ALGORITHMIC THINKING WITH PYTHON

KTU 2024 SCHEME
(Common to all branches)

Dr. Jerrin Thomas Panachakel
Dr. Anusha S. P.
Abhiraj S. Kumar
(College of Engineering Trivandrum)

ALGORITHMIC THINKING WITH PYTHON

Dr. Jerrin Thomas Panachakel
Assistant Professor,
Department of Electronics & Communication Engineering

Dr. Anusha S. P.
Associate Professor
Department of Civil Engineering

Abhiraj S. Kumar
Project Associate

College of Engineering Trivandrum

Publisher Name
2024

PREFACE

Welcome to "Algorithmic Thinking with Python," a textbook for the university core course “UCEST105 Algorithmic Thinking with Python”, for first-year undergraduate engineering students of the 2024 scheme of engineering colleges affiliated with APJ Abdul Kalam Technological University. As part of the core curriculum, this course introduces students to the essentials of computational thinking and problem-solving using Python, ensuring that every student is equipped with critical skills relevant to their engineering discipline.

Python’s versatility and its widespread application across a spectrum of fields make it an invaluable tool for today’s engineers. Whether it’s developing software, enhancing cybersecurity measures, optimizing electronic circuits, or engineering novel materials and biological systems, Python serves as a bridge between theoretical concepts and practical application. Its simplicity and readability, combined with powerful libraries, make Python an ideal language for students embarking on their engineering journey.

Engineering disciplines at APJ Abdul Kalam Technological University are diverse, ranging from Computer Science to Food Technology, each engaging with technology in unique ways. Regardless of specialization, all students benefit from a robust understanding of algorithmic processes, making Python an ideal tool for fostering such skills. This course aims to unify these diverse branches by providing a common foundation in computational thinking that is adaptable to the various challenges encountered in the engineering field.

“Algorithmic Thinking with Python” encompasses:

- Essential Python programming techniques from basic to advanced levels.
- Detailed exploration of algorithmic paradigms such as brute force, divide-and-conquer, dynamic programming, and heuristics.
- Effective problem-solving strategies demonstrated through Python.
- A blend of theoretical concepts and practical applications with hands-on exercises

tailored to reinforce and apply learning in real-world contexts.

Our aim is not merely to teach Python but to embed a deep-seated capability for algorithmic thinking that students can apply across various engineering challenges. By the end of this course, students will have developed not only proficiency in Python programming but also an advanced approach to engineering problem-solving that will serve them throughout their careers.

Here's to embarking on a transformative learning experience that harnesses the power of Python to fuel your engineering aspirations!

Jerrin Thomas Panachakel, Anusha S.P, Abhiraj S. Kumar

CONTENTS

INDEX

SYLLABUS		
Module No.	Syllabus Description	Contact Hours
1	PROBLEM-SOLVING STRATEGIES: - Problem-solving strategies defined, Importance of understanding multiple problem-solving strategies, Trial and Error, Heuristics, Means-Ends Analysis, and Backtracking (Working backward). THE PROBLEM-SOLVING PROCESS: - Computer as a model of computation, Understanding the problem, Formulating a model, Developing an algorithm, Writing the program, Testing the program, and Evaluating the solution. ESSENTIALS OF PYTHON PROGRAMMING: - Creating and using variables in Python, Numeric and String data types in Python, Using the math module, Using the Python Standard Library for handling basic I/O - print, input, Python operators and their precedence.	7
2	ALGORITHM AND PSEUDOCODE REPRESENTATION:- Meaning and Definition of Pseudocode, Reasons for using pseudocode, The main constructs of pseudocode - Sequencing, selection (if-else structure, case structure) and repetition (for, while, repeat-until loops), Sample problems* FLOWCHARTS** :- Symbols used in creating a Flowchart - start and end, arithmetic calculations, input/output operation, decision (selection), module name (call), for loop (Hexagon), flow-lines, on-page connector, off-page connector.	9

	** - Evaluate an expression,* ***d=a+b*c****, find simple interest, determine the larger of two numbers, determine the smallest of three numbers, determine the grade earned by a student based on KTU grade scale (using if-else and case structures), print the numbers from 1 to 50 in descending order, find the sum of n numbers input by the user (using all the three loop variants), factorial of a number, largest of n numbers* ***(Not to be limited to these exercises. More can be worked out if time permits).*** ***** Only for visualizing the control flow of Algorithms. The use of tools like RAPTOR (https://raptor.martincarlisle.com/) is suggested. Flowcharts for the sample problems listed earlier may be discussed***	
3	SELECTION AND ITERATION USING PYTHON:- if-else, elif, for loop, range, while loop. Sequence data types in Python - list, tuple, set, strings, dictionary, Creating and using Arrays in Python (using Numpy library). DECOMPOSITION AND MODULARISATION* :- Problem decomposition as a strategy for solving complex problems, Modularisation, Motivation for modularisation, Defining and using functions in Python, Functions with multiple return values RECURSION:- Recursion Defined, Reasons for using Recursion, The Call Stack, Recursion and the Stack, Avoiding Circularity in Recursion, *Sample problems - Finding the nth Fibonacci number, greatest common divisor of two positive integers, the factorial of a positive integer, adding two positive integers, the sum of digits of a positive number **.*	10

	** The idea should be introduced and demonstrated using Merge sort, the problem of returning the top three integers from a list of n>=3 integers as examples.* ***(Not to be limited to these two exercises. More can be worked out if time permits).*** *** Not to be limited to these exercises. More can be worked out if time permits.*	
4	COMPUTATIONAL APPROACHES TO PROBLEM-SOLVING *(Introductory diagrammatic/algorithmic explanations only. Analysis not required):-* Brute-force Approach – *Example: Padlock, Password guessing* Divide-and-conquer Approach – *Example: The Merge Sort Algorithm* Advantages of Divide and Conquer Approach - Disadvantages of the Divide and Conquer Approach Dynamic Programming Approach *Example: Fibonacci series* - Recursion vs Dynamic Programming Greedy Algorithm Approach *- Example: Given an array of positive integers each indicating the completion time for a task, find the maximum number of tasks that can be completed in the limited amount of time that you have.* - Motivations for the Greedy Approach - Characteristics of the Greedy Algorithm - Greedy Algorithms vs Dynamic Programming Randomized Approach	10

	- Example 1: A company selling jeans gives a coupon for each pair of jeans. There are n different coupons. Collecting n different coupons would give you free jeans. How many jeans do you expect to buy before getting a free one? *- Example 2: n people go to a party and drop off their hats to a hat check person. When the party is over, a different hat-check person is on duty and returns the n hats randomly back to each person. What is the expected number of people who get back their hats?* - Motivations for the Randomized Approach	