



Department of Computer Science and Engineering (Data Science)
JHANVI PAREKH

60009210033

CSE(Data Science)

A-1

Experiment No 2

AIM: Implement Object Oriented Programming Concepts using Python Theory:

Object Oriented Programing

Object-oriented programming (OOP) is a computer programming model that organizes software design around data, or objects, rather than functions and logic. This includes programs for manufacturing and design, as well as mobile applications; for example, OOP can be used for manufacturing system simulation software.

Object Oriented Programming in Python

Python is a multi-paradigm programming language. It supports different programming approaches. One of the popular approaches to solve a programming problem is by creating objects. This is known as Object-Oriented Programming (OOP).

An object has two characteristics:

attributes behavior

Let's take an example:

A parrot is an object, as it has the following properties:

name, age, color as attributes singing,

dancing as behavior

The concept of OOP in Python focuses on creating reusable code. This concept is also known as DRY (Don't Repeat Yourself).

In Python, the concept of OOP follows some basic principles:

Class

A class is a blueprint for the object.

We can think of class as a sketch of a parrot with labels. It contains all the details about the name, colors, size etc. Based on these descriptions, we can study about the parrot. Here, a parrot is an object.

The example for class of parrot can be:

class Parrot:



Department of Computer Science and Engineering (Data Science)

pass

Here, we use the class keyword to define an empty class Parrot. From class, we construct instances. An instance is a specific object created from a particular class.

Object

An object (instance) is an instantiation of a class. When class is defined, only the description for the object is defined. Therefore, no memory or storage is allocated.

The example for object of parrot class can be:

```
obj = Parrot()
```

Here, obj is an object of class Parrot.

Suppose we have details of parrots. Now, we are going to show how to build the class and objects of parrots.

Example 1: Creating Class and Object in Python

```
class Parrot:    # class attribute    species = "bird"

# instance attribute    def __init__(self, name,
age):

        self.name = name

self.age = age

# instantiate the Parrot class blu = Parrot("Blu", 10)

woo = Parrot("Woo", 15) # access the class attributes

print("Blu is a {}".format(blu.__class__.species))

print("Woo is also a

{}".format(woo.__class__.species))

# access the instance attributes print("{} is {} years

old".format( blu.name, blu.age)) print("{} is {} years

old".format( woo.name, woo.age))
```

Output

Blu is a bird

Woo is also a bird

Blu is 10 years old

Woo is 15 years old



Department of Computer Science and Engineering (Data Science)

Methods

Methods are functions defined inside the body of a class. They are used to define the behaviors of an object.

Example 2 : Creating Methods in Python class

Parrot:

```
# instance attributes    def
__init__(self, name, age):
    self.name = name
self.age = age

# instance method
def sing(self, song):
    return "{} sings {}".format(self.name, song)=
def dance(self):
    return "{} is now dancing".format(self.name)

# instantiate the object blu
= Parrot("Blu", 10) # call

our instance methods
print(blu.sing("Happy"))
print(blu.dance())
```

Output

Blu sings 'Happy'

Blu is now dancing

In the above program, we define two methods i.e `sing()` and `dance()`. These are called instance methods because they are called on an instance object i.e `blu`.

Inheritance

Inheritance is a way of creating a new class for using details of an existing class without modifying it. The newly formed class is a derived class (or child class). Similarly, the existing class is a base class (or parent class).



Department of Computer Science and Engineering (Data Science)

Example 3: Use of Inheritance in Python

parent class class Bird:

```
def __init__(self):
```

```
    print("Bird is ready")    def
```

```
    whoisThis(self):
```

```
        print("Bird")    def
```

```
        swim(self):        print("Swim
```

```
        faster") # child class class
```

```
Penguin(Bird):    def
```

```
    __init__(self):    #
```

callsuper() function

```
    super().__init__()
```

```
    print("Penguin is ready")
```

```
    def whoisThis(self):
```

```
        print("Penguin")    def
```

```
        run(self):        print("Run
```

```
        faster") peggy =
```

```
Penguin()
```

```
peggy.whoisThis()
```

```
peggy.swim() peggy.run()
```

Output

Bird is ready

Penguin is ready

Penguin

Swim faster

Run faster

Penguin (child class).

In the above program, we created two classes i.e. Bird (parent class) and

The child class inherits the functions of parent class. We can see this from the swim() method.

Again, the child class modified the behavior of the parent class. We can see this from the whoisThis() method. Furthermore, we extend the functions of the parent class, by creating a new run() method.



Department of Computer Science and Engineering (Data Science)

Additionally, we use the `super()` function inside the `__init__()` method. This allows us to run the `__init__()` method of the parent class inside the child class.

Encapsulation

Using OOP in Python, we can restrict access to methods and variables. This prevents data from direct modification which is called encapsulation. In Python, we denote private attributes using underscore as the prefix i.e single `_` or double `__`.

E class Computer:

```
def __init__(self):
    self._maxprice = 900

def sell(self):
    print("Selling Price: {}".format(self._maxprice))

def setMaxPrice(self, price):
    self._maxprice = price

c = Computer()
c.sell()

# change the price
c._maxprice = 1000
c.sell()

# using setter function
c.setMaxPrice(1000)
c.sell()
```

Output

Selling Price: 900

Selling Price: 900

Selling Price: 1000

Example 4: Data Encapsulation in Python In the above program, we defined a `Computer` class.

We used `__init__()` method to store the maximum selling price of `Computer`. Here, notice the code

```
c._maxprice = 1000
```

Here, we have tried to modify the value of `__maxprice` outside of the class. However, since `__maxprice` is a private variable, this modification is not seen on the output.



Department of Computer Science and Engineering (Data Science)

As shown, to change the value, we have to use a setter function i.e `setMaxPrice()` which takes price as a parameter.

Polymorphism

Polymorphism is an ability (in OOP) to use a common interface for multiple forms (data types).

Suppose, we need to color a shape, there are multiple shape options (rectangle, square, circle). However, we could use the same method to color any shape. This concept is called Polymorphism.

Example 5: Using Polymorphism in Python

```
class Parrot:    def fly(self):
    print("Parrot can fly")
    def swim(self):
    print("Parrot can't swim")
class Penguin:    def fly(self):
    print("Penguin can't fly")
    def swim(self):
    print("Penguin can swim")
# common interface
def flying_test(bird):
    bird.fly() #instantiate
objects blu = Parrot()

peggy = Penguin()
```

```
# passing the object
```

```
flying_test(blu)
```

```
flying_test(peggy)
```

Output

Parrot can fly

Penguin can't fly

Key Points to Remember:



Department of Computer Science and Engineering (Data Science)

Object-Oriented Programming makes the program easy to understand as well as efficient.

Since the class is sharable, the code can be reused.

Data is safe and secure with data abstraction.

Polymorphism allows the same interface for different objects, so programmers can write efficient code.

Lab Assignments to complete in this session Exercise 1: Implement following programs using Python Inheritance

1.Create a Vehicle class without any variables and methods

```
a=10
b=10
print(id(a))
print(id(b))

9793376
9793376

[3] class Vehicle:
    pass

class Vehicle:
    def __init__(self,max_speed,mileage):
        self.max_speed = max_speed
        self.mileage = mileage

v1=Vehicle(150,2100)
print(v1.max_speed)
print(v1.mileage)

150
2100

[6] class Vehicle:
    def __init__(self,name,max_speed,mileage):
        self.name = name
        self.max_speed = max_speed
        self.mileage = mileage
```




Department of Computer Science and Engineering (Data Science)

2. Create a Class with instance attributes

Write a Python program to create a Vehicle class with `max_speed` and `mileage` instance attributes.

```
a=10
b=10
print(id(a))
print(id(b))

9793376
9793376

[3] class Vehicle:
    pass

class Vehicle:
    def __init__(self,max_speed,mileage):
        self.max_speed = max_speed
        self.mileage = mileage

v1=Vehicle(150,2100)
print(v1.max_speed)
print(v1.mileage)

150
2100

[6] class Vehicle:
    def __init__(self,name,max_speed,mileage):
        self.name = name
        self.max_speed = max_speed
        self.mileage = mileage
```

3. Create a child class Bus that will inherit all of the variables and methods of the Vehicle class

```
class Vehicle:
    def __init__(self, name,
max_speed, mileage):
```




Department of Computer Science and Engineering (Data Science)

```
self.name = name      self.max_speed =  
max_speed             self.mileage = mileage
```

```
[4] class Vehicle:  
    def __init__(self,max_speed,mileage):  
        self.max_speed = max_speed  
        self.mileage = mileage  
  
    v1=Vehicle(150,2100)  
    print(v1.max_speed)  
    print(v1.mileage)  
  
150  
2100  
  
class Vehicle:  
    def __init__(self,name,max_speed,mileage):  
        self.name = name  
        self.max_speed = max_speed  
        self.mileage = mileage  
  
    class Bus(Vehicle):  
        pass  
  
    a=Bus("School Volvo",180,12)  
  
    print(a.name)  
    print(a.max_speed)  
    print(a.mileage)  
  
School Volvo  
180  
12
```

4.Create a Bus object that will inherit all of the variables and methods of the parent Vehicle class and display it.

Expected Output:

Vehicle Name: School Volvo Speed: 180 Mileage: 12



Department of Computer Science and Engineering (Data Science)

```
class Vehicle:
    def __init__(self, name, max_speed, mileage):
        self.name = name
        self.max_speed = max_speed
        self.mileage = mileage

class Bus(Vehicle):
    pass

a=Bus("School Volvo",180,12)

print(a.name)
print(a.max_speed)
print(a.mileage)
```

School Volvo
180
12

```
[7] class Vehicle:
    def __init__(self, max_speed, mileage):
        self.max_speed = max_speed
        self.mileage = mileage

    class Bus(Vehicle):
        pass
```

```
[8] class Vehicle:
    def __init__(self, name, max_speed, mileage):
        self.name = name
```

5. Create a Bus class that inherits from the Vehicle class. Give the capacity argument of Bus.seating_capacity() a default value of 50.

Use the following code for your parent Vehicle class.

```
class Vehicle:
    def __init__(self, name, max_speed, mileage):
        self.name = name
        self.max_speed = max_speed
        self.mileage = mileage
```

```
    def seating_capacity(self, capacity):
        return f"The seating capacity of a {self.name} is {capacity} passengers"
```

Expected Output:

The seating capacity of a bus is 50 passengers



Department of Computer Science and Engineering (Data Science)

```
[6] 180
    12

[7] class Vehicle:
    def __init__(self,max_speed,mileage):
        self.max_speed = max_speed
        self.mileage = mileage

    class Bus(Vehicle):
        pass

[8] class Vehicle:
    def __init__(self, name, max_speed, mileage):
        self.name = name
        self.max_speed = max_speed
        self.mileage = mileage

    def seating_capacity(self, capacity):
        return f"The seating capacity of a {self.name} is {capacity} passengers"

    class Bus(Vehicle):
        def seating_capacity(self, capacity=50):
            return super().seating_capacity(capacity=50)

    b = Bus("School Volvo",180,12)
    print(b.seating_capacity())

The seating capacity of a School Volvo is 50 passengers

[9] class Vehicle:
    def __init__(self, name, mileage, capacity):
```

6. Class Inheritance

Given:

Create a **Bus** child class that inherits from the **Vehicle** class. The default fare charge of any vehicle is **seating capacity * 100**. If **Vehicle** is **Bus** instance, we need to add an extra 10% on full fare as a maintenance charge. So total fare for bus instance will become the **final amount = total fare + 10% of the total fare**.

Note: The bus seating capacity is **50**. so the final fare amount should be **5500**. You need to override the **fare()** method of a **Vehicle** class in **Bus** class.

Use the following code for your parent **Vehicle** class. We need to access the parent class from inside a method of a child class.

```
class Vehicle:
    def __init__(self, name,
        mileage, capacity):
```



Department of Computer Science and Engineering (Data Science)

```
self.name = name
self.mileage = mileage
self.capacity = capacity

def fare(self):    return
self.capacity * 100

class Bus(Vehicle):
    pass

School_bus = Bus("School Volvo", 12, 50)
print("Total Bus fare is:", School_bus.fare())
```

Expected Output:

Total Bus fare is: 5500.0

```
[8] class Bus(Vehicle):
    def seating_capacity(self, capacity=50):
        return super().seating_capacity(capacity=50)
b = Bus("School Volvo", 12, 50)
print(b.seating_capacity())

The seating capacity of a School Volvo is 50 passengers

[9] class Vehicle:
    def __init__(self, name, mileage, capacity):
        self.name = name
        self.mileage = mileage
        self.capacity = capacity
    def fare(self):
        return self.capacity*100
class Bus(Vehicle):
    def fare(self):
        amount = super().fare()
        amount = amount+amount*10/100
        return amount

School_bus = Bus("School Volvo", 12, 50)
print("Total Bus fare is:", School_bus.fare())

Total Bus fare is: 5500.0

[10] print(type(School_bus))

<class '__main__.Bus'>
```

7. Check type of an object



Department of Computer Science and Engineering (Data Science)

Write a program to determine which class a given Bus object belongs to.

```
PLP_Exp02_60009210033_Jhanvi Parekh.ipynb
File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text
[10] print(type(School_bus))
<class '__main__.Bus'>
[11] print(isinstance(School_bus,Vehicle))
True
[12] print(issubclass(Vehicle,Bus))
False

class Vehicle:
    def __init__(self,name,mileage,capacity):
        self.name = name
        self.mileage = mileage
        self.capacity = capacity
    def fare(self):
        return self.capacity*100
class Bus(Vehicle):
    pass
class Mini_bus(Bus):
    def price(self):
        amount = super().fare()
        amount = amount+amount*10/100
        return amount

School_bus = Mini_bus("School Volvo",12,50)
```

The screenshot shows a Google Colab notebook interface. The top bar includes the Google Colab logo and the file name 'PLP_Exp02_60009210033_Jhanvi Parekh.ipynb'. The notebook has a menu bar with 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help'. The left sidebar shows a file explorer with '+ Code' and '+ Text' tabs. The main area displays the code and its output. The code defines a 'Vehicle' class with an '__init__' method and a 'fare' method. It then defines a 'Bus' class that inherits from 'Vehicle' and a 'Mini_bus' class that inherits from 'Bus'. The 'Mini_bus' class has a 'price' method. The code creates an instance 'School_bus' of the 'Mini_bus' class. The output shows the type of 'School_bus' is 'Bus', it is an instance of 'Vehicle', and 'issubclass(Vehicle, Bus)' is 'False'. The bottom status bar shows the temperature as 31°C, the location as Smoke, and the date and time as 19:11 on 04-09-2023.



Department of Computer Science and Engineering (Data Science)

8. Determine if School_bus is also an instance of the Vehicle class

```
PLP_Exp02_60009210033_Jhanvi Parekh.ipynb
File Edit View Insert Runtime Tools Help All changes saved

[10] print(type(School_bus))
<class '__main__.Bus'>

[11] print(isinstance(School_bus,Vehicle))
True

[12] print(issubclass(Vehicle,Bus))
False

class Vehicle:
    def __init__(self,name,mileage,capacity):
        self.name = name
        self.mileage = mileage
        self.capacity = capacity
    def fare(self):
        return self.capacity*100
class Bus(Vehicle):
    pass
class Mini_bus(Bus):
    def price(self):
        amount = super().fare()
        amount = amount+amount*10/100
        return amount

School_bus = Mini_bus("School Volvo",12,50)
```

9. Determine if School_bus is Sub class of Vehicle Class

```
PLP_Exp02_60009210033_Jhanvi Parekh.ipynb
File Edit View Insert Runtime Tools Help All changes saved

[10] print(type(School_bus))
<class '__main__.Bus'>

[11] print(isinstance(School_bus,Vehicle))
True

[12] print(issubclass(Vehicle,Bus))
False

class Vehicle:
    def __init__(self,name,mileage,capacity):
        self.name = name
        self.mileage = mileage
        self.capacity = capacity
    def fare(self):
        return self.capacity*100
class Bus(Vehicle):
    pass
class Mini_bus(Bus):
    def price(self):
        amount = super().fare()
        amount = amount+amount*10/100
        return amount

School_bus = Mini_bus("School Volvo",12,50)
```

10. Create a child class for Bus Class named Mini bus inheriting Bus class and priceattribute and Print Price Method (Multilevel Inheritance)



Department of Computer Science and Engineering (Data Science)

Use the following code for your parent Vehicle class. We need to access the parent class from inside a method of a child class.

```
class Vehicle:
    def __init__(self, name, mileage, capacity):
        self.name = name
        self.mileage = mileage
        self.capacity = capacity

    def fare(self):
        return self.capacity * 100

class Bus(Vehicle):
    pass

School_bus = Bus("School Volvo", 12, 50)
print("Total Bus fare is:", School_bus.fare())
```

```
class Vehicle:
    def __init__(self, name, mileage, capacity):
        self.name = name
        self.mileage = mileage
        self.capacity = capacity
    def fare(self):
        return self.capacity*100
class Bus(Vehicle):
    pass
class Mini_bus(Bus):
    def price(self):
        amount = super().fare()
        amount = amount+amount*10/100
        return amount

School_bus = Mini_bus("School Volvo",12,50)
print("Total Bus fare is:",School_bus.fare())

Total Bus fare is: 5000
```

11. Create a car class inheriting Vehicle class and add type and Print attribute and displayType method method for the same

Use the following code for your parent Vehicle class. We need to access the parent class from inside a method of a child class.



Department of Computer Science and Engineering (Data Science)

```
class Vehicle:
    def __init__(self, name, mileage, capacity):
        self.name = name
        self.mileage = mileage
        self.capacity = capacity

    def fare(self):
        return self.capacity * 100

class Bus(Vehicle):
    pass

School_bus = Bus("School Volvo", 12, 50)
print("Total Bus fare is:", School_bus.fare())
```

```
class Vehicle:
    def __init__(self, name, mileage, capacity):
        self.name = name
        self.mileage = mileage
        self.capacity = capacity

    def fare(self):
        return self.capacity * 100

class Car(Vehicle):
    def __init__(self, name, mileage, capacity, car_type):
        super().__init__(name, mileage, capacity)
        self.car_type = car_type

    def display_type(self):
        print("Type of car:", self.car_type)

    def display_info(self):
        print("Name of car:", self.name)
        print("Mileage of car:", self.mileage)
        print("Capacity of car:", self.capacity)
        self.display_type()

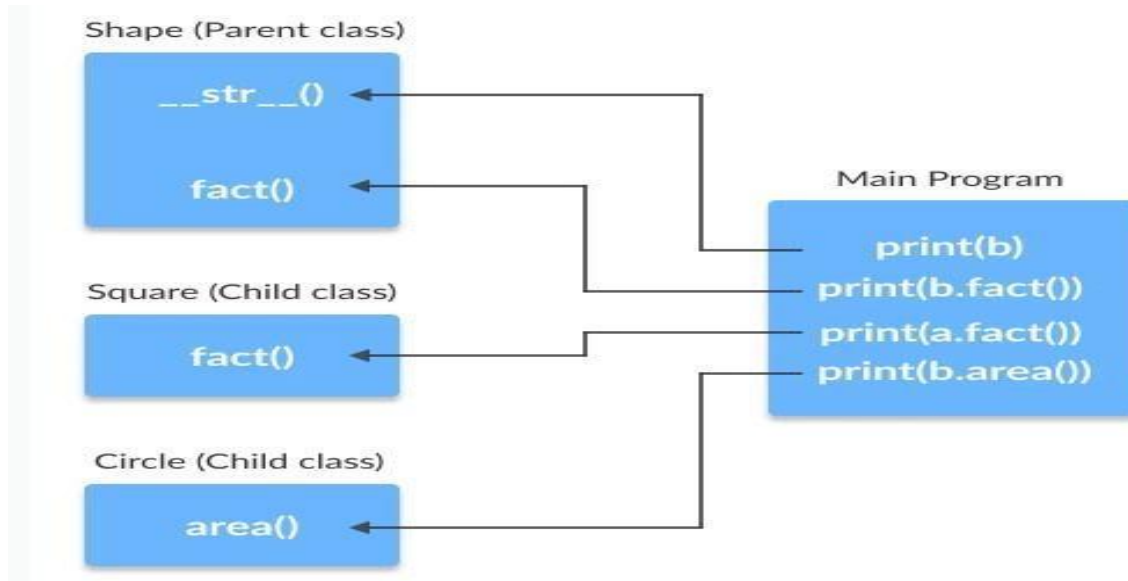
my_car = Car("Toyota Corolla", 20, 5, "Sedan")
my_car.display_info()
```

Name of car: Toyota Corolla
Mileage of car: 20
Capacity of car: 5
Type of car: Sedan



Department of Computer Science and Engineering (Data Science)

12. Write a program in python to Implement Method Overriding considering the following



```
PLP_Exp02_60009210033_Jhanvi Parekh.ipynb
File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text
Name of car: Toyota Corolla
Mileage of car: 20
Capacity of car: 5
Type of car: Sedan

[9] class Shape:
    def fact(self):
        msg = "A shape is the form of an object or its external boundary."
        return msg
    def __str__(self):
        return "This is a shape."
class Square(Shape):
    def fact(self):
        msg = "A square is a regular quadrilateral with four equal sides and four equal angles."
        return msg

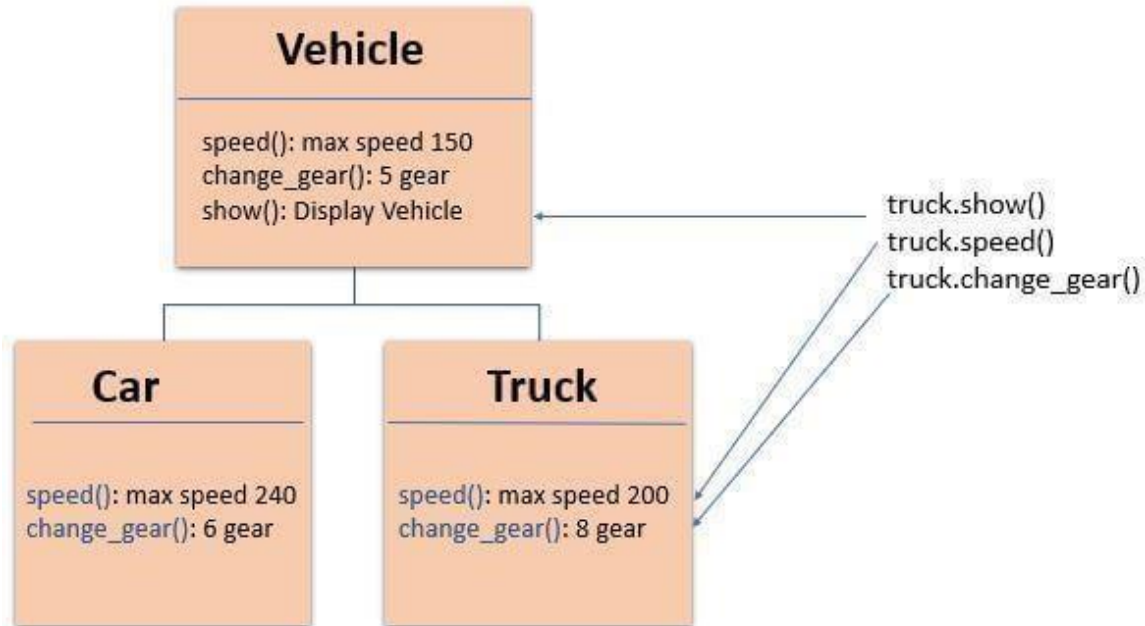
b = Shape()
a = Square()
print(b)
print(b.fact())
print(a.fact())

This is a shape.
A shape is the form of an object or its external boundary.
A square is a regular quadrilateral with four equal sides and four equal angles.
```

13. Write a program in python to Implement Method Overriding considering the following classes, methods and Variables



Department of Computer Science and Engineering (Data Science)



Method overridden in Car and Truck class

```
class Vehicle:
    def __init__(self, max_speed=150, gears=5):
        self.max_speed = max_speed
        self.gears = gears
    def speed(self):
        print("Current speed is {} km/h".format(self.max_speed))
    def change_gear(self, gear=None):
        print("Current gear is {}".format(self.gears))
    def show(self):
        print("This is a vehicle")

class Car(Vehicle):
    def __init__(self, max_speed=240, gears=6):
        super().__init__(max_speed, gears)
    def speed(self):
        print("Current speed is {} km/h".format(self.max_speed))
    def change_gear(self, gear=None):
        print("Current gear is {}".format(self.gears))
    def show(self):
        print("This is a car")

class Truck(Vehicle):
    def __init__(self, max_speed=200, gears=8):
        super().__init__(max_speed, gears)
    def speed(self):
        print("Current speed is {} km/h".format(self.max_speed))
    def change_gear(self, gear=None):
        print("Current gear is {}".format(self.gears))
    def show(self):
        print("This is a truck")

car = Car()
truck = Truck()
car.show()
car.speed()
car.change_gear()
print("")
truck.show()
truck.speed()
truck.change_gear()
```



Department of Computer Science and Engineering (Data Science)

```
self.gears = gears
def speed(self):
    print("Current speed is {} km/h".format(self.max_speed))
def change_gear(self, gear=None):
    print("Current gear is {}".format(self.gears))
def show(self):
    print("This is a vehicle")
class Car(Vehicle):
    def __init__(self, max_speed=240, gears=6):
        super().__init__(max_speed, gears)
    def speed(self):
        print("Current speed is {} km/h".format(self.max_speed))
    def change_gear(self, gear=None):
        print("Current gear is {}".format(self.gears))
    def show(self):
        print("This is a car")
class Truck(Vehicle):
    def __init__(self, max_speed=200, gears=8):
        super().__init__(max_speed, gears)
    def speed(self):
        print("Current speed is {} km/h".format(self.max_speed))
    def change_gear(self, gear=None):
        print("Current gear is {}".format(self.gears))
    def show(self):
        print("This is a truck")
car = Car()
truck = Truck()
car.show()
car.speed()
car.change_gear()
print("")
truck.show()
truck.speed()
truck.change_gear()
```

This is a car
Current speed is 240 km/h
Current gear is 6

This is a truck
Current speed is 200 km/h
Current gear is 8

14. Write a program in python to demonstrate how method overloading is achieved. Create a function to perform addition operation. If 2 no's are given it should perform addition of 2 no's otherwise it should add 3 nos.

```
def add(num1, num2, num3=None):
    if num3 is None:
        return num1 + num2
    else:
        return num1 + num2 + num3
n1 = int(input("Enter first number: "))
n2 = int(input("Enter second number: "))
ch = input("Do you want to add a third number? y/n: ")
if ch == 'y':
    n3 = int(input("Enter third number: "))
    print("Addition: ", add(n1, n2, n3))
else:
    print("Addition: ", add(n1, n2))
```

Enter first number: 5
Enter second number: 6
Do you want to add a third number? y/n: n
Addition: 11

```
[16] def calculate_area(side=None, length=None, breadth=None):
    if side is not None and length is None and breadth is None:
        area = side * side
        return area
    elif side is None and length is not None and breadth is not None:
        area = length * breadth
        return area
    else:
        return None
side1 = int(input("Enter side1: "))
ch = input("Do you want to enter another side? y/n: ")
if ch == 'y':
    side2 = int(input("Enter side 2: "))
```

```
from multipledispatch import dispatch
@dispatch(int,int,int)
def func(a,b,c):
    return a + b + c
@dispatch(int,int)
def func(a,b):
    return a + b
print(func(25,20,25))
print(func(25,20))
```



Department of Computer Science and Engineering (Data Science)

15. Write a program in python to perform method overloading. Create a function to calculate area of given shape. If 1 side is given calculate area of square, if Length and breadth is given calculate the area of Rectangle

```
def calculate_area(side=None, length=None, breadth=None):  
    if side is not None and length is None and breadth is None:  
        area = side * side  
        return area  
    elif side is None and length is not None and breadth is not None:  
        area = length * breadth  
        return area  
    else:  
        return None  
side1 = int(input("Enter side1: "))  
ch = input("Do you want to enter another side? y/n: ")  
if ch == "y":  
    side2 = int(input("Enter side 2: "))  
    print("Area of rectangle - ", calculate_area(side1, side2))  
else:  
    print("Area of square - ", calculate_area(side1))  
  
Enter side1: 6  
Do you want to enter another side? y/n: n  
Area of square - 36
```

16. Write a program to Implement Abstract class using Following problem statement The Employee class represents an employee, either full-time or hourly. the Employee class should be an abstract class because there're only full-time employees and hourly employees, no general employees exist.

The Employee class should have a property that returns the full name of an employee. In addition, it should have a method that calculates salary. The method for calculating salary should be an abstract method.

The Full_time_Employee class inherits from the Employee class. It'll provide the implementation for the get_salary () method.

Since full-time employees get fixed salaries, you can initialize the salary in the constructor of the class.

The HourlyEmployee also inherits from the Employee class.

However, hourly employees get paid by working hours and their rates. Therefore, you can initialize this information in the constructor of the class.

To calculate the salary for the hourly employees, you multiply the working hours and rates.



Department of Computer Science and Engineering (Data Science)

Activities Microsoft Edge Mar 6 08:47

(2) Assignments | Micros... x PLP_Exp02_60009210033 x +

https://colab.research.google.com/drive/1Rs_SPbpl-dbwQQBgku0d1Cu54NE5bljG#scrollTo=qvJJBU_txh9r

PLP_Exp02_60009210033_Jhanvi Parekh.ipynb ☆

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
import abc
class Employee(abc,ABC):
    def __init__(self, first_name, last_name):
        self.first_name = first_name
        self.last_name = last_name
    @property
    def full_name(self):
        return f'{self.first_name} {self.last_name}'
    @abc.abstractmethod
    def get_salary(self):
        pass
class Full_time_Employee(Employee):
    def __init__(self, first_name, last_name, salary):
        super().__init__(first_name, last_name)
        self.salary = salary
    def get_salary(self):
        return self.salary
class Hourly_Employee(Employee):
    def __init__(self, first_name, last_name, working_hours, hourly_rate):
        super().__init__(first_name, last_name)
        self.working_hours = working_hours
        self.hourly_rate = hourly_rate
    def get_salary(self):
        return self.working_hours * self.hourly_rate
full_time_employee = Full_time_Employee("John", "Doe", 50000)
hourly_employee = Hourly_Employee("Jane", "Doe", 40, 15)
print(full_time_employee.full_name)
print(full_time_employee.get_salary())
print(hourly_employee.full_name)
print(hourly_employee.get_salary())
```

John Doe
50000
Jane Doe
600



Department of Computer Science and Engineering (Data Science)

17. Write a Python Program to overload + operator to add Point (X1, Y1) and Point (X2, Y2)

```
class Point:
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y
    def __add__(self, other):
        x = self.x + other.x
        y = self.y + other.y
        return Point(x, y)
    def __str__(self):
        return "( {}, {} )".format(self.x, self.y)

# Create two points
p1 = Point(1, 2)
p2 = Point(3, 4)

# Add the points using the + operator
p3 = p1 + p2
# Print the result
print(p1, "+", p2, "=", p3)
```

(1, 2) + (3, 4) = (4, 6)

18. Write a Python Program to perform division of Two Numbers, take both the nos from users from user, use TRY, EXCEPT and FINALLY block to raise an Exception when Diving Number by Zero.

```
while True:
    try:
        num1 = float(input("Enter the first number: "))
        num2 = float(input("Enter the second number: "))
        result = num1 / num2
        print("The result of division is: (result)")
    except ZeroDivisionError:
        print("Error: Division by zero is not allowed.")
    finally:
        print("Exiting")
        break
```

Enter the first number: 25.2
Enter the second number: 5
The result of division is: 5.04
Exiting

https://colab.research.google.com/drive/1Rs_SPbpl-dbwQQBgku0d1Cu54NE5bIjG?usp=sharing