**Session1 Class and OOPS concept**

**What is OOPs? and What are the various elements of OOPs?**\ OOPs (Object-Oriented Programming) is a paradigm based on objects and classes.\ **Elements of OOPs**:

* Class
* Object
* Inheritance
* Polymorphism
* Encapsulation
* Abstraction

**Enumerate some differences between procedural programming and OOP.**

| **Procedural Programming** | **Object-Oriented Programming** |
| --- | --- |
| Based on functions | Based on objects and classes |
| Less reusable code | Promotes code reuse |
| Harder to maintain | Easier to maintain and scale |

**What is the purpose of the \_\_slots\_\_ in Python classes, and how does it improve performance?**\ \_\_slots\_\_ restricts dynamic creation of attributes and saves memory.

class MyClass:

**slots** = ['x', 'y']

**What will happen if a subclass does not implement an abstract method defined in its base class?**\ It will raise a TypeError when trying to instantiate the subclass.

**Create a Python class hierarchy for a Vehicle, with subclasses Car and Motorcycle, and implement a speed method for both subclasses.**

class Vehicle:

    def speed(self):

        pass

class Car(Vehicle):

    def speed(self):

        return "Car speed is 120 km/h"

class Motorcycle(Vehicle):

    def speed(self):

        return "Motorcycle speed is 80 km/h"

**Write a Python class to represent a Bank Account with methods to deposit, withdraw, and check balance.**

class BankAccount:

    def **init**(self):

        self.balance = 0

    def deposit(self, amount):

        self.balance += amount

    def withdraw(self, amount):

        if amount <= self.balance:

            self.balance -= amount

    def check\_balance(self):

        return self.balance

**Session2 Regular Expressions**

**Explain the concept of method overriding and method overloading in Python**

* **Overriding**: Redefining a method in a subclass.
* **Overloading**: Not directly supported; can be mimicked using default arguments.

**Design a Python class to represent a Rectangle and a Square, and ensure Square inherits from Rectangle. Include methods to calculate the area and perimeter.**

class Rectangle:

    def **init**(self, length, width):

        self.length = length

        self.width = width

    def area(self):

        return self.length *self.width*

*def perimeter(self):*

*return 2* (self.length + self.width)

class Square(Rectangle):

    def **init**(self, side):

        super().**init**(side, side)

**Design a Class System: “Design a simple class system for a library that includes classes for books, members, and loans. How would you ensure that books can be checked out, returned, and overdue books tracked?”**

class Book:

    def **init**(self, title):

        self.title = title

        self.available = True

class Member:

    def **init**(self, name):

        self.name = name

        self.loans = []

class Loan:

    def **init**(self, book, member, due*date):*

*self.book = book*

*self.member = member*

*self.due*date = due\_date

        book.available = False

**How does Python support multiple inheritance, and what are the potential issues?**\ Python supports multiple inheritance using comma-separated base classes.\ **Issue**: Diamond problem, resolved using MRO (Method Resolution Order).

**How do you implement a property in Python, and how does it differ from a regular attribute?**\ Use @property decorator to define getter/setter methods.

class Person:

    def **init**(self, age):

        self.*age = age*

*@property*

*def age(self):*

*return self.*age

**GUI in Python**

**What is python tkinter? What is tkinter used for in Python?**\ Tkinter is a standard GUI library in Python used to create desktop applications.

**Write a Python GUI program to import Tkinter package and create a window and set its title.**

import tkinter as tk

window = tk.Tk()

window.title("My Window")

window.mainloop()

**Create a window and set the default window size using tkinter module**

window.geometry("400x300")

**Python considered a bad choice for GUI desktop app development? Do you agree, if not justify**\ Not necessarily. While Python is slower than some compiled languages, libraries like Tkinter, PyQt, and Kivy make it suitable for many GUI apps.

**Write a Python GUI program to create three labels and 2 textboxes using Tkinter Library. Once the user gives a value of N**

import tkinter as tk

window = tk.Tk()

tk.Label(window, text="Label 1").pack()

tk.Label(window, text="Label 2").pack()

tk.Label(window, text="Label 3").pack()

entry1 = tk.Entry(window)

entry2 = tk.Entry(window)

entry1.pack()

entry2.pack()

window.mainloop()

**Write a Python GUI program that adds labels and buttons to the Tkinter window.**

tk.Label(window, text="Click below").pack()

tk.Button(window, text="Button 1").pack()

tk.Button(window, text="Button 2").pack()

**Write a Python program that implements event handling for button clicks using Tkinter.**

def on*click():*

*print("Button clicked!")*

*btn = tk.Button(window, text="Click Me", command=on*click)