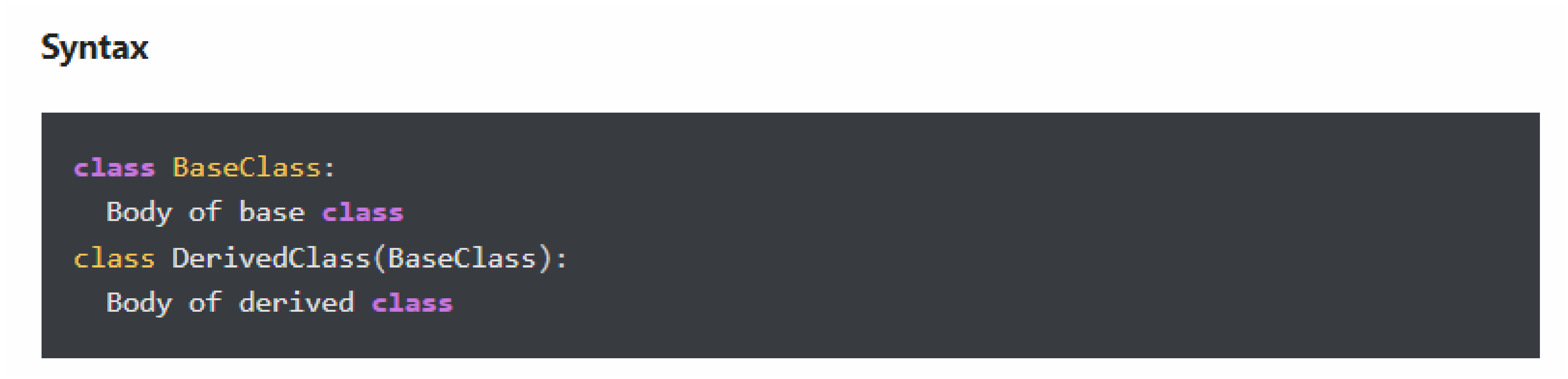
Define class method

Inheritance in Python

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 this Python lesson, you will learn inheritance, method overloading, method overriding, types of inheritance, and MRO (Method Resolution Order).

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.



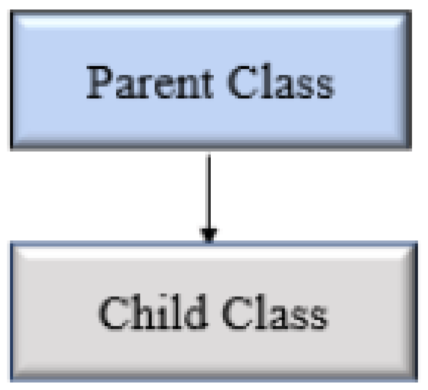
Syntax

```
class BaseClass:
  Body of base class
class DerivedClass(BaseClass):
  Body of derived 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



Python Single Inheritance

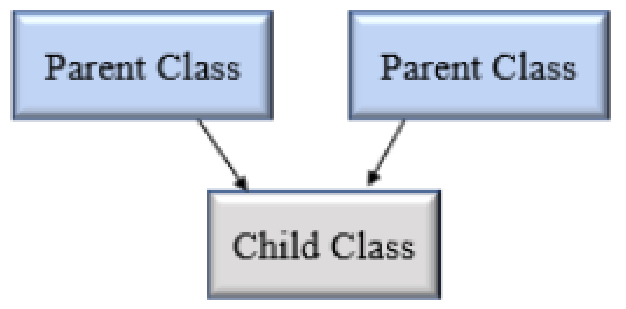
Example

Let's create one parent class called `ClassOne` and one child class called `ClassTwo` to implement 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()
```

Multiple Inheritance

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



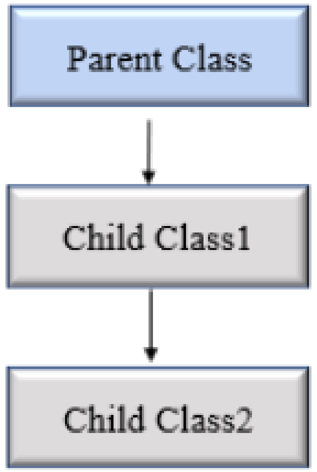
Python Multiple Inheritance

Example

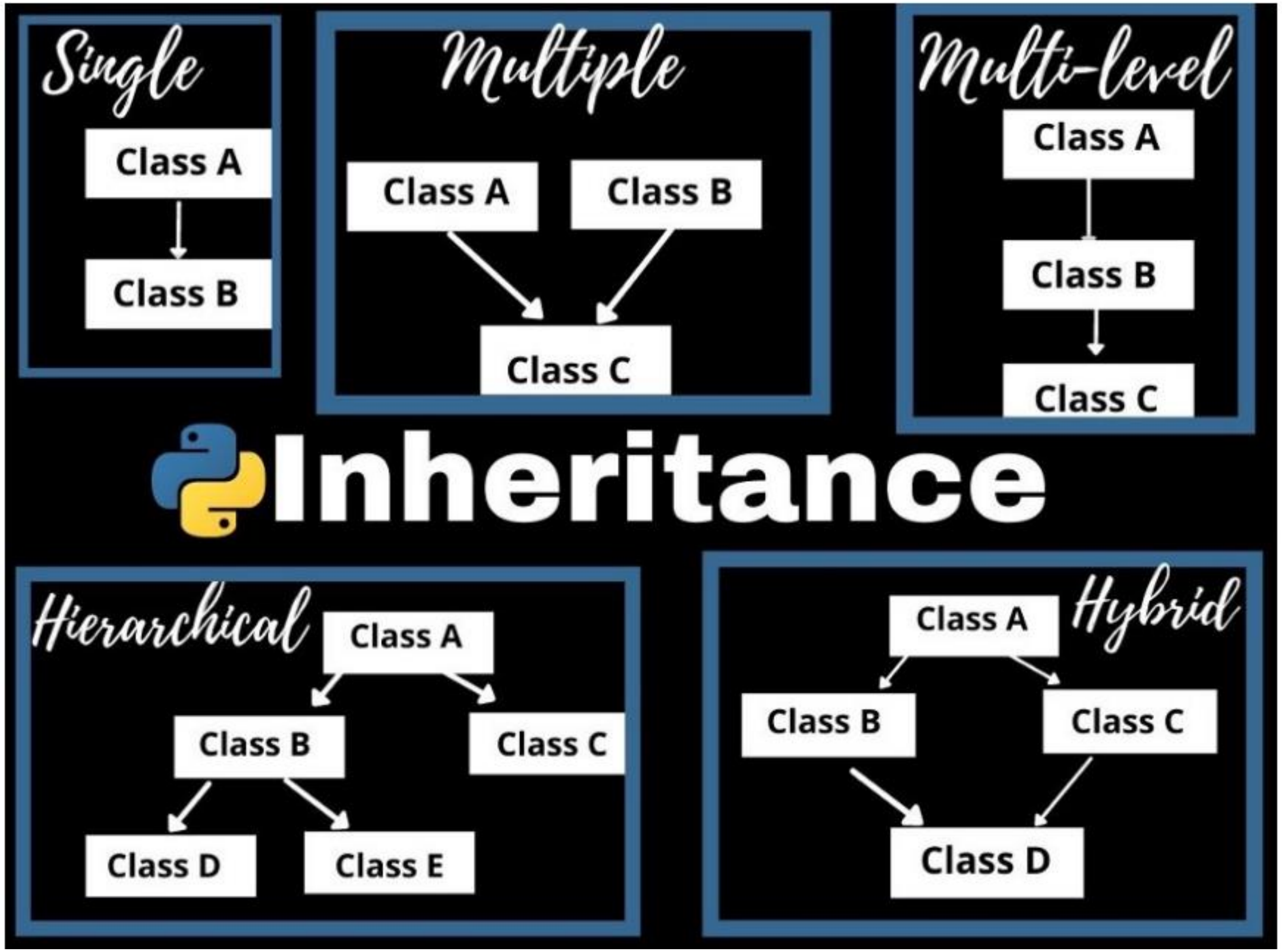
```python
# Parent class 1
class Person:
    def person_info(self, name, age):
        print('Inside Person class')
        print('Name:', name, 'Age:', age)

# 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')
```

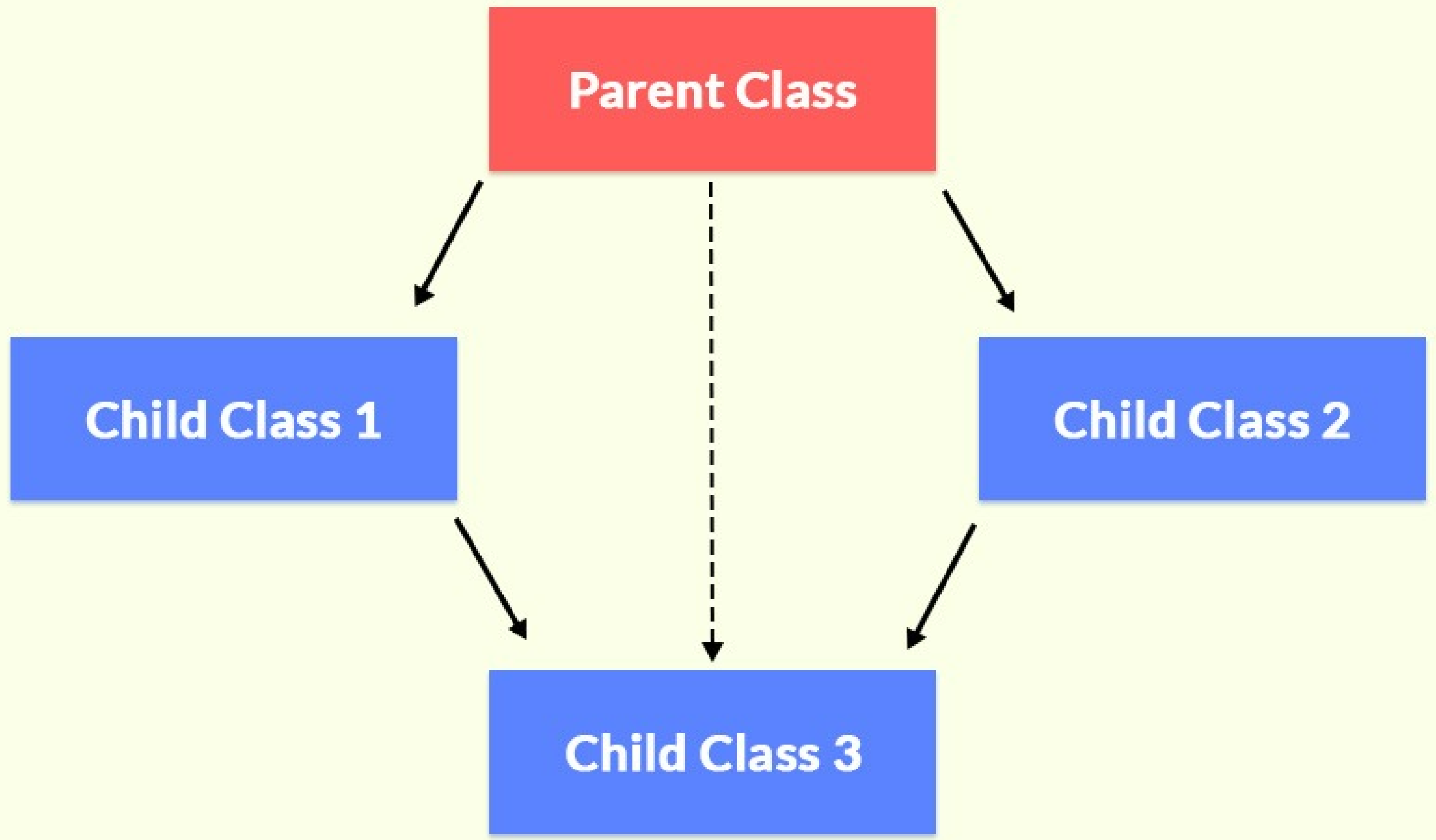
Multilevel inheritance

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.**



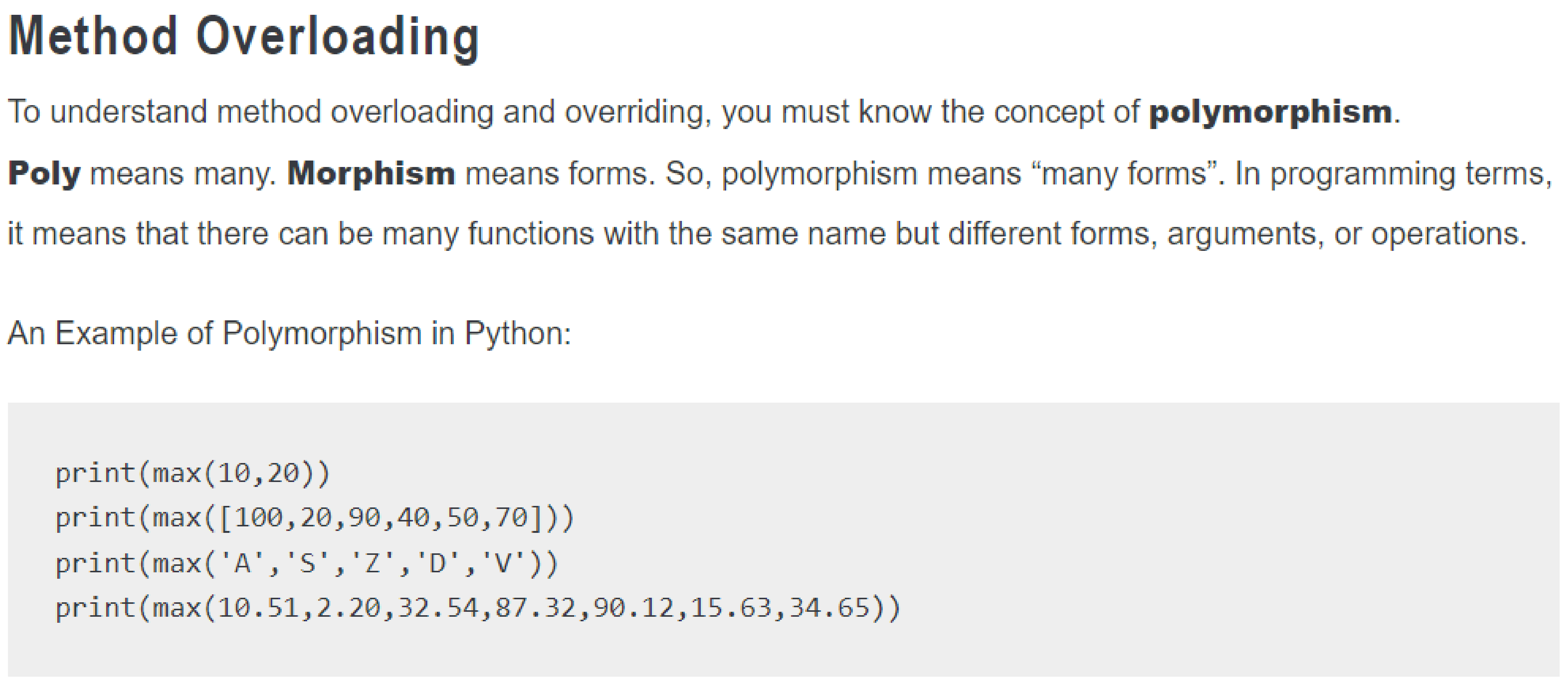
Python Multilevel Inheritance


Single
Class A
Class B
Multiple
Class A
Class B
Class C
Multi-level
Class A
Class B
Class C
Inheritance
Hierarchical
Class A
Class B
Class C
Class D
Class E
Hybrid
Class A
Class B
Class C
Class D



Python Hybrid Inheritance

Polymorphism



Method Overloading

To understand method overloading and overriding, you must know the concept of **polymorphism**.

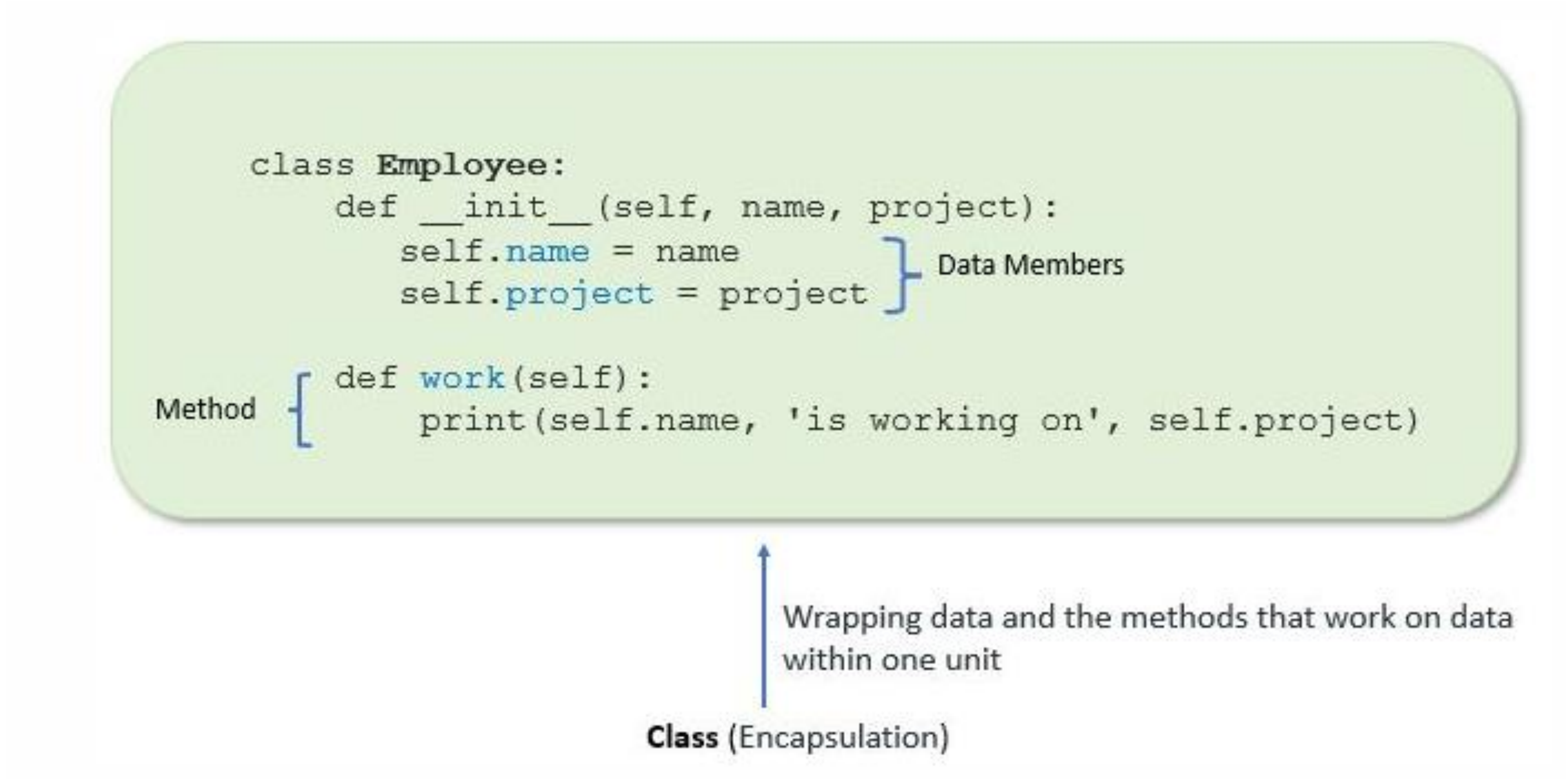
Poly means many. **Morphism** means forms. So, polymorphism means “many forms”. In programming terms, it means that there can be many functions with the same name but different forms, arguments, or operations.

An Example of Polymorphism in Python:

```
print(max(10,20))
print(max([100,20,90,40,50,70]))
print(max('A','S','Z','D','V'))
print(max(10.51,2.20,32.54,87.32,90.12,15.63,34.65))
```

What is Encapsulation in Python?

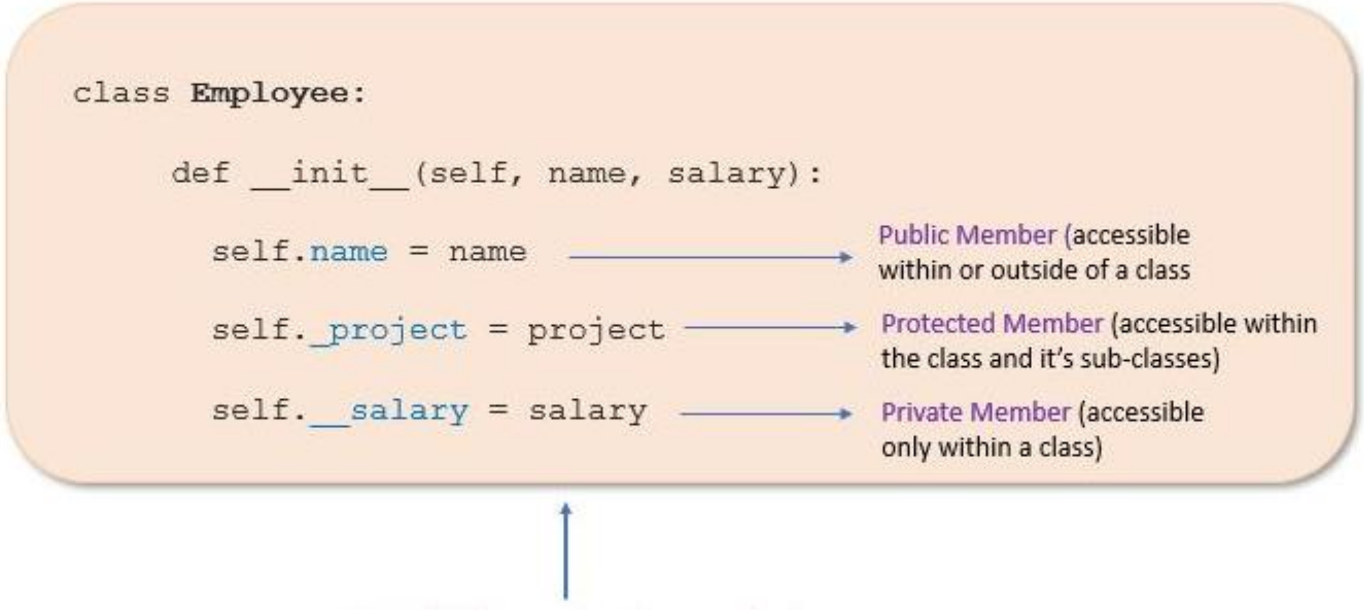
Encapsulation in Python describes the concept of bundling data and methods within a single unit. So, for example, when you create a class, it means you are implementing encapsulation. A class is an example of encapsulation as it binds all the data members (instance variables) and methods into a single unit.



```
class Employee:
    def __init__(self, name, project):
        self.name = name
        self.project = project

    def work(self):
        print(self.name, 'is working on', self.project)
```

Data Members
Method
Wrapping data and the methods that work on data within one unit
Class (Encapsulation)

- **Public Member**: Accessible anywhere from otside oclass.
- **Private Member**: Accessible within the class
- **Protected Member**: Accessible within the class and its sub-classes



```
class Employee:

    def __init__(self, name, salary):

        self.name = name
        self._project = project
        self.__salary = salary
```

Public Member (accessible within or outside of a class
Protected Member (accessible within the class and it's sub-classes)
Private Member (accessible only within a class)

Data Hiding using Encapsulation

Data hiding using access modifiers

```
class OverloadingExample:
   def add(self,a,b):
       print(a+b)
   def add(self,a,b,c):
       print(a+b+c)

a=OverloadingExample()
a.add(5,10)
a.add(5,10,20)
```

Note: Python does not support method overloading, this is because python always picks up the latest defined method. We can still overload methods in python but it would be of no use. However, you can implement method overloading in the above way in Java, C++, etc.

An alternative to performing method overloading in Python would be in this way:

```
class OverloadingExample:
    def add(self, x = None, y = None):
        if x != None and y != None:
            return x + y
        elif x != None:
            return x
        else:
            return 0

obj = OverloadingExample()

print("Value:", obj.add())
print("Value:", obj.add(4))
print("Value:", obj.add(10,20))
```

Example

```python
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()
```

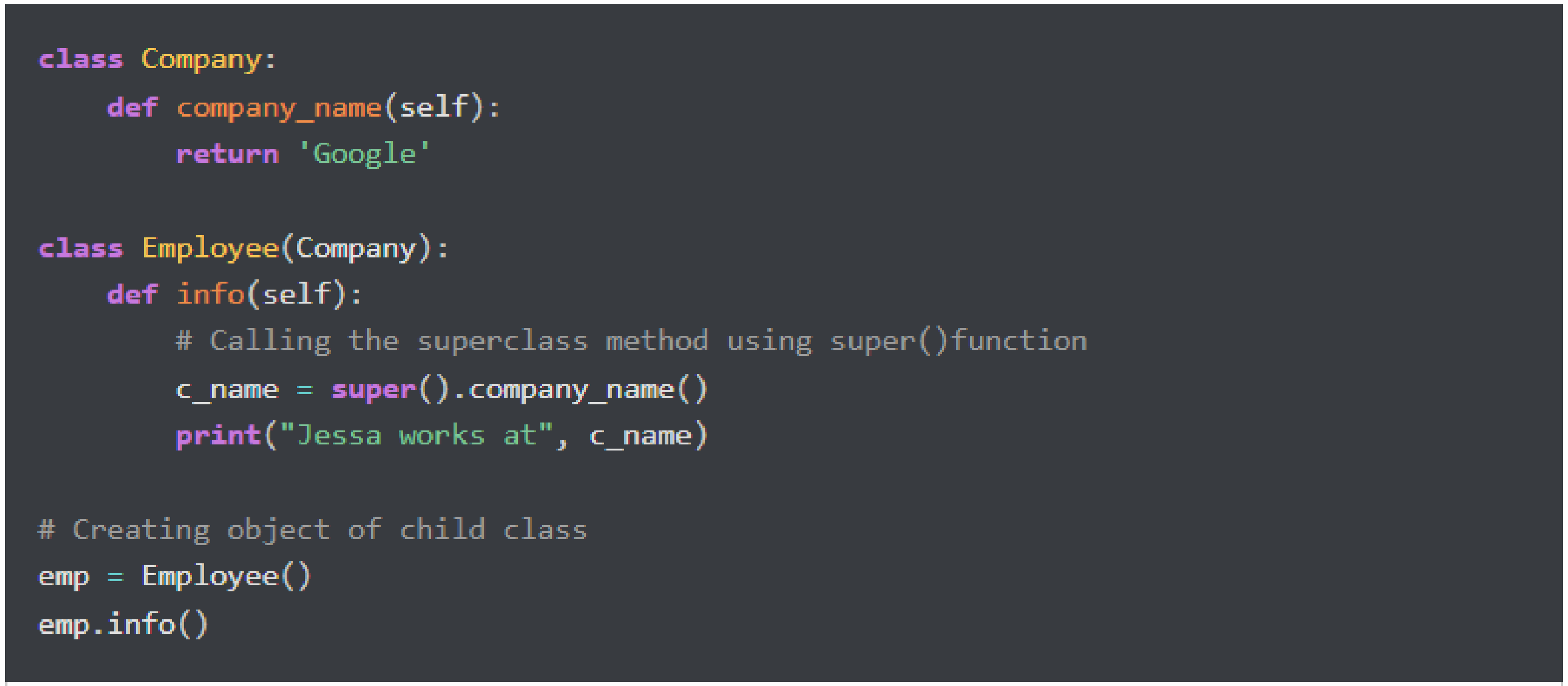
Python super() function

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.

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
Example



```python
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()
```