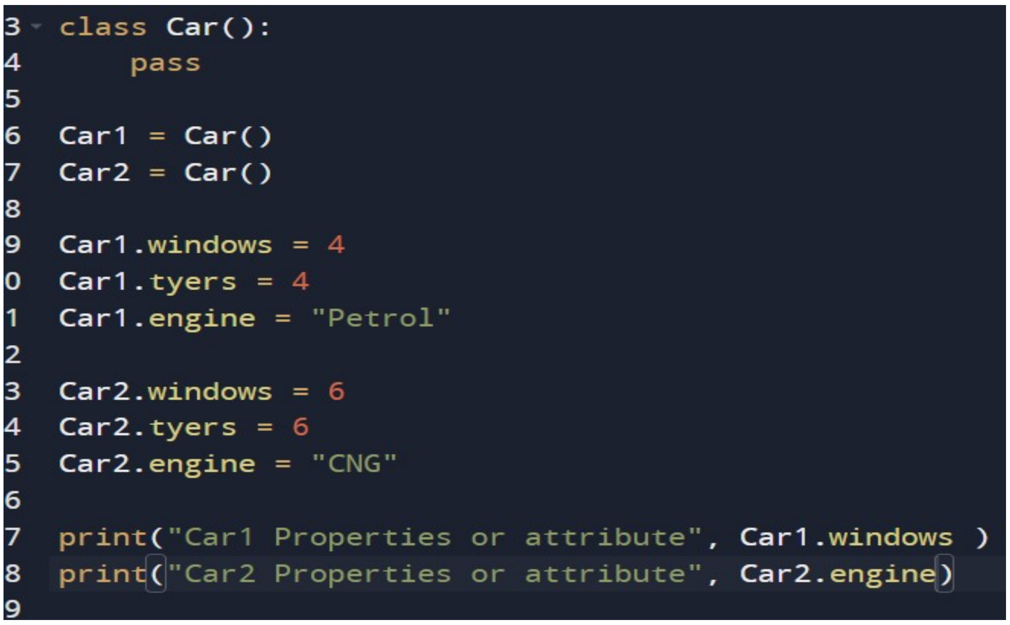
OOPS Concepts for the Interviews

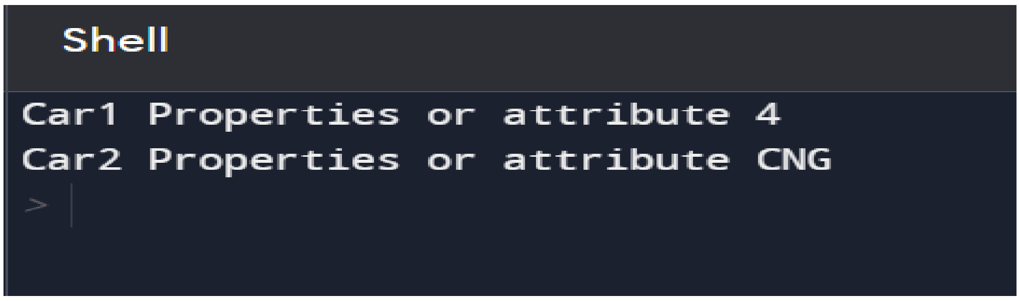
1. Classes:-

- Real World Entity or object (inside Different-2 Attribute and Method).
- Code Readability, Code Re-usability, Code Maintainability, and Modular Programming for used.

Example with code:-



```
class Car():
    pass

Car1 = Car()
Car2 = Car()

Car1.windows = 4
Car1.tyers = 4
Car1.engine = "Petrol"

Car2.windows = 6
Car2.tyers = 6
Car2.engine = "CNG"

print("Car1 Properties or attribute", Car1.windows )
print("Car2 Properties or attribute", Car2.engine)
```

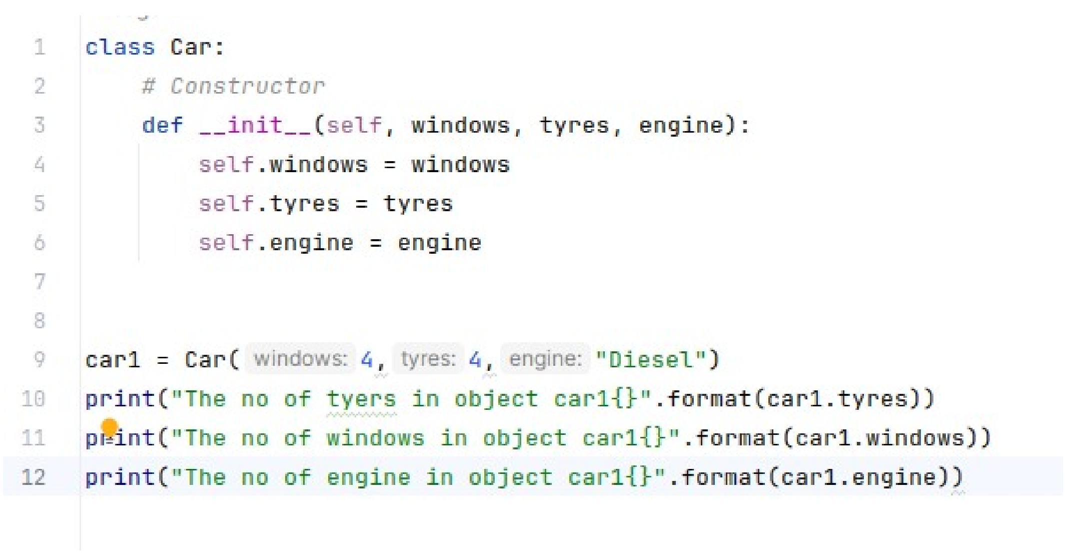


```
Car1 Properties or attribute 4
Car2 Properties or attribute CNG
>
```

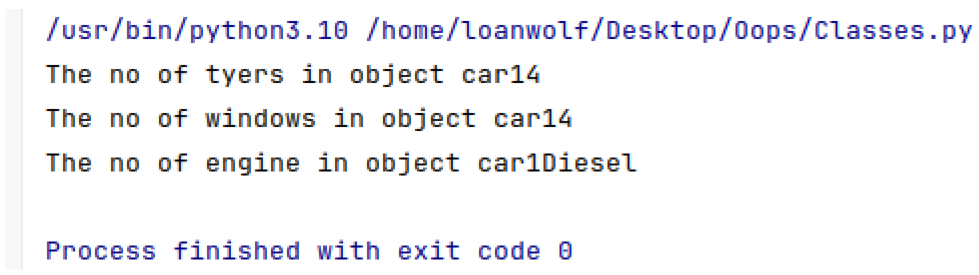
2. Constructor:-

- Constructor a better way write the classes of the code.

Example with code:-

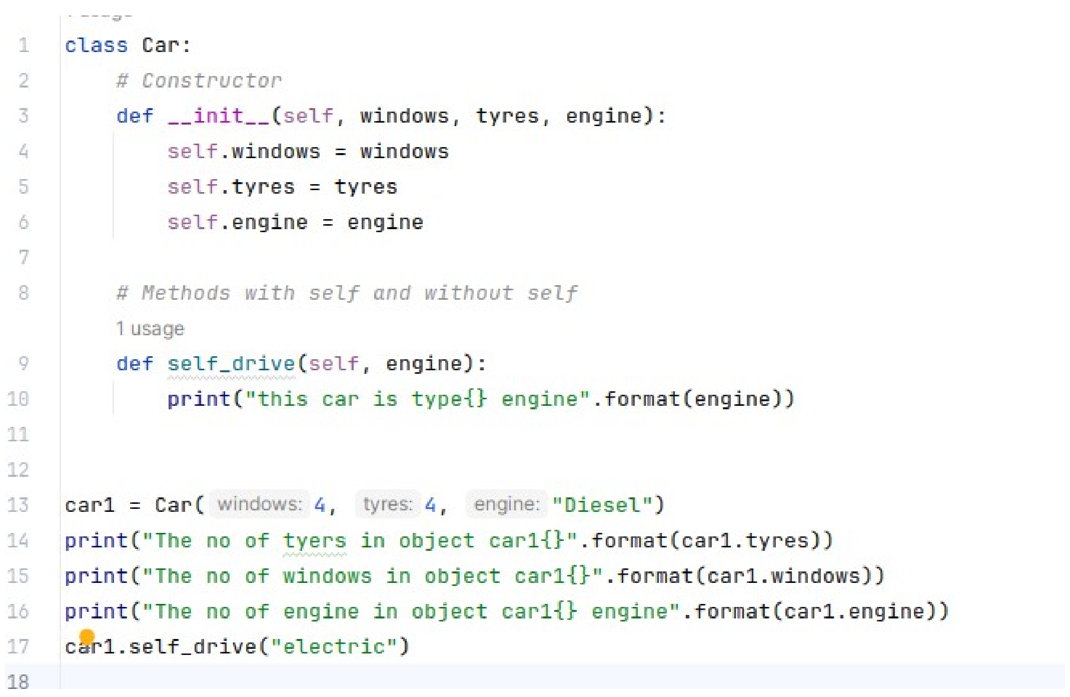


```python
class Car:
    # Constructor
    def __init__(self, windows, tyres, engine):
        self.windows = windows
        self.tyres = tyres
        self.engine = engine


car1 = Car( windows: 4, tyres: 4, engine: "Diesel")
print("The no of tyers in object car1{}".format(car1.tyres))
print("The no of windows in object car1{}".format(car1.windows))
print("The no of engine in object car1{}".format(car1.engine))
```

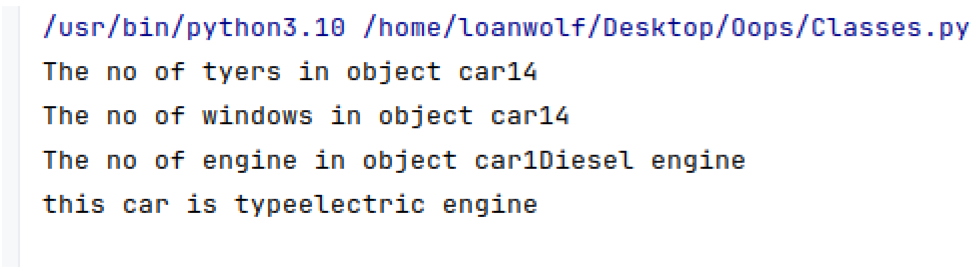


```
/usr/bin/python3.10 /home/loanwolf/Desktop/Oops/Classes.py
The no of tyers in object car14
The no of windows in object car14
The no of engine in object car1Diesel

Process finished with exit code 0
```

3. Method:-



```python
class Car:
    # Constructor
    def __init__(self, windows, tyres, engine):
        self.windows = windows
        self.tyres = tyres
        self.engine = engine

    # Methods with self and without self
    1 usage
    def self_drive(self, engine):
        print("this car is type{} engine".format(engine))


car1 = Car( windows: 4, tyres: 4, engine: "Diesel")
print("The no of tyers in object car1{}".format(car1.tyres))
print("The no of windows in object car1{}".format(car1.windows))
print("The no of engine in object car1{} engine".format(car1.engine))
car1.self_drive("electric")
```

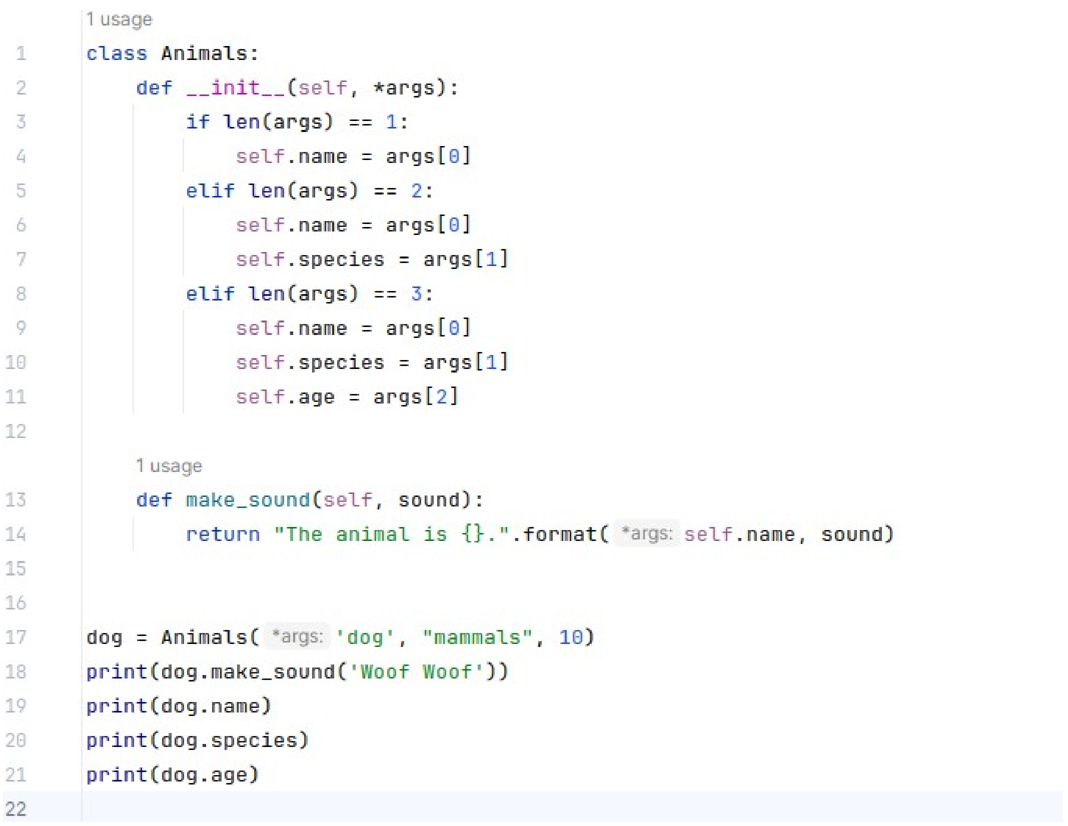


```
/usr/bin/python3.10 /home/loanwolf/Desktop/Oops/Classes.py
The no of tyers in object car14
The no of windows in object car14
The no of engine in object car1Diesel engine
this car is typeelectric engine
```

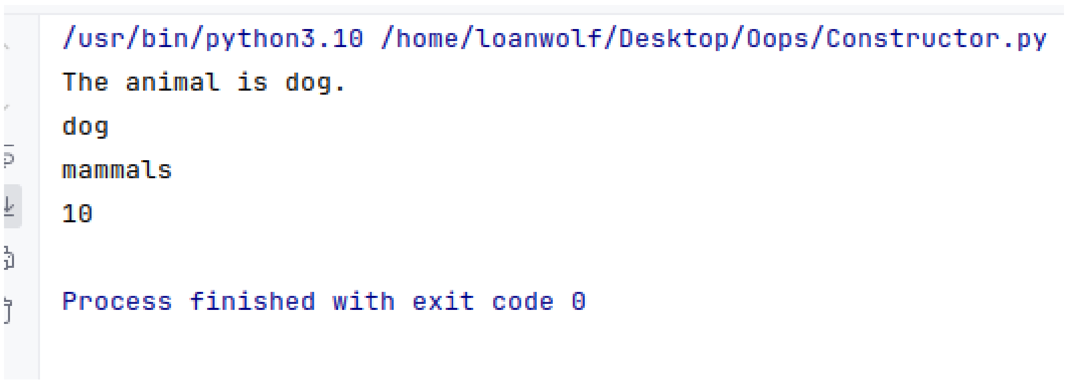
4. Multiple Constructor :-

- Multiple Constructor is not allowed because last constructor override the above constructor.
- But we write the mutliple constructor with help of Postional Arguments (*args) and Key word arguments (*kargs).

Example with code:-



```
1 usage
class Animals:
    def __init__(self, *args):
        if len(args) == 1:
            self.name = args[0]
        elif len(args) == 2:
            self.name = args[0]
            self.species = args[1]
        elif len(args) == 3:
            self.name = args[0]
            self.species = args[1]
            self.age = args[2]

    1 usage
    def make_sound(self, sound):
        return "The animal is {}.".format( *args: self.name, sound)


dog = Animals( *args: 'dog', "mammals", 10)
print(dog.make_sound('Woof Woof'))
print(dog.name)
print(dog.species)
print(dog.age)
```

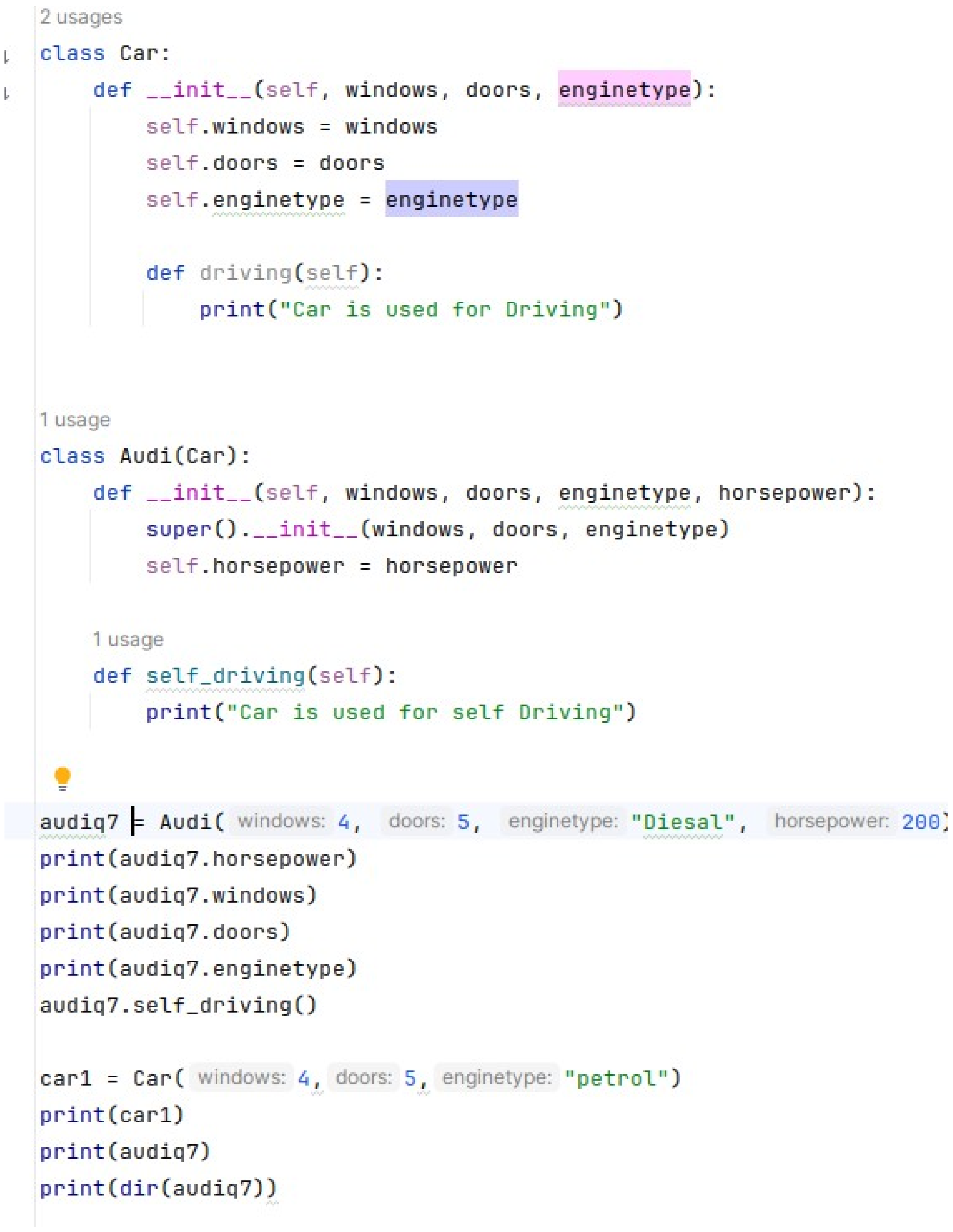


```
/usr/bin/python3.10 /home/loanwolf/Desktop/Oops/Constructor.py
The animal is dog.
dog
mammals
10

Process finished with exit code 0
```

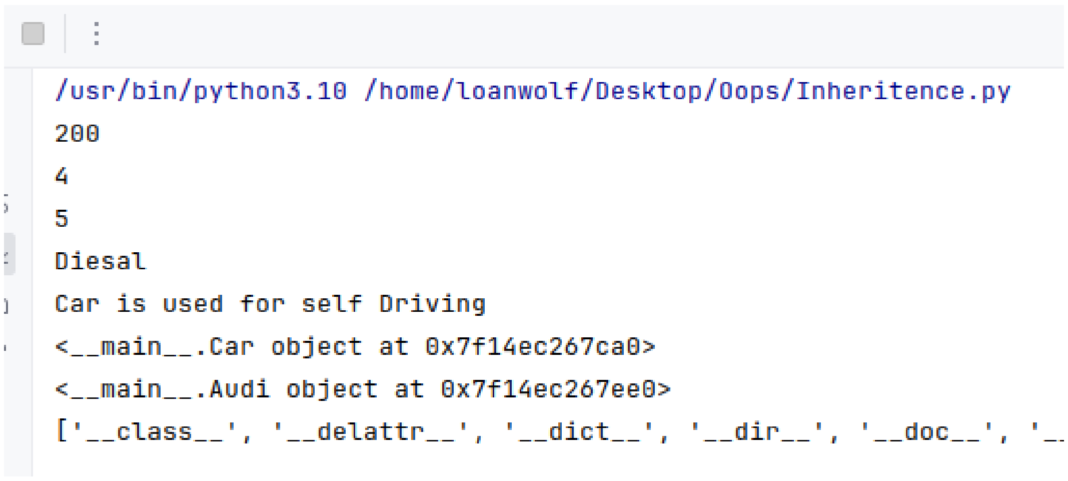
5. Inheritance :-

- In the inheritance child class is inherit the property of his parent class but parent class is not inherit property his child class.
- It provides the re-usability 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.

Example with code:-



```
2 usages
class Car:
    def __init__(self, windows, doors, enginetype):
        self.windows = windows
        self.doors = doors
        self.enginetype = enginetype

        def driving(self):
            print("Car is used for Driving")


1 usage
class Audi(Car):
    def __init__(self, windows, doors, enginetype, horsepower):
        super().__init__(windows, doors, enginetype)
        self.horsepower = horsepower

    1 usage
    def self_driving(self):
        print("Car is used for self Driving")


audiq7 = Audi( windows: 4,  doors: 5,  enginetype: "Diesal",  horsepower: 200)
print(audiq7.horsepower)
print(audiq7.windows)
print(audiq7.doors)
print(audiq7.enginetype)
audiq7.self_driving()

car1 = Car( windows: 4, doors: 5, enginetype: "petrol")
print(car1)
print(audiq7)
print(dir(audiq7))
```



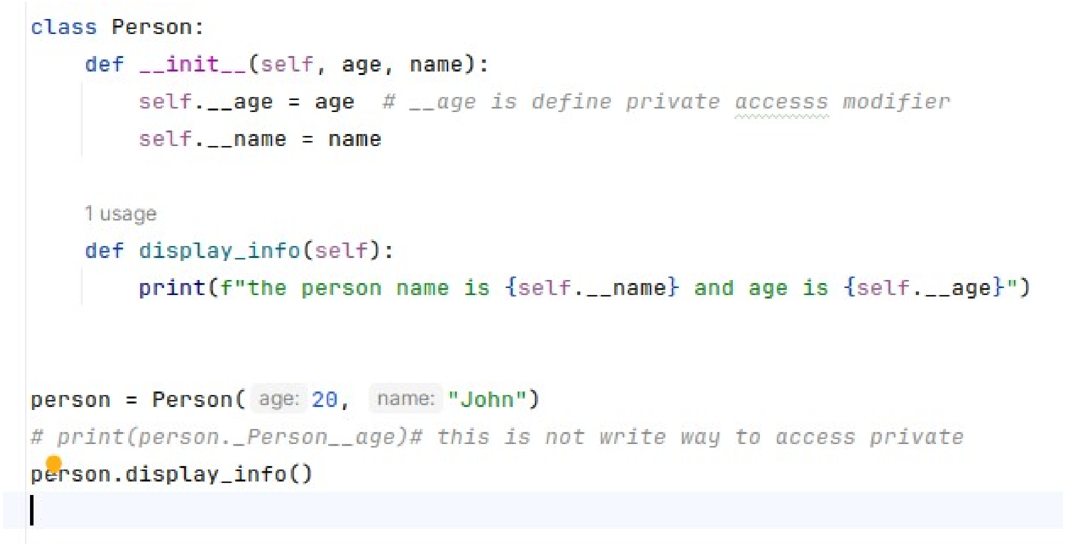
```
/usr/bin/python3.10 /home/loanwolf/Desktop/Oops/Inheritence.py
200
4
5
Diesal
Car is used for self Driving
<__main__.Car object at 0x7f14ec267ca0>
<__main__.Audi object at 0x7f14ec267ee0>
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '_
```

Types of Inheritance:-

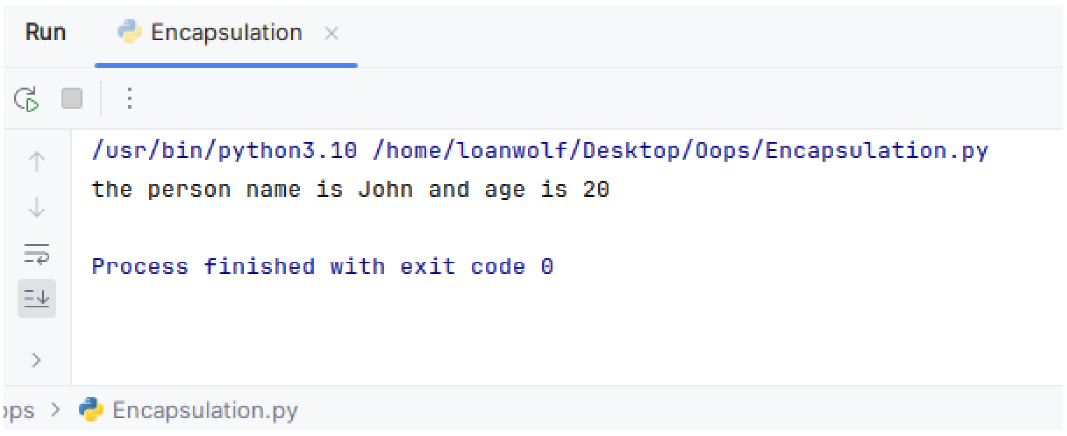
- **Single Inheritance:** Single-level inheritance enables a derived class to inherit characteristics from a single-parent class.
- **Multilevel Inheritance:** Multi-level inheritance enables a derived class to inherit properties from an immediate parent class which in turn inherits properties from his parent class.
- **Hierarchical Inheritance:** Hierarchical-level inheritance enables more than one derived class to inherit properties from a parent class.
- **Multiple Inheritance:** Multiple-level inheritance enables one derived class to inherit properties from more than one base class.

6. Encapsulation :-

Access Modifier:- Use for encapsulation(Internal functionality hidden)
Public Access Modifier:- Any where access-able class.
Protected Access Modifier:- In Drive Class variable(Parent class variable access).
Private Access Modifier:- Which access-able with in class.



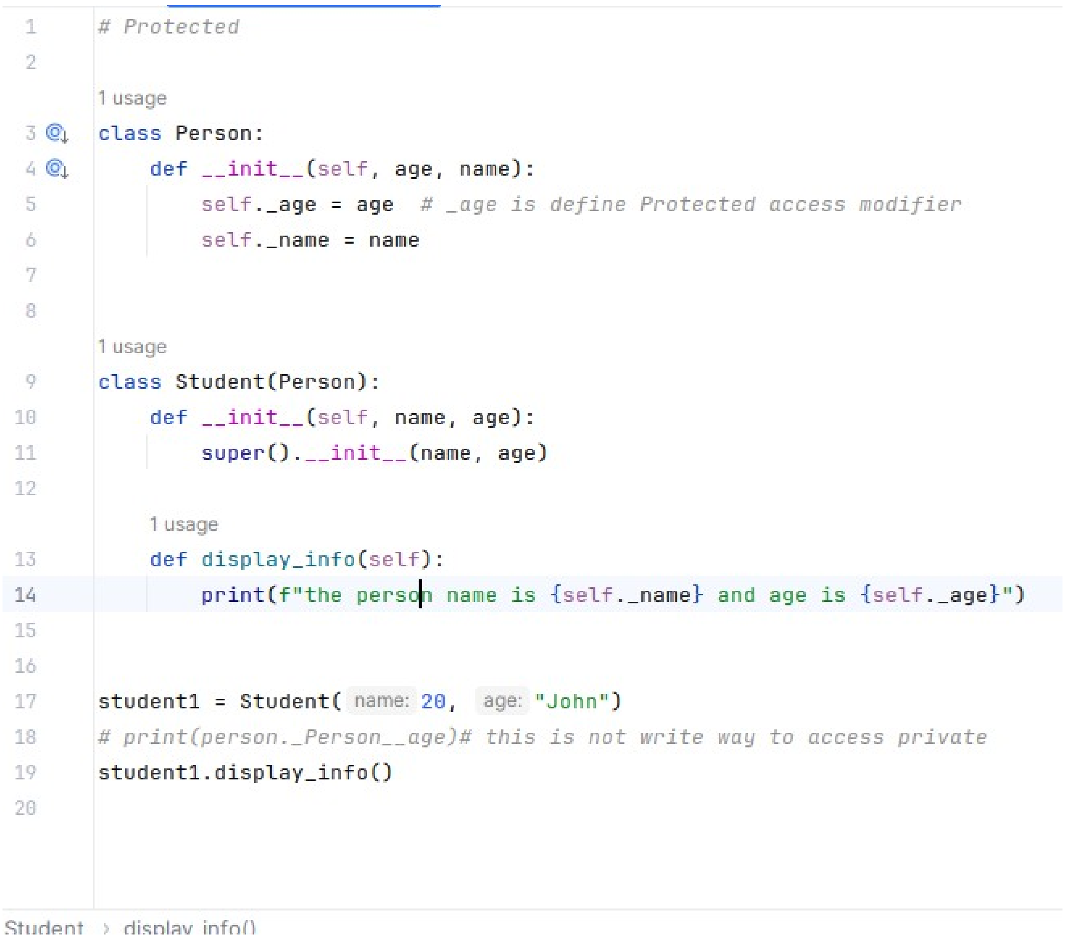
```
class Person:
    def __init__(self, age, name):
        self.__age = age  # __age is define private accesss modifier
        self.__name = name

    1 usage
    def display_info(self):
        print(f"the person name is {self.__name} and age is {self.__age}")


person = Person( age: 20,  name: "John")
# print(person._Person__age)# this is not write way to access private
person.display_info()
```


Run   Encapsulation

```
/usr/bin/python3.10 /home/loanwolf/Desktop/Oops/Encapsulation.py
the person name is John and age is 20

Process finished with exit code 0
```

ps > Encapsulation.py

Protected Access Modifier:-



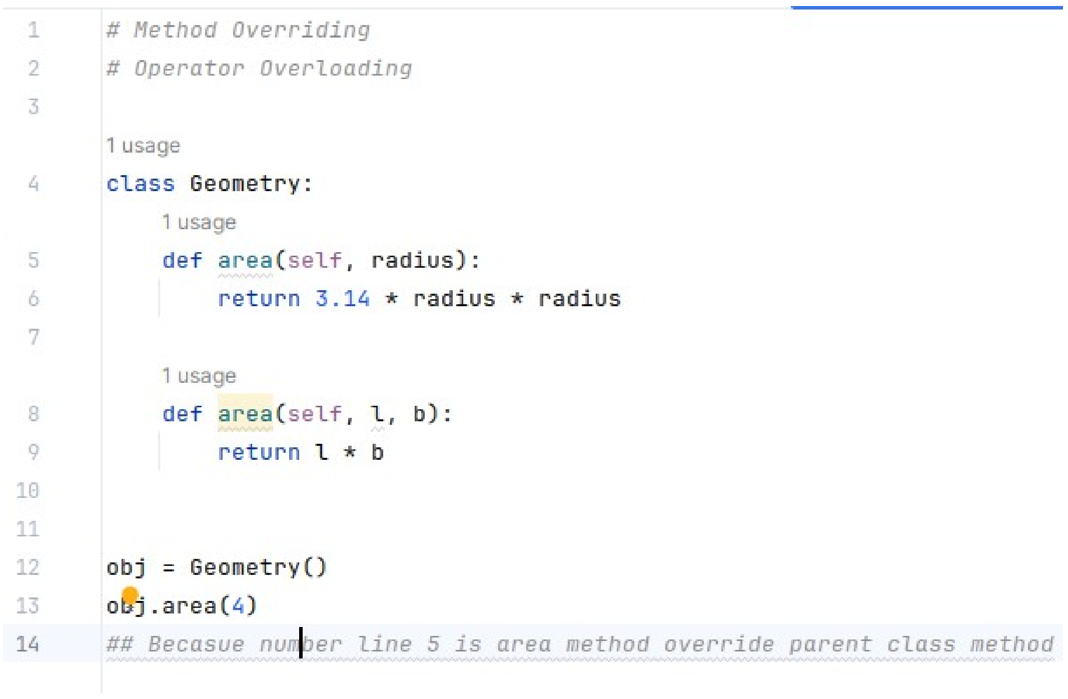
```
# Protected

1 usage
class Person:
    def __init__(self, age, name):
        self._age = age  # _age is define Protected access modifier
        self._name = name


1 usage
class Student(Person):
    def __init__(self, name, age):
        super().__init__(name, age)

    1 usage
    def display_info(self):
        print(f"the person name is {self._name} and age is {self._age}")


student1 = Student( name: 20,  age: "John")
# print(person._Person__age)# this is not write way to access private
student1.display_info()
```

Student > display_info()

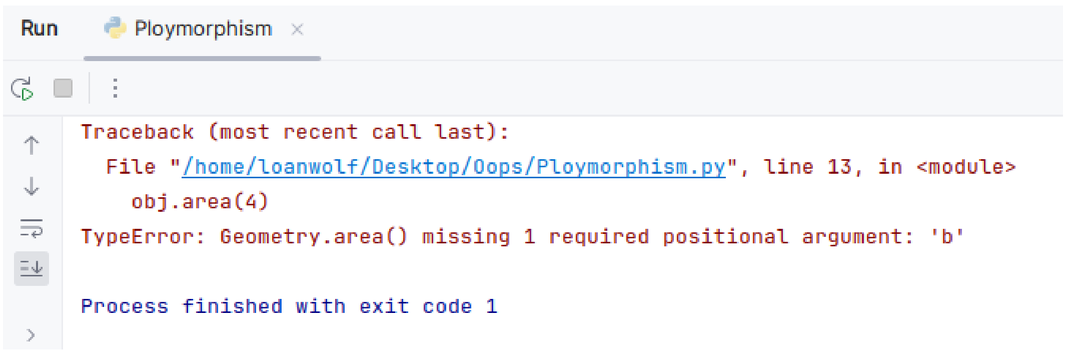
7. Polymorphism :-

This code demonstrates the concept of inheritance and method overriding in Python classes. It shows how subclasses can override methods defined in their parent class to provide specific behavior while still inheriting other methods from the parent class.

Example with code:-

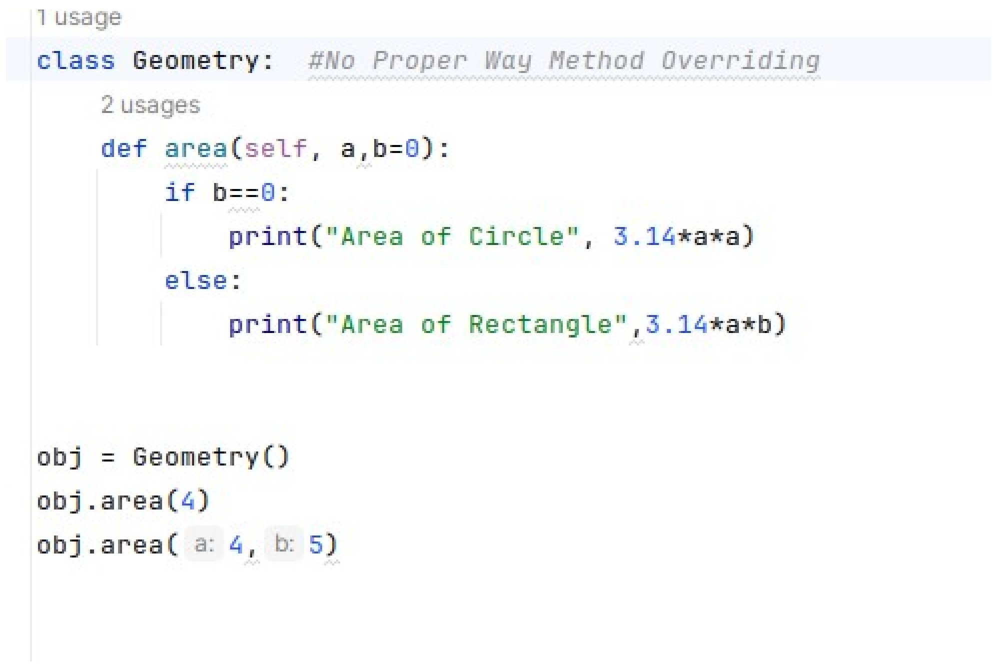


```
1   # Method Overriding
2   # Operator Overloading
3
    1 usage
4   class Geometry:
        1 usage
5       def area(self, radius):
6           return 3.14 * radius * radius
7
        1 usage
8       def area(self, l, b):
9           return l * b
10
11
12  obj = Geometry()
13  obj.area(4)
14  ## Becasue number line 5 is area method override parent class method
```

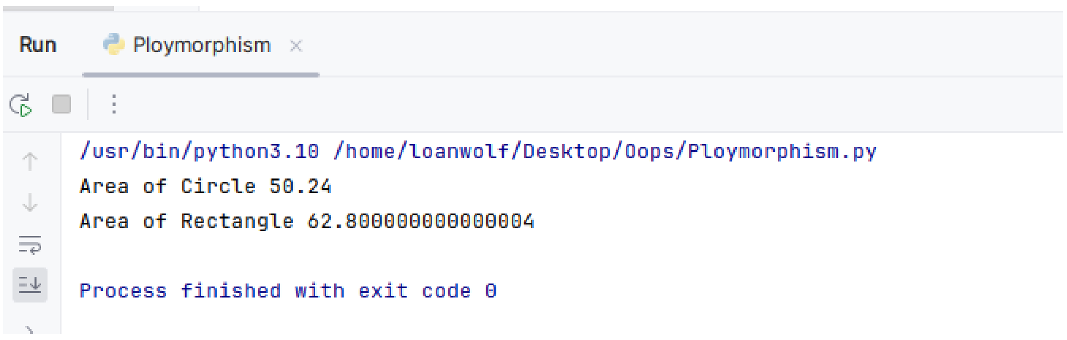


Run   Ploymorphism ×

```
Traceback (most recent call last):
  File "/home/loanwolf/Desktop/Oops/Ploymorphism.py", line 13, in <module>
    obj.area(4)
TypeError: Geometry.area() missing 1 required positional argument: 'b'

Process finished with exit code 1
```



```
1 usage
class Geometry:  #No Proper Way Method Overriding
    2 usages
    def area(self, a,b=0):
        if b==0:
            print("Area of Circle", 3.14*a*a)
        else:
            print("Area of Rectangle",3.14*a*b)


obj = Geometry()
obj.area(4)
obj.area( a: 4, b: 5)
```



```
/usr/bin/python3.10 /home/loanwolf/Desktop/Oops/Ploymorphism.py
Area of Circle 50.24
Area of Rectangle 62.800000000000004

Process finished with exit code 0
```

8. <u>Data Abstraction :-</u>

It hides unnecessary code details from the user. Also, when we do not want to give out sensitive parts of our code implementation and this is where data abstraction came.