```python
#dictionary in python
dict = {
    "name" : "Piyush"
}
print(dict)
dict["name"]="Pushkar"
print(dict)

roll ={
}
i=0
for i in range(0,5):
    roll[i]=i
    i+=1

print(roll)
```

```
{'name': 'Piyush'}
{'name': 'Pushkar'}
{0: 0, 1: 1, 2: 2, 3: 3, 4: 4}
```

```python
#nested dictionary
dict = {
    "name": "Piyush",
    "marks" : {
    "Chemistry":98,
    "Physics":99,
    "maths":100
    }
}

print(dict)
print(dict.keys())
print(list(dict))
print(tuple(dict))
print(len(dict)) #no of keys
print(dict.values())
dict.update({"Address":"Lucknow"})
print(dict)
```

```
{'name': 'Piyush', 'marks': {'Chemistry': 98, 'Physics': 99, 'maths':
100}}
dict_keys(['name', 'marks'])
['name', 'marks']
('name', 'marks')
2
dict_values(['Piyush', {'Chemistry': 98, 'Physics': 99, 'maths':
100}])
{'name': 'Piyush', 'marks': {'Chemistry': 98, 'Physics': 99, 'maths':
100}, 'Address': 'Lucknow'}
```

```
set_1={1,1,2,3,4,5,6,6}
print(set_1)
for i in set_1:
    print(i)
```

```
{1, 2, 3, 4, 5, 6}
1
2
3
4
5
6
```

```
#for creating empty dictionARY use collection=set()---->this syntax
collection= set()
print(type(collection))
collection.add(5)
collection.add(6)
collection.add(7)
collection.remove(5)
print(collection)
set_2={1,2,3}
print(collection.union(set_1))
```

```
<class 'set'>
{6, 7}
{1, 2, 3, 4, 5, 6, 7}
```

```
#functions in python
def calc_sum(a,b):
    return a+b
print(calc_sum(5,6))
 # use end=" " in print statement to print in same line
```

```
11
```

```
#calculating factorial
def factorial(n):
    if n==1:
        return 1
    return n*factorial(n-1)
    n-=1
print(factorial(6))
```

```
720
```

```
#OOPS in Python
#classes
class Student:
    def __init__(self,name,rollno):
        self.name=name
```

```
        self.rollno=rollno


s1=Student("piyush",29)
print(s1.name,s1.rollno)
```

```
piyush 29
```

```
#inheritance
class base_class:
    def __init__(self,a,b):
        print(a+b)
    def __init__(self,a,b):
        print(a-b)
s1=base_class(5,6)
```

```
-1
```

```
#single inheritance
class Employee:
    def __init__(self,name,salary):
        self.name=name
        self.salary=salary

    def display_name(self):
        print(f"The name of the employee is:{self.name}"+f"salary
is{self.salary}")
class Manager(Employee):
    def __init__(self,name,salary,department):
        super().__init__(name,salary)
        self.department=department

    def  manager_show(self):
        print(f"The name of the employee is:{self.name}"+f"salary
is{self.salary}"+f"position is {self.department}")

manager1=Manager("alice",80000,"HR")
manager1.display_name()
manager1.manager_show()
```

```
The name of the employee is:alicesalary is80000
The name of the employee is:alicesalary is80000position is HR
```

```
#multiple inheritance
class Employee:
    def __init__(self,name,salary):
        self.name=name
        self.salary=salary
```

```
    def display_name(self):
        print(f"The name of the employee is:{self.name}"+f"salary
is{self.salary}")
class Manager:
    def __init__(self,department):
        self.department=department

    def  manager_show(self):
        print(f"position is {self.department}")

class Final_d(Employee,Manager):
    def __init__(self,name,salary,department,address):
        self.address=address
        Employee.__init__(self,name,salary)#don't forget to pass the
data with self here
        Manager.__init__(self,department)#don't forget to pass the
data with self here
        self.address=address

    def show_details(self):
        print(f"The name of the employee is{self.name}"+f"and salary
is{self.salary}"+f"and department is{self.department}"f"and address
is{self.address}")

manager1=Final_d("alice",80000,"HR","Lucknow")
manager1.show_details()
```

```
The name of the employee isaliceand salary is80000and department
isHRand address isLucknow
```

```
#multilevel inheritance

#multiple inheritance
class Employee:
    def __init__(self,name,salary):
        self.name=name
        self.salary=salary

    def display_name(self):
        print(f"The name of the employee is:{self.name}"+f"salary
is{self.salary}")
class Manager(Employee):
    def __init__(self,name,salary,department):
        super().__init__(name,salary)
        self.department=department

    def  manager_show(self):
        print(f"position is {self.department}")

class Final_d(Manager):
```

```
    def __init__(self,name,salary,department,address):
        super().__init__(name,salary,department)
        self.address=address

    def show_details(self):
        print(f"The name of the employee is{self.name}"+f"and salary
is{self.salary}"+f"and department is{self.department}"f"and address
is{self.address}")

manager1=Final_d("alice",80000,"HR","Lucknow")
manager1.show_details()

#super ke saath self nhi use krte
```

```
The name of the employee isaliceand salary is80000and department
isHRand address isLucknow
```

```
#Method Overriding
class Employee:
    def __init__(self,name,salary):
        self.name=name
        self.salary=salary

    def display_name(self):
        print(f"The name of the employee is:{self.name}"+f"salary
is{self.salary}")
class Manager(Employee):
    def __init__(self,name,salary,department):
        super().__init__(name,salary)
        self.department=department

    def  display_name(self):
        print(f"The name of the employee is:{self.name}"+f"salary
is{self.salary}"+f"position is {self.department}")

manager1=Manager("alice",80000,"HR")
manager1.display_name()
```

```
The name of the employee is:alicesalary is80000position is HR
```

```
#Method_Overloading
class Calculator:
    # Method with default values for overloading behavior
    def add(self, a, b=0, c=0):
        return a + b + c

# Create an object of Calculator class
calc = Calculator()

# Call the add method with different numbers of arguments
print(calc.add(5))          # Output: 5
```

```
print(calc.add(5, 10))      # Output: 15
print(calc.add(5, 10, 15))  # Output: 30
```

```
5
15
30
```

```
#Hybrid Inheritance
```

```
#Hybrid Inheritance
class Employee:
    def __init__(self,name,salary):
        self.name=name
        self.salary=salary
    def show_name(self):
        print(f"The name of the employee is{self.name}")
class Manager1(Employee):
    def print_1(self):
        print(f"{self.name}is desginer")

class Manager2(Employee):
    def print_2(self):
        print(f"{self.name}is Video Editor")

manager1=Manager1("alice",80000)
manager1.show_name()
manager1.print_1()

manager2=Manager2("venice",90000)
manager2.show_name()
manager2.print_2()
```

```
The name of the employee isalice
aliceis desginer
The name of the employee isvenice
veniceis Video Editor
```