Course Objectives

After completing the course in Programming Fundamentals, students will be able to:

Course student learning outcomes:

- Solve problems by writing programs using standard language elements such as data declarations, arithmetic operations, conditional statements, loops, and functions.
- List and explain the key concepts of object-oriented development: inheritance, abstraction, information hiding, and polymorphism.
- Describe problems that typically plague software: rigidity, fragility, and immobility.
- Define the following object-oriented patterns: Factory, Singleton, Delegation, and Model-View-Controller.
- Define and provide examples for object-oriented design principles, and be able to describe SOLID: Single responsibility principle, Open-closed principle, Liskov substitution principle, Interface segregation principle, and Dependency inversion principle.
- Write class definitions and create objects from them.
- Declare and use special types of functions for classes, including constructors, accessors, mutators, and properties.
- Create hierarchies of classes that start with abstract base classes and add functionality in descendant classes.
- Design an object-oriented program in UML (Unified Modeling Language) that is organized around a set of classes whose objects interact.
- Describe what exceptions are and write programs that deal with them.
- Perform screen-scraping by retrieving data from a website.
- Write programs that use various collections.
- Use generic data types in programs.
- Work with collections of objects from related classes polymorphically.
- Explain the difference between classes and interfaces.
- Define interfaces that specify behaviors that particular objects must have.
- Perform input and output with text file streams.
- Perform input and output with XML file streams and serialization.
- Use an API as a reference when writing programs.
- Build attractive, intuitive graphical user interfaces.
- Write programs that use a graphical interface and manage user events using event-handling.
- Describe and use the client-server computing model
- Define serialization.
- Compare the advantages and disadvantages of various serialization sources and destinations.
- Write a program that stores and retrieves data with a relational database.
- Describe how Java achieves cross-platform compatibility.
- Distinguish between heavyweight and lightweight components.
- Define callback functions as they relate to event handling.
- Respond to user events in Java and Python.
- Describe how layout managers arrange components.
- Write unit tests to verify the correctness of software modules.
- Manage programming projects using GIT.

Program student learning outcomes:

1. Analyze a complex computing problem and apply principles of computing and other relevant disciplines to identify solutions.
2. Design, implement, and evaluate a computing-based solution to meet a given set of computing requirements in the context of the program's discipline.