→ "Inheritance"

It allows us to define a class that inherits all the methods and properties from another class.

Parent class is the class being inherited from, it is also known as base class.

Child class is the class that inherits from another class, also called derived class

Creating a Parent Class :⇒ The class whose attributes and methods are inherited is called as parent class. It is defined just like other classes i.e. using the class keyword.

Syntax :

```
Class ParentClassName:
    { Class body }
```

Creating a Child class :⇒ Classes that inherit from the base classes are declared similarly to their parent class, however, we need to provide the name of parent classes within the parentheses.

Syntax :

```
Class SubclassName ( ParentClass1, ParentClass2 ---- ):
    { sub class body }
```

→ Types of Inheritance :→

In python, inheritance can be divided into five different categories :—

1> Single Inheritance
2> Multiple Inheritance
3> Multilevel Inheritance
4> Hierarchical Inheritance
5> Hybrid Inheritance

1> "Single Inheritance" :→ This is the simplest form of inheritance, where a child class inherits attributes and methods from only one parent class.

Eg :→

```
class Parent:
    def parentmethod(self):
        print("Calling parent method")

class Child(Parent):
    def childmethod(self):
        print("Calling child method")

c = Child()
c.childmethod()
c.parentmethod()
```

O/p:

```
Calling child method
calling parent method.
```

P.T.O

2> "Multiple Inheritance" :⇒ Multiple Inheritance in python allows you to construct a class based on more than one parent class. The child class thus inherits the attributes & methods from all the parents. The child can override the methods inherited from any parent.

Syntax :⇒

```
Class parent1:
  #statements

Class parent2:
  #statements

Class child(parent1, parent2):
  # statements.
```

Example :⇒

```
Class division:
    def __init__(self, a, b):
        self.n = a
        self.d = b

    def divide(self):
        return self.n/self.d

Class modulus:
    def __init__(self,a,b):
        self.n = a
        self.d = b

    def mod_divide(self):
        return self.n%self.d
```

```
class div_mod(division, modulus):
    def __init__(self, a, b):
        self.n = a
        self.d = b

    def div_and_mod(self):
        divval = division.divide(self)
        modval = modulus.mod_divide(self)
        return (divval, modval)

x = div_mod(10, 3)
print("division:", x.divide())
print("mod_division:", x.mod_divide())
print("divmod:", x.div_and_mod())
```

O/p :-

```
division : 3.3333
mod_division : 1
divmod: (3.3333, 1).
```

→ ③ "Multilevel Inheritance" :-

In multilevel inheritance, a class is derived from another derived class. There exists multiple layers of inheritance. We can imagine it as a grandparent – parent – child relationship.

[P.T.O]

Example :->

```
class Universe:
    def Universemethod(self):
        print("I am in the Universe")

class Earth(Universe):
    def earthmethod(self):
        print("I am on Earth")

class India(Earth):
    def Indianmethod(self):
        print("I am in India")

person = India()
person.Universemethod()
person.earthmethod()
person.indianmethod()
```

O/P:-> I am in the Universe
I am on Earth
I am in India

-> (4) Hierarchical Inheritance :->

This type of inheritance contains multiple derived classes that are inherited from a single base class. This is similar to the hierarchy within an organization.

[P.T.O]

Example :->

```
Class Manager:
    def Managermethod(self):
        print("I am the Manager")

Class Employee1(Manager):
    def employee1methode(self):
        print("I am Employee one")

Class Employee2(Manager):
    def employee2method(self):
        print("I am employee too")

empl = Employee1()
emp2 = Employee2()

empl.managermethod()
empl.employee1method()
emp2.managermethod()
emp2.employee2method()
```

O/p:->

```
I am the manager
I am Employee one
I am the manager
I am Employee too.
```

(5) -> "Hybrid Inheritance" :->

Combination of two or more types of inheritance is called Hybrid Inheritance. For instance, it could be a mix of single &

[P.T.O]

multiple Inheritance.

Example :→ Here we have combined single & multiple inheritance to form a hybrid inheritance of classes.

```
Class CEO:
    def ceomethod(self):
        print("I am the CEO")

class Manager(CEO):
    def managermethod(self):
        print("I am the manager")

Class Employee1(Manager):
    def employee1method(self):
        print("I am employee one")

Class Employee2(Manager, CEO):
    def employee2method(self):
        print("I am Employee two")

emp = Employee2()
emp.managermethod()
emp.ceomethod()
emp.employee2method()
```

O/P: I am the manager
I am the CEO
I am Employee two.

[P.T.O]

→ Super() function :→

In python super() function allows you to access methods & attributes of the parent class from within a child class.

Example :→ In the following we create a parent class & access its constructor from a subclass using the super() function.

```
Class ParentDemo:
    def __init__(self, msg):
        self.message = msg
    def showMessage(self):
        print(self.message)

Class ChildDemo(ParentDemo):
    def __init__(self, msg):
        super().__init__(msg)

obj = ChildDemo("Welcome to Python Programming!!")
obj.showMessage()
```

O/p:→ Welcome to Python Programming!!

—— o ——

Q:⇒ We want to manage details of Teachers & Students. Both share details like; Name & Age, but both have their own specific details too, like student will be having a student id & teacher will be having subject he teaches.

Sol:⇒

```
Class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def display_details(self):
        print(f"Name : {self.name}")
        print(f"Age : {self.age}")

Class Student(Person):
    def __init__(self, name, age, student_id):
        super().__init__(name, age)
        self.student_id = student_id

    def display_student(self):
        self.display_details()
        print(f"Student. ID : {self.student_id}")

Class Teacher(Person):
    def __init__(self, name, age, subject):
        super().__init__(name, age)
        self.subject = subject

    def display_teacher(self):
        self.display_details()
        print(f"Subject : {self.subject}")
```

```
s1 = Student("Rahul", 16, "S123")
t1 = Teacher("Vikas", 38, "Mathematics")

print("Student Details:")
s1.display_student()
print("\nTeacher Details:")
t1.display_teacher()
```

O/p:→

Student Details:
Name: Rahul
Age: 16
Student ID: S123

Teacher Details:
Name: Vikas
Age: 38
Subject: Mathematics.

Q:→ Write a python program to create a class structure for a Banking system using inheritance.

Requirements :→

Create a parent class → Account with attributes { Account holder name & Account Number }

function: display_account_details() (to display name & account no.)

Create a child class → Savings account that inherits from account class & has balance (initially provided at object creation.

function: display_saving_details() → Display account details & balance.

Create another child class → CurrentAccount() that inherits from Account class & has: overdraft limit.

function: display_current_details() → Display account details & overdraft limit.

Sol:→

```
class Account:
    def __init__(self, name, acc_no):
        self.name = name
        self.accno = acc_no

    def display_account_details(self):
        print(f"Account Holder Name : {self.name}")
        print(f" Account Number : {self.acc_no}")

class SavingAccount(Account):
    def __init__(self, name, acc_no, balance):
        super().__init__(name, acc_no)
        self.balance = balance

    def display_saving_details(self):
        self.display_account_details()
        print(f"Balance : {self.balance}")
```

[P.T.O]

```
class CurrentAccount(Account):
    def __init__(self, name, acc_no, overdraft_limit):
        super().__init__(name, acc_no)
        self.overdraft_limit = overdraft_limit

    def display_current_details(self):
        self.display_account_details()
        print(f"Overdraft Limit: {self.overdraft_limit}")

print("Saving Account Details:")
s1 = SavingAccount("Ravi", 12345, 25000)
s1.display_saving_details()

print("\nCurrent Account Details:")
c1 = CurrentAccount("Priya", 67890, 50000)
c1.display_current_details()
```

O/P:→

```
Saving Account Details:
Account Holder Name: Ravi
Account Number: 12345
Balance: 25000

Current Account Details:
Account Holder Name: Priya
Account Number: 67890
Overdraft Limit: 50000.
```

—o—