```
# Inheritance
# one class acquire properties of another class

class B:
    def day(self):
        print("Today is friday. Yay!!!")

class A(B):   # day method is inherited to A class
    pass


a=A()
a.day()

Today is friday. Yay!!!

class B:
    def day(self):
        print("Today is friday. Yay!!!")

class A(B):            # day method from B is inherited to A class
    def month(self):   # one property of itself and u have one more
property of class B
        print("July")

a=A()
a.day()
a.month()

Today is friday. Yay!!!
July

# practical benefit of inheritance
class D:
    def date(self):
        print("it is 2nd july")
class A(D):
    def day(self):
        print("friday")
class B(D):
    def month(self):
        print("july")
class C(D):
    def year(self):
        print("2021")

a=A()
a.day()
a.date()

b=B()
```

```
b.date()
b.month()

c=C()
c.date()
c.year()
```

```
friday
it is 2nd july
it is 2nd july
july
it is 2nd july
2021
```

```python
# Multiple inheritance
class B():
    def season(self):
        print("it is summer season")
class C():
    def pizza(self):
        print("pizza is delicious")
class A(B,C):
    def week(self):
        print("it is the first week of july")
a=A()
a.season()
a.pizza()
a.week()
```

```
it is summer season
pizza is delicious
it is the first week of july
```

```python
# multilevel inheritance
class D:
    def date(self):
        print("it is 2nd july")
class C(D):
    def day(self):
        print("friday")
class B(C):
    def month(self):
        print("july")
class A(B):
    def year(self):
        print("2021")
a=A()
a.year()
a.month()
```

```python
a.day()
a.date()
```

```
2021
july
friday
it is 2nd july
```

```python
# Task
class A:
    def monday(self):
        print("Today is monday")
class B:
    def tuesday(self):
        print("Today is tuesday")
class C:
    def wednesday(self):
        print("Today is wednesday")
```

```python
class C:
    def wednesday(self):
        print("Today is wednesday")
class B(C):
    def tuesday(self):
        print("Today is tuesday")
class A(B):
    def monday(self):
        print("Today is monday")

a=A()
a.monday()
a.tuesday()
a.wednesday()
```

```
Today is monday
Today is tuesday
Today is wednesday
```

```python
#a.py
from b import B
class A(B):
    def monday(self):
        print("Today is Monday")
a=A()
a.monday()
a.tuesday()
a.wednesday()
```

```python
#b.py
```

```python
#c.py
from b import B
```

```
a=A()
a.monday()
a.tuesday()
a.wednesday()



# Access Modifiers

# public access modifiers
class Car:
    # constructor is a special type of method because it always has
the following syntax
    def __init__(self, color,engine,tyre_color):  # properties i want
to attach with car
        self.color_of_car=color
        self.engine_type=engine
        self.color_of_tyre=tyre_color

 #  """ def print_car_info(self):
  #      print(f"color of car={self.color_of_car}")
   #     print(f"type of engine={self.engine_type}")
  #    #@static method
# def number_of_steering_wheels(self):
#       print("The car has 1 steering wheel")"""

mercedes=Car('silver','petrol','black')
print(mercedes.color_of_car)
print(mercedes.engine_type)
print(mercedes.color_of_tyre)

silver
petrol
black

class Test:
    mercedes=Car('silver','petrol','black')
    print(mercedes.color_of_car)
    print(mercedes.engine_type)
    print(mercedes.color_of_tyre)

silver
petrol
black

# private Modifiers
class Car:
    # constructor is a special type of method because it always has
the following syntax
    def __init__(self, color,engine,tyre_color):  # properties i want
to attach with car
        self.color_of_car=color  #when i put double underscore then it
```

```
is private
        self.__engine_type=engine
        self.__color_of_tyre=tyre_color
class Test:
    mercedes=Car('silver','petrol','black')
    print(mercedes.color_of_car)
    print(mercedes.engine_type)
    print(mercedes.color_of_tyre)

silver

----------------------------------------------------------------------
-----
AttributeError                            Traceback (most recent call
last)
<ipython-input-26-f735119e39a9> in <module>
      6         self.__engine_type=engine
      7         self.__color_of_tyre=tyre_color
----> 8 class Test:
      9     mercedes=Car('silver','petrol','black')
     10     print(mercedes.color_of_car)

<ipython-input-26-f735119e39a9> in Test()
      9     mercedes=Car('silver','petrol','black')
     10     print(mercedes.color_of_car)
---> 11     print(mercedes.engine_type)
     12     print(mercedes.color_of_tyre)

AttributeError: 'Car' object has no attribute 'engine_type'

class Car:
    # constructor is a special type of method because it always has
the following syntax
    def __init__(self, color,engine,tyre_color):  # properties i want
to attach with car
        self.__color_of_car=color  #when i put double underscore then
it is private
        self.__engine_type=engine
        self.__color_of_tyre=tyre_color
mercedes=Car('silver','petrol','black')
print(mercedes.__color_of_car)

----------------------------------------------------------------------
-----
AttributeError                            Traceback (most recent call
last)
<ipython-input-27-3d5a60e88202> in <module>
      6         self.__color_of_tyre=tyre_color
      7 mercedes=Car('silver','petrol','black')
----> 8 print(mercedes.__color_of_car)
```

```
AttributeError: 'Car' object has no attribute '__color_of_car'
```

```python
class Car:
    # constructor is a special type of method because it always has
the following syntax
    def __init__(self, color,engine,tyre_color):  # properties i want
to attach with car
        self.__color_of_car=color  #when i put double underscore then
it is private
        self.__engine_type=engine
        self.__color_of_tyre=tyre_color

    def print_car_info(self):
        print(self.__color_of_car)
mercedes=Car('silver','petrol','black')
mercedes.print_car_info()
```

```
silver
```

```python
# protected
```

```python
# Method Overloading
# method pverloading is not allowed in python
# Task 1
class Calculator:
    def add(self,num1,num2):
        print(f"addition of num1 and num2 = {num1+num2}")

    def add(self,num1,num2,num3):
        print(f"addition of num1 and num2 = {num1+num2+num3}")

    def subtract(self,num1,num2):
        print(f"subtraction of num1 and num2 = {num1-num2}")

    def multiply(self,num1,num2):
        print(f"multiplication of num1 and num2 = {num1*num2}")

    def divide(self,num1,num2):
        print(f"division of num1 and num2 = {num1/num2}")


calculator=Calculator()
```

```python
calculator.add(2,4)
```

```
----------------------------------------------------------------------
-----
TypeError                                 Traceback (most recent call
last)
```

```
<ipython-input-33-2deeb4859bcd> in <module>
----> 1 calculator.add(2,4)

TypeError: add() missing 1 required positional argument: 'num3'
```

```python
calculator.add(2,4,8)
```

```
addition of num1 and num2 = 14
```

```python
# method overloading is not allowed in python
```

```python
# Method Overriding
class B():
    def month(self):
        print("july")
class A(B):
    def year(self):
        print("2021")

    def month(self):
        print("August")  #method of child class is preferred over 
parent class
a=A()
a.year()
a.month()
```

```
2021
August
```

```python
#Task
```

```python
class Aeroplane():
    def fly():
        print("Aeroplane flies at 800mph!")
    def travel():
        print("Aeroplane help us travel faster")
class Helicopter():
    def hover():
        print("Helicopter can hover over ground")
class FlyingMachine():
    def travel():
        print("Machines which fly are used to travel from one point to 
another")
```

```python
class FlyingMachine():
    def travel(self):
        print("Machines which fly are used to travel from one point to 
another")
class Aeroplane(FlyingMachine):
    def fly(self):
        print("Aeroplane flies at 800mph!")
    def travel(self):
        print("Aeroplane help us travel faster")
class Helicopter(FlyingMachine):
    def hover(self):
        print("Helicopter can hover over ground")

h=Helicopter()
h.hover()
h.travel()
```

```
Helicopter can hover over ground
Machines which fly are used to travel from one point to another
```