# OOPS:

- OOPS→ Object Oriented Programming Structure
- OOPS is a paradigm (Style of writing the code/Programs)
    - Dart Mouth Conference→ 1956-1960
        - 1956→ AI → John Maccati
        - 1957→ Perceptron
        - 1959→ ML
        - 1960→ Oops→ Alan Kay
- First language with OOPS -> SIMULA in 1996→ Followed by SmallTalk in 1972

**USE OF OOPS:**

- Reusability
    - Inheritance
    - Polymorphism
- Security
    - Encapsulation
    - Abstraction

**Four Major Concepts of Oops:**

- Inheritance
- Polymorphism
- Encapsulation
- Abstraction

**Concepts of Oops:**

- Class
- Object
- Reference Variable
- “Self” Keyword
- Types of Variables
    - Local
    - Instance
    - Static
- Types of Methods
    - Instance
    - Class
    - Static
- Constructor
- “Super” Keyword
- Nested Class

# Dt-06/03/2023

# OOPS:

- OOPS→ Object Oriented Programming Structure
- OOPS is a paradigm (Style of writing the code)
- First language with OOPS -> SIMULA in 1996→ Followed by SmallTalk in 1972

**Class:**

- It is a structure which consists of member variables and member methods.
    - Variables→ which refers to the memory location.
    - Methods→ It's a function within the class.
- It's not a real world entity.
    - It does not occupy memory.
- It can be considered as prototype/template/Blue-print.

**Syntax:**

Class <classname>:

    ----------

    ----------

    ----------

    ----------

**Object:**

- Instance of the class.
- Object is a real world entity.
    - Object occupy memory.
- For each class we can derive "n" number of objects.
- Object is mandatory to access members of the class, without object we cannot access members of the class.
- It is a built-in Class.

**Syntax:**

<Classname>()

**"SELF" KEYWORD:**

It refers to current object.

**Types of Variables:**

**Local Variables:**

- These variables are declared and accessed within the same method of a particular class.

**Instance Variables:**

- These variables are class level global variables,
- These variables are declared within a method which can be accessed in any method of same class.
- Instance is prefixed with “self” keyword.

  Self.name=”Arjun”(Instance Variable)

- Instance Variable can be accessed using “self” Keyword.

  Print(self.name)

**Static Variables:**

- These variables are considered as a Global Variable which can be accessed outside the class also.
- These variables are declared outside all the method and inside the class.

  **Accessibility:**

- These variables are accessible within the method of same class using class name.

  City=”Hydrabad”

  Print(<classname>.city)  # Inside the method
- These variables are accessible outside the class using class name or class object.

  <classname>.city  # Outside the method,, Using class

  Obj.city  # Using object

# Dt-07/03/2023

**Types of Methods**

**Instance:**

- This is default method.
- A method with “self” as default argument is considered as instance method.

**Accessibility:**

- These methods can be accessed within the class and from outside the class.

Within Class:

- It is accessed using “self” keyword.

Self.<methodname>()

Outside Class:

- It can be accessed using object.

Obj.<methodname>()

❖ All the variables are accessible.

**Class:**

- A method with “cls” as default argument is considered as class method.
- These class methods are defined using decorator/ annotation @Classmethod.

**Accessibility:**

Outside Class:

- These methods are accessed using two ways,
- <classname>.<methodname>()
- <obj>.<methodname>()

**Variables:**

- Local variables
- Static Variables

<classname>.<staticvariable>

Cls.staticvariable

- Instance variables cannot be accessed.

**Static:**

- A method with no default arguments is called as static method.
- These static methods are defined using decorator/annotation @staticmethod

**Accessibility:**

Outside class:

These methods are accessed using two ways:

<classname>.<methodname>()

<obj>.<methodname>()

**Variables:**

- Local variables
- Static Variables
  <classname>.<staticvariable>
  Cls.staticvariable
- Instance variables cannot be accessed.

**Constructor:**

- A method which can be executed without explicitly calling.
- Constructor is a special type of method which is executed whenever an object is created.
- These constructors are used to initialize the variables with values.
- Name of constructor is : __init__()
- Constructor comes under the Instance Method category, i.e. default argument is self.
- Any number of constructor can be defined within a class but only one is executed i.e. latest one.
- These constructor is of 2 types,
  **Default Constructor**
  - A constructor without any arguments.

  **Parameterized Constructor**
  - A constructor with arguments.

**Syntax:**

```
Def __init__(self):
        ----------------------
        ------------------------
        ----------------------
```

**Nested Class:**

# Dt-09/03/2023

## INHERITANCE:

- It is a process of inheriting the properties of one class to another class.
- Class whose properties are transferred is called as Parent Class/Base Class/Super Class.
- Class which receives properties are called as Child Class/Derived Class/Sub Class.

**Syntax:**

Class A: # Parent

---

---

---

Class B(A): # Child

**********

**********

**********

❖ By default all user defined classes are inherited from built-in class called “object”.

**Types of Inheritance:**

1. **Simple Inheritance/Single Inheritance**
   - It is one-level of inheritance where single parent properties are inherited to single child.

   Class A:

   ---

   ---

   Class B(A):

   _____

   _____

2. **Multi-Level Inheritance**
   - It is combination of multiple levels of inheritances.
   - In this Parent properties are inherited to child class and Child class properties are inherited to Grand Child class.
   - Grand Child can access member of Parent Class, Child Class, Grand Child Class.

Class A:

---

---

---

Class B(A):

_____

_____

_____

Class C(B):

********

********

********

3. **Hierarchical Inheritance**
    - In this type of inheritance, the parent class properties are inherited to multiple child class.
    - i.e. one parent-→ Multiple Child
    - This is one to many relationship.
    - There is no relationship between Child Classes.

Class A:

---

---

Class B(A):

---

---

Class C(A):

---

---

4. **Multiple Inheritance** → [Diamond Problem/ Ambiguity Problem]

5. Hybrid Inheritance

# Dt-10/03/2023

[Diamond Problem/ Ambiguity Problem]

**4. Multiple Inheritance:**

- Single child class is inherited with multiple parent class properties.

**Syntax:**

Class Child(A,B):

**5.Hybrid Inheritance:**

- It's a combination of two or more above inheritance,
    - Multiple+Hierarchical
    - Mult-Level+ Hierarchical
    - Multiple+Multi-Level

# POLYMORPHISM:

Poly→ Many

Morphism→ Forms

- An object which has many forms is called as **Polymorphism**.
- An object which behaves differently based on situation is called **Polymorphism**.

**Implementation of Polymorphism:**

Overloading(Same Class)

- Object behaves differently based on situation within the same class.
    - Variable Overloading
    - Operator Overloading
    - Method Overloading
    - Constructor Overloading
    - MO & CO is not supported by Python.

❖ VARIABLE-LENGTH ARGUMENT

Overriding(Multiple Class)

- Object behaves differently based on situation within multiple class.
    - Variable Overriding
    - Method Overriding
    - Constructor Overriding

# Dt-13/03/2023

**METHOD OVERLOADING & CONSTRUCTOR OVERLOADING:**

- Within the same class.
- Multiple Methods with Same Name
- Methods with Different Signature(Different Arguments)
- Methods with Different implementation(Logics)

**OVERRIDING:**

**METHOD OVERRIDING & CONSTRUCTOR OVERRIDING:**

- Across multiple inheritated class
- Multiple methods with same name(Across Class)
- Methods with Same Signatures
- Different Implementation

# ENCAPSULATION:

- It is data level security.
- It is a process of restricting unauthorized user to access data from class.

❖ Access Specifier→Implementation

- This data level security is implemented using access specifier
    - **Public**
        - Within the same class, outside the class, inherited class also.
    - **Protected**
        - An object Which is accessible inside the inherited class, within the class.
    - **Private**
        - It is accessible within the same class only.

**According to Python**

- Public & Protected access anywhere:
    - Within the same class
    - Inherited class
    - Outside the class(from main program)

- Private objects can be accessed within the same class only. It is strictly restricted from accessing from inherited class and outside the class.

Public Variable➔ city=”Hydrabad”

Protected Variable➔ _city=”Delhi”

Private Variable➔ __city=”New York”

**Access Specifier in Python can be implemented**

- Variable Level
- Method Level

# Dt-14/03/2023

## ABSTRACTION:

- It provides security.
- It is implementing the security by hiding internal mechanism.
    - To avoid unauthorized user accessing internal mechanism.
    - Hiding internal mechanism not to show unnecessary information to the user.

**Implementation Of Abstraction:**

It is implemented in two steps process:

- Creating Abstract Class
    - Abstract class is created by inheriting properties of build-in Abstract class by the name "ABC" to your user defined class.

```
From abc import ABC

Class sky(ABC):
```

- Creating Abstract Method
    - Abstract method is created with the help of decorator "@abstractmethod".

```
@abstractmethod
Def xyz(self):
```

    - Abstract method should not have implementation(Logic) within abstract class.
    - We can have "n" number of abstract method within the abstract class.

❖ Non abstract method within abstract class is called **Concrete Method.**
❖ These concrete holds actual logic within abstract class.

**Dt-15/03/2023**