Elective: Programming Paradigms for Complex Problems - Case Studies in Python

Teaching Scheme:
Lectures:4 Hrs/Week

Examination Scheme
Theory: 100 Marks

Course Objectives:

Why does one write programs? To solve **problems**.

The process of **refining** a problem to a solution is an engineering science – the **science** of programming.

In this course we will explore an approach to programming that…

- approaches problems using foundational principles
- uses Python as the programming language for hands-on demonstration
- scales up toy problems to realistic case studies

Examples and case studies will be drawn from the following areas:

- Combinatorics
- Language processing – computer and natural
- Games
- Finance
- Graphics
- System Administration Scripting

Course Outcomes:

This course brings together major programming idioms and shows how they can be used to solve different problems. The emphasis is on creating an understanding of these idioms through carefully chosen problems and developing solutions in the Python language.

By the end of the course, the student will know:

a. How imperative and declarative programming are related

b. How to reason logically about program control and data structures

c. Informal introduction to program semantics

d. How types are related to values and operations on data

e. Appreciate the pros and cons of mutability in data structures

f. Recursion in control and data structures

g. Understand Modularity Object oriented-ness of programs

h. Have familiarity with lightweight software analysis tools

Unit	Content	Teaching Scheme (Hours / Unit)
I	Imperative vs Declarative Programming - Why we need both views **Introduction to formal reasoning** - Predicates and Invariants **Reasoning about control** Developing a program from its intended goal (or postcondition) **Reasoning about data** Data is organized around *types* which may be *concrete* – defined by its *values*, or *abstract* – defined by its *operations*	5
II	'Typeful' Thinking - **Type systems:** background and examples, Type calculus, Tools for 'typeful' thinking	6
III	Value vs Object Orientation - State: essential vs incidental Value and object oriented data structures Data driven programming Functional programming	10
IV	**Recursion** - Recursive functions (the usual recursion), Recursive data (lazy data structures), Code mirrors data, Structural induction	4
V	**Object Oriented Programming** - Abstract data types, Type definition and extension in a typeless world, Coroutines/generators/iterators, Properties, Metaprogramming, Scripting: system management principles, techniques, do's and don't's	10
VI	**Modular Programming** - Namespaces: concepts, language features, tools; dicionaries, modules, packages - Installation, deployment, version management	5

Text Books:

1. Learning Python by Mark Lutz; O Reilly
2. Core Python Programming by Chun; Pearson LPE

Reference Books:

1. Python in a Nutshell by Alex Martelli; O Reilly
2. Python Language and Library manuals: https://docs.python.org
3. Types and programming languages by Pierce; MIT Press