		L	T	P	C	Int	Ext
CH/AD/CD/CE/ CM/CS/EC/EE/ IT/ME/QC113	**COMPUTATIONAL THINKING AND PROBLEM SOLVING WITH PYTHON** **Semester I [First Year]**	**3**	**-**	**-**	**3.0**	**40**	**60**

COURSE OUTLINE:

This course introduces first-year engineering students to computational thinking and its translation into working Python programs, establishing the problem-solving foundation that underlies all subsequent computing work. It develops the discipline of decomposing problems, recognizing patterns, abstracting details, and designing algorithms and flowcharts before writing code. Building on this foundation, the course covers Python control structures, core data structures, functions and recursion, file and exception handling, and a first exposure to data analysis and visualization through a capstone project. The course is common to all branches and feeds directly into higher-semester courses in data structures, object-oriented programming, and branch specific computational and data - analytics electives.

COURSE OBJECTIVES:

1. To introduce the fundamental pillars of computational thinking - decomposition, pattern recognition, abstraction, and algorithmic thinking - for systematic problem-solving.
2. To develop skills in designing algorithms, pseudocode, and flowcharts as precursors to programming.
3. To enable proficiency in Python programming using control structures and core data structures such as strings, lists, tuples, and dictionaries.
4. To build the ability to write modular, reusable Python code using functions, recursion, and built-in library modules.
5. To develop competence in file handling, exception handling, and basic data analysis and visualization using Python, culminating in an integrated capstone project.

COURSE OUTCOMES:

After successful completion of the course, the students are able to

CO	Course Outcome	RBTL
CO1	understand the pillars of computational thinking and apply them to decompose problems, design algorithms, and construct flowcharts.	Apply (BL3)
CO2	apply Python control structures - conditionals, loops, and loop-control statements - to implement solution logic for a given problem.	Apply (BL3)
CO3	apply appropriate Python data structures - strings, lists, tuples, and dictionaries - to organize, access, and manipulate data.	Apply (BL3)
CO4	apply functions, recursion, and modular programming techniques, including built-in library modules, to build reusable Python code.	Apply (BL3)
CO5	apply file handling, exception handling, and basic data-analysis tools in Python to complete an integrated end-to-end programming project.	Apply (BL3)

CO - PO / PSO Mapping

CO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PSO1	PSO2
CO1	3	2	1	1	2	-	-	-	-	-	1	-	-
CO2	3	3	2	1	3	-	-	1	-	-	1	-	-
CO3	3	2	2	1	3	-	-	1	-	-	1	-	-
CO4	3	2	2	1	3	-	-	1	1	-	2	-	-
CO5	2	3	2	2	3	1	-	1	2	-	2	-	-

UNIT I [CO:1] **(9)**

Computational Thinking, Algorithm Design & Flowcharting

Computational thinking pillars - decomposition, pattern recognition, abstraction, algorithmic thinking, and the problem-solving process, Algorithm design - characteristics of a good algorithm, stepwise refinement, pseudocode, Flowcharts - standard symbols, construction rules, and manual tracing for verification, Translating algorithms and flowcharts into Python code - variables, data types, operators, and input/output, Introduction to good coding practice - docstrings, inline comments, and print-based debugging, threaded through every unit from here on.

AI Integration with Contents:

Students use an AI coding assistant to generate a pseudocode draft for a simple problem, then critically compare it against their own hand-written pseudocode to identify gaps in decomposition and correctness before coding it in Python.

Unit Learning Outcomes:

1.Decompose a simple problem into a step-by-step algorithm and represent it as pseudocode and a flowchart.

2.Translate a verified flowchart into a working Python program using variables, data types, and basic I/O.

Practical Examples for In-Class Explanation:

1.Tracing a flowchart to compute the largest of three numbers.

2.Converting a pseudocode for calculating simple interest into a Python program.

Exercise / Practice Problems:

1.Draw a flowchart and write pseudocode to check whether a given number is even or odd.

2.Write a Python program to swap two numbers without using a third variable.

3.Design an algorithm and flowchart to convert temperature from Celsius to Fahrenheit.

UNIT II [CO:2] **(9)**

Control Structures

Conditional statements - as abstraction over multiple cases, Loops (for, while) - as pattern recognition of repetition, Nested loops and loop control (break, continue, pass) - as decomposition of complex repetitive logic.

AI Integration with Contents:

Students prompt an AI assistant to suggest two different loop-based approaches (for vs while) to the same repetitive task, then evaluate which pattern better fits the problem and justify their choice.

Unit Learning Outcomes:

1.Select and apply the appropriate conditional construct to abstract a multi-case decision problem.

2.Apply for, while, and nested loops with correct loop-control statements to implement repetitive logic.

Practical Examples for In-Class Explanation:

1.Using if-elif-else to classify a student's grade from marks.

2.Using nested for loops to print a right-angled triangle number pattern.

Exercise / Practice Problems:

1.Write a Python program to check whether a given year is a leap year.

2.Write a program to print the multiplication table of a number using a for loop.

3.Write a program that uses break and continue to find the first prime number greater than a given input.

UNIT III [CO:3] **(9)**

Data Structures - Strings, Lists, Tuples & Dictionaries

Strings - pattern matching and abstraction of text as a sequence, Lists - abstraction of ordered, mutable collections, Tuples and dictionaries - abstraction of fixed and key-based data.

AI Integration with Contents:

Students ask an AI assistant to suggest the most suitable data structure (list, tuple, or dictionary) for a given real-world dataset and then justify or challenge the suggestion based on mutability and access-pattern requirements.

Unit Learning Outcomes:

1.Apply string operations and built-in methods to process and analyze text data.

2.Choose and apply the appropriate data structure - list, tuple, or dictionary - to organize a given dataset.

Practical Examples for In-Class Explanation:

1.Using string slicing and methods to check whether a word is a palindrome.

2.Using a dictionary to store and look up student roll numbers against names.

Exercise / Practice Problems:

1.Write a Python program to count the frequency of each word in a sentence using a dictionary.

2.Write a program to find the second-largest number in a list without using built-in sort.

3.Write a program that uses a tuple to store and display the coordinates of a set of points.

UNIT IV [CO:4] **(9)**

Functions & Modular Programming

Functions - decomposition of a program into independent, reusable sub-problems, Recursion - algorithmic thinking applied to self-similar sub-problems, Basic algorithmic efficiency - informal comparison of iterative vs. recursive approaches and their time trade-offs, Modules and built-in libraries, including responsible AI-assisted coding practices.

AI Integration with Contents:

Students use an AI assistant to generate a recursive function for a problem, then manually trace its call stack and compare its readability and efficiency against an iterative version they write themselves, documenting where the AI-generated code needed correction.

Unit Learning Outcomes:

1.Decompose a program into functions with appropriate parameters and return values.

2.Apply recursion to solve a self-similar problem and informally compare it with an iterative solution.

Practical Examples for In-Class Explanation:

1.Writing a function to compute the factorial of a number iteratively and recursively.

2.Using the math and random modules to build a simple number-guessing program.

Exercise / Practice Problems:

1.Write a recursive Python function to generate the first n terms of the Fibonacci series.

2.Write a function with default arguments to compute the area of a rectangle or square.

3.Write a modular program that uses at least three separate functions to validate, process, and display user input.

UNIT V [CO:5] **(9)**

File Handling, Exception Handling, Data Analysis & Capstone Project

File handling - decomposition of a data workflow into read/process/write stages, Exception handling - abstraction of error cases into predictable, manageable patterns, Introduction to data analysis and visualization (NumPy/Pandas/Matplotlib basics) - pattern recognition in real datasets, Capstone integration - combining all prior units into one end-to-end project.

AI Integration with Contents:

Students use an AI assistant to help draft exception-handling code for anticipated error cases in their capstone project, then test it against unexpected inputs to identify cases the AI missed, reinforcing that AI suggestions must be verified rather than trusted outright.

Unit Learning Outcomes:

1.Apply file handling and exception handling to build a robust, error-tolerant Python program.

2.Apply basic NumPy/Pandas/Matplotlib operations to summarize and visualize a simple dataset within an integrated capstone project.

Practical Examples for In-Class Explanation:

1.Reading a text file of student marks and writing the computed averages to a new file, with exception handling for missing files.

2.Using Pandas to load a small CSV dataset and Matplotlib to plot a bar chart of the results.

Exercise / Practice Problems:

1.Write a Python program that reads numbers from a file and handles exceptions for non-numeric entries.

2.Write a program using try-except-finally to validate user input for a simple calculator.

3.Using a small sample CSV dataset, write a program to compute column-wise averages with Pandas and plot the result with Matplotlib.

LEARNING RESOURCES:

TEXT BOOK(s):

1. Downey, A. B. (2015). Think Python: How to think like a computer scientist (2nd ed.). O'Reilly Media.
2. Guttag, J. V. (2021). Introduction to computation and programming using Python (3rd ed.). MIT Press.

REFERENCE BOOK(s):

1. Deitel, P., & Deitel, H. (2019). Python for programmers. Pearson.
2. Zelle, J. M. (2016). Python programming: An introduction to computer science (3rd ed.). Franklin, Beedle & Associates.
3. McKinney, W. (2022). Python for data analysis (3rd ed.). O'Reilly Media.

WEB RESOURCES:

1. Python Software Foundation. (n.d.). The Python tutorial. Python Documentation. https://docs.python.org/3/tutorial/
2. NumPy Developers. (n.d.). NumPy documentation. https://numpy.org/doc/
3. pandas Development Team. (n.d.). pandas documentation. https://pandas.pydata.org/docs/