BCSC 0702: COMPUTER PROGRAMMING- II

Course Objectives: *The course is designed to provide advanced concepts of Object-Oriented Programming (OOP), Graphical User Interface (GUI) development, and Network Programming. By the end of this course, students will be equipped to develop advanced Python applications, utilizing OOP principles, GUI frameworks, and networking capabilities*

Credits: 03 **L–T–P: 3–0–0**

Module No.	Content	Hours
I	**Introduction to Object-Oriented Programming (OOP):** What is Object-Oriented Programming? Benefits of OOP: Modularity, Reusability, and Extensibility. Difference between Procedural and Object-Oriented Programming. Real-world analogies of OOP concepts **Core OOP Principles: Encapsulation:** Definition and Importance, Implementing Encapsulation using Classes and Objects. Access Modifiers: Public, Protected, and Private. **Abstraction:** Definition and Role in Simplifying Complexity Achieving Abstraction using Abstract Classes Examples and Use Cases. **Inheritance:** Understanding Code Reusability through Inheritance. Superclass and Subclass Relationship. Types of Inheritance: Single, Multiple, Multilevel. **Polymorphism:** Multiple Implementations, Achieving Polymorphism through: Method Overloading, Method Overriding, Duck Typing **Class and Object Concepts: Definitions:** Class, Instance, Object, Attribute, and Method. **Variable Scope:** Class Variables vs. Instance Variables. Scope and Lifetime of Variables in Python. **Class Relationships:** Superclass and Subclass Hierarchies. Understanding Is-A and Has-A Relationships **Constructors and Methods:** Constructors: The Role of the __init__() Method. Initializing Object Attributes. Understanding self: The Purpose of self in Instance Methods. Examples to Demonstrate self. Method Types: Instance Methods: Operate on Object Instances. Class Methods: Defined using @classmethod, Operate on the Class. Static Methods: Defined using @staticmethod, Utility Methods. **Method Resolution Order (MRO),** Overriding Methods in Subclasses, Using super() to Access Parent Class Methods. Special Methods and Operator Overloading **Custom Comparison and Magic Methods:** Overloading Operators: __eq__(), __lt__(), etc. Numeric Operations using Magic Methods: __add__(), __sub__(), __abs__(). Practical Applications: Writing Custom Classes with Overloaded Operators **Abstract Classes and Interfaces:** Introduction to Abstract Classes: Using the abc Module. Defining Abstract Methods. Implementing Abstract Classes Interfaces: Role in Designing Extendable Architectures. Comparison with Abstract Classes **Projects and Real-World Applications:** Mini-Projects to Demonstrate OOP Concepts: Employee Management System, Library Management System, Banking Application.	20
II	**GUI Programming with Tkinter:** Introduction to Widgets/Controls Overview of Common Widgets: Buttons, Labels, Text Fields, Checkboxes, and Radio Buttons. Simple Examples: Creating a basic window with interactive features. **Windows and Layout Management:** Customizing Window Titles and Title Bars. Arranging Widgets with Layout Managers: **Grid**: Organizing widgets in rows and columns. **Pack**: Stacking widgets vertically or horizontally. **Place**: Specifying exact positions for widgets. Event-Driven Programming, Basics of Event Binding using the bind() Method. Writing Callback Functions to Handle User Actions. Practical Examples: Adding Button Click Actions. Handling Mouse Events. **Network Programming**: Core Networking Concepts, Understanding Domains, IP Addresses, Ports, and Protocols. Real-World Applications: Browsers, Chat Apps, and File Transfers. **Socket Programming:** Introduction to the socket Module. Building Client-Server Communication: Sending and Receiving Data. Error Handling to Prevent Crashes. Practical Example: Creating a Basic Chat and Data Transfer Application. **Python Modules for Data Handling and Visualization** NumPy for Data Handling: Creating Arrays and Performing Basic Operations: Adding, Subtracting, and Multiplying Arrays. Simple Statistical Functions: Calculating Mean, Median, and Sum. Hands-On Practice: Analyzing Small Datasets. **Matplotlib for Data Visualization**: Creating Basic Graphs: Line, Bar, Scatter, and Pie Charts. Customizing Graphs: Adding Titles, Labels, and Legends. Practical Example: Visualizing Simple Data Trends. **Hands-On Projects and Activities** GUI Mini-Project Build a Tkinter-Based Application: Calculator, To-Do List, or Expense Tracker. Networking Mini-Project Create a Simple Chat Application Using Sockets. Data Visualization Mini-Project Visualize Real-World Data: Class Performance, Personal Expenses, or Weather Trends.	20

Text Books:

- Irv Kalb, Object-Oriented Python: Master OOP by Building Real-World Applications, O'Reilly Media.
- Burkhard Meier, Python GUI Programming with Tkinter: Develop Responsive and Powerful GUI Applications with Tkinter, Packt Publishing.

- Dusty Phillips, Python 3 Object-Oriented Programming: Build Robust and Maintainable Software with Object-Oriented Design Patterns, Packt Publishing.

Reference Books:

- John Zelle, Python Programming: An Introduction to Computer Science, Franklin, Beedle & Associates Inc.
- Mark Lutz, Programming Python: Powerful Object-Oriented Programming, O'Reilly Media.

Outcome: Upon completion of this course, the students will be able to:

- CO1: Demonstrate a clear understanding of Object-Oriented Programming concepts such as encapsulation, inheritance, and polymorphism.
- CO2: Design and implement advanced Python applications using OOP principles and real-world problem-solving skills.
- CO3: Create user-friendly GUI applications utilizing the Tkinter library and event-driven programming techniques.
- CO4: Apply networking principles to develop Python-based client-server applications using socket programming.
- CO5: Leverage Python libraries like NumPy and Matplotlib to perform data analysis and visualization effectively.