CMPE 103

Module 5

INHERITANCE & POLYMORPHISM

At the end of the module, you should be able to:

- define Inheritance and Polymorphism
- discuss the syntax and program structure of Inheritance
- demonstrate application programs on inheritance
- discuss the types of inheritance
- demonstrate application program on the different type of inheritance
- demonstrate application programs on polymorphism
- discuss the concept of overloading and Overriding
- demonstrate application programs on overloading and overriding

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.

- 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

***For example**, In the real world, Car is a sub-class of a Vehicle class. We can create a Car by inheriting the properties of a Vehicle such as Wheels, Colors, Fuel tank, engine, and add extra properties in Car as required.*

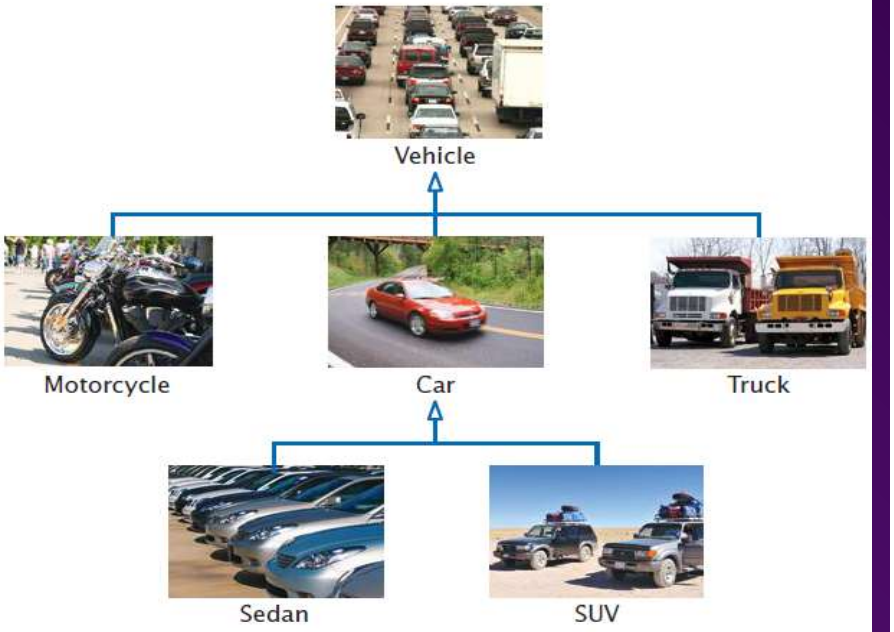
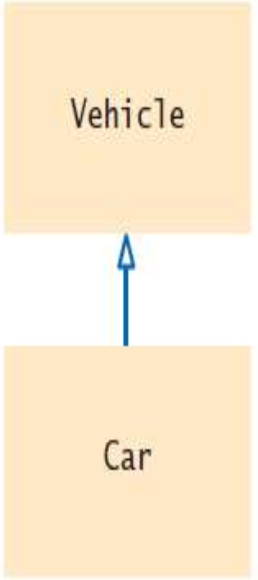
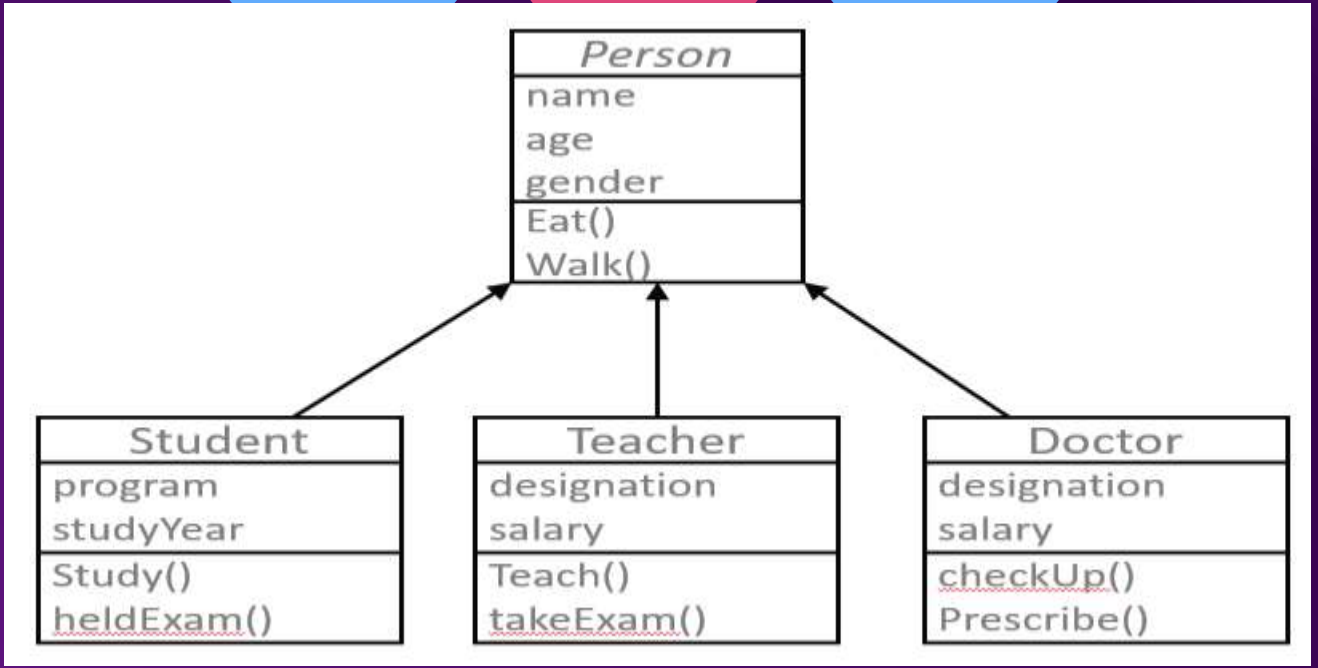


Figure 1 An Inheritance Hierarchy of Vehicle Classes

Figure 2
An Inheritance Diagram



Inheritance – "IS A" or "IS A KIND OF" Relationship

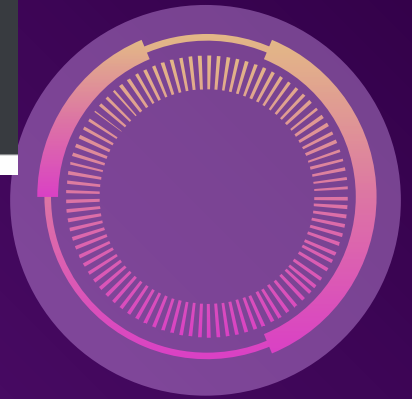


Example – "IS A" Relationship

How to implement Inheritance in Python

Syntax

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



Create a Parent Class

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

'''Use the Person class to create an
object, and then execute the printname
method'''

x = Person("Jose", "Rizal")
x.printname()
```

Create a Child Class

```
class Student(Person):
  pass

'''Use the Student class to create an object,
and then execute the printname method'''

x = Student("Melchora", "Aquino")
x.printname()
```

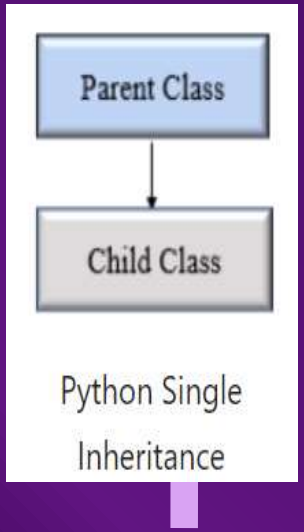
Types Of Inheritance

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:

- Single inheritance
- Multiple Inheritance
- Multilevel inheritance
- Hierarchical Inheritance
- 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.



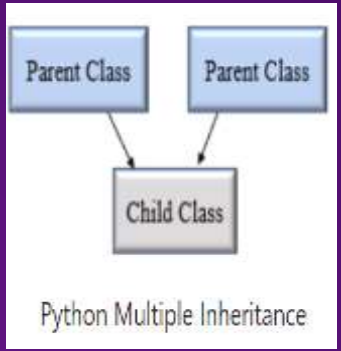

```python
# Base class
class Vehicle:
    def Vehicle_info(self):
        print('Inside Vehicle class')

# 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()
```

Output:

```
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.



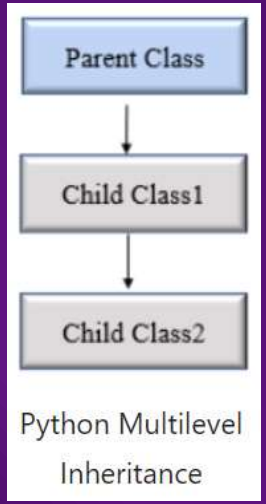

```python
# Parent class 1
class Person:
    def person_info(self, name, age):
        print('Inside Person class')
        print('Name:', name, 'Age:', age)

# 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')
```

Output:

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

Inside Company class
Name: Google location: Atlanta

Inside Employee class
Salary: 12000 Skill: 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.**



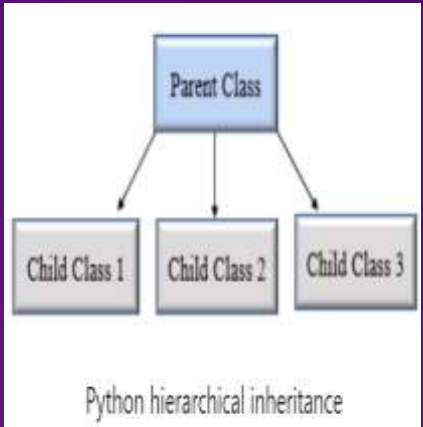

```python
# Base class
class Vehicle:
    def Vehicle_info(self):
        print('Inside Vehicle class')

# 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()
```

Output:

```
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.



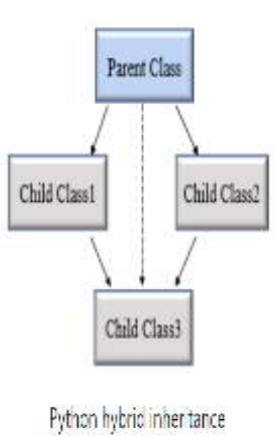

```python
class Vehicle:
    def info(self):
        print("This is Vehicle")

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')
```

Output:

```
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.




```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()
```

Output:

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

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 sub class and the latter class is the parent class. In child class, we can refer to parent class by using the `super()` function.

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*

```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()
```

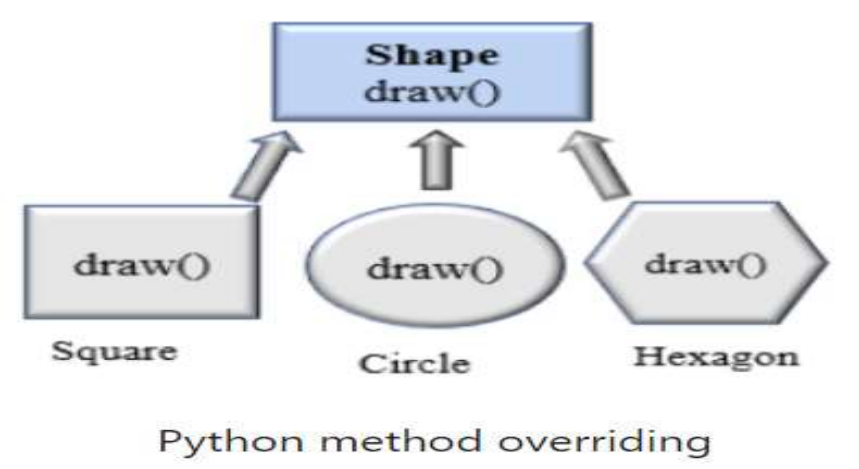
Output:

```
Jessa works at Google
```

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.



Python method overriding

```python
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()
```

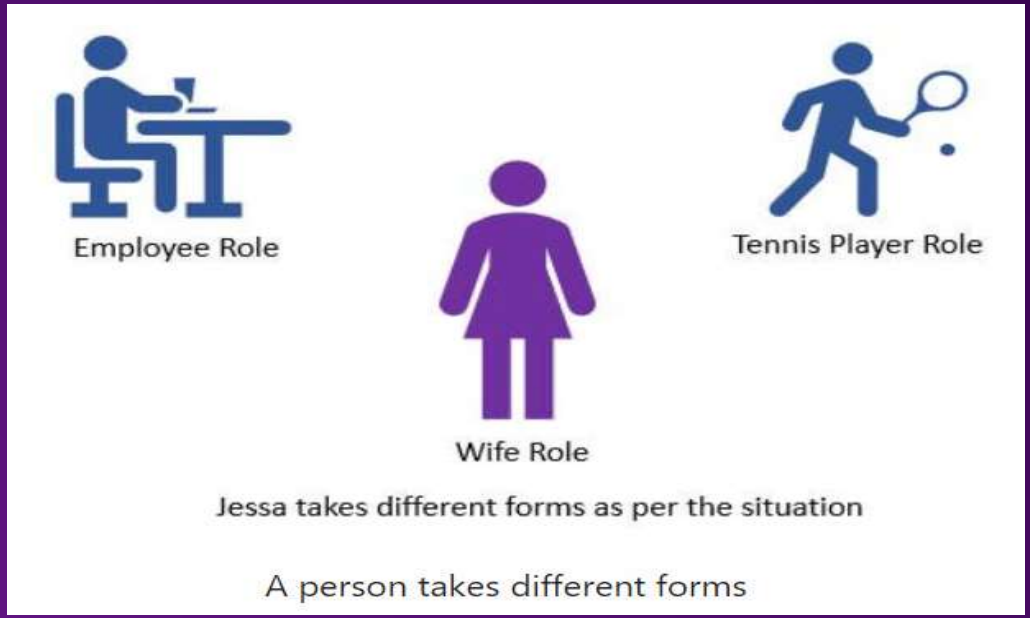
Output:

```
max speed is 200 Km/Hour
```

In the above example, we create two classes named Vehicle (Parent Class) and Car (Child Class) The class Car extends from the class Vehicle so, all properties of the parent class are available in the child class. In addition to that, the child class redefined the method max_speed().

What is Polymorphism in Python?

Polymorphism in Python is the ability of an object to take many forms. In simple words, polymorphism allows us to perform the same action in many different ways. For example, Jessa acts as an employee when she is at the office. However, when she is at home, she acts like a wife. Also, she represents herself differently in different places. Therefore, the same person takes different forms as per the situation.



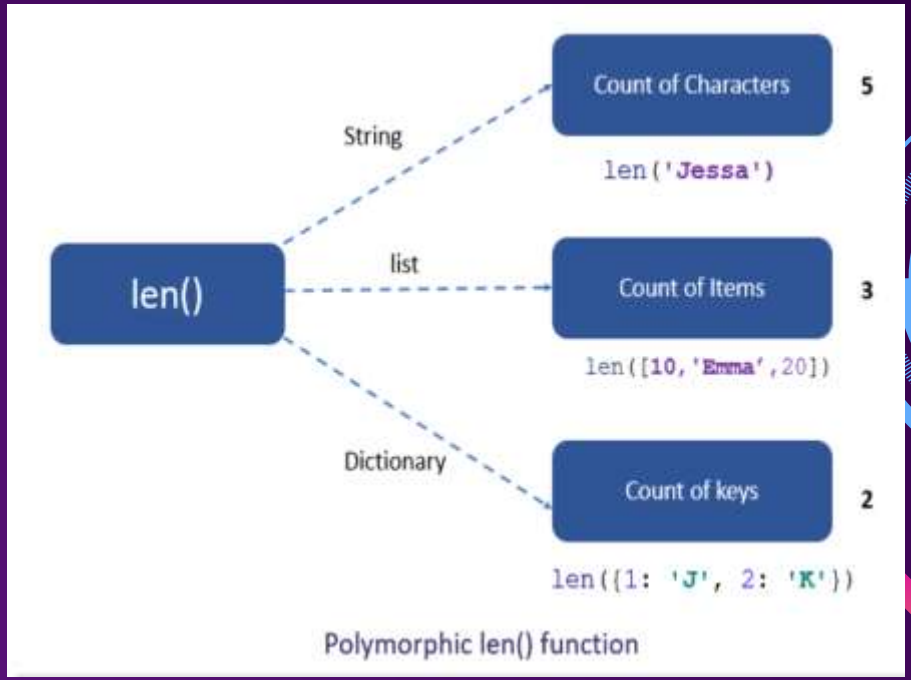
Polymorphism in Built-in function len()

The built-in function len() calculates the length of an object depending upon its type. If an object is a string, it returns the count of characters, and If an object is a list, it returns the count of items in a list.

```
students = ['Emma', 'Jessa', 'Kelly']
school = 'ABC School'

# calculate count
print(len(students))
print(len(school))
```

Output:

```
3
10
```



Polymorphic len() function

Polymorphism With Inheritance

Polymorphism is mainly used with inheritance. In **inheritance**, child class inherits the attributes and methods of a parent class. 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.

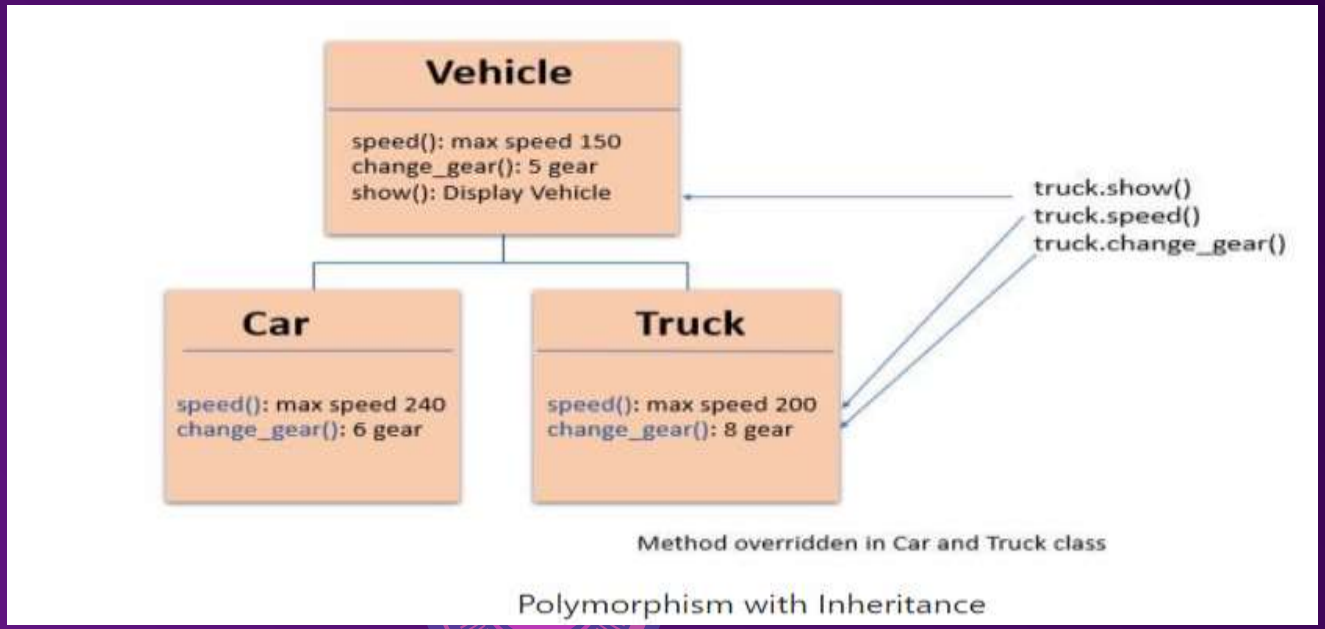
Using **method overriding** polymorphism allows us to defines methods in the child class that have the same name as the methods in the parent class. This **process of re-implementing the inherited method in the child class** is known as Method Overriding.

Advantage of method overriding

- It is effective when we want to extend the functionality by altering the inherited method. Or the method inherited from the parent class doesn't fulfill the need of a child class, so we need to re-implement the same method in the child class in a different way.
- Method overriding is useful when a parent class has multiple child classes, and one of that child class wants to redefine the method. The other child classes can use the parent class method. Due to this, we don't need to modification the parent class code

In polymorphism, **Python first checks the object's class type and executes the appropriate method** when we call the method.

Example of Polymorphism in Inheritance



```python
class Vehicle:

    def __init__(self, name, color, price):
        self.name = name
        self.color = color
        self.price = price

    def show(self):
        print('Details:', self.name, self.color, self.price)

    def max_speed(self):
        print('Vehicle max speed is 150')

    def change_gear(self):
        print('Vehicle change 6 gear')


# inherit from vehicle class
class Car(Vehicle):
    def max_speed(self):
        print('Car max speed is 240')

    def change_gear(self):
        print('Car change 7 gear')


# Car Object
car = Car('Car x1', 'Red', 20000)
car.show()
# calls methods from Car class
car.max_speed()
car.change_gear()

# Vehicle Object
vehicle = Vehicle('Truck x1', 'white', 75000)
vehicle.show()
# calls method from a Vehicle class
vehicle.max_speed()
vehicle.change_gear()
```

Output:

```
Details: Car x1 Red 20000
Car max speed is 240
Car change 7 gear

Details: Truck x1 white 75000
Vehicle max speed is 150
Vehicle change 6 gear
```

Method Overloading

The process of calling the same method with different parameters is known as method overloading. Python does not support method overloading. Python considers only the latest defined method even if you overload the method. Python will raise a TypeError if you overload the method.

```
def addition(a, b):
    c = a + b
    print(c)


def addition(a, b, c):
    d = a + b + c
    print(d)


# the below line shows an error
# addition(4, 5)

# This line will call the second product method
addition(3, 7, 5)
```

```
class Shape:
    # function with two default parameters
    def area(self, a, b=0):
        if b > 0:
            print('Area of Rectangle is:', a * b)
        else:
            print('Area of Square is:', a ** 2)

square = Shape()
square.area(5)

rectangle = Shape()
rectangle.area(5, 3)
```

Thanks!

Do you have any questions?

dcoe_chair@pup.edu.ph