



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

Preface	3
Contents	5
1 Introduction to Python Programming	9
1 PROBLEM SOLVING STRATEGIES	10
1.1 Types of Problems in Problem Solving	10
1.2 Importance of Understanding Problem-Solving Strategies	14
1.3 Trial and Error	15
1.4 Algorithm	15
1.5 Heuristic	15
1.6 Means-Ends Analysis	16
1.7 Backtracking	16
1.8 Application of Problem-Solving Strategies	20
2 PROBLEM SOLVING PROCESS	21
2.1 Computer as a Model of Computation	21
2.2 Understanding the Problem	22
2.3 Formulating a Model	23
2.4 Developing an Algorithm	23
2.5 Writing the Program	23
2.6 Testing the Program	24
2.7 Evaluating the Solution	24
3 ESSENTIALS OF PYTHON	25
3.1 Integrated Development Environments (IDEs) for Python	27
3.2 Writing your First program in Python	31
3.3 Using comments in Python	32
3.4 The print() function	32
3.5 The input() function	33
3.6 Common Pitfalls: Understanding and Fixing Errors in Python .	35
3.7 Creating and Using Variables in Python	36

3.8	Keywords	38
3.9	Basic Operators	40
3.10	Numeric types in Python	56
3.11	Using strings in Python	61
3.12	Basic String Operations in Python	62
3.13	Using the math module	69
3.14	Worked Out Examples	78
4	Question and Answers	83
2	Algorithm Basics: Pseudocode & Flowcharts	97
1	ALGORITHM	98
1.1	Reasons for Algorithms	98
1.2	Steps Involved in Algorithm Development	98
1.3	Characteristics of an Algorithm	99
1.4	Representation of Algorithms	100
2	PSEUDOCODE	102
2.1	Reasons for Using Pseudocode	102
2.2	Main Constructs in Pseudocode	104
2.3	Pseudocode Worked Examples	111
3	FLOWCHART	121
3.1	Symbols Used in Creating a Flowchart	121
3.2	Common Types of Flowcharts	127
3.3	Areas for Using Flowcharts	127
3.4	Considerations in Flowcharting	128
3.5	Differences Between Algorithm and Flowchart	128
3.6	Flowchart Examples	129
4	Questions and Answers	137
3	Advanced Programming Concepts in Python	143
1	Selection and Iteration using Python	144
1.1	Understanding Conditional Statements	144
1.2	Iteration Control (Loops in Python)	146
1.3	Combining <code>if</code> Statements with Loops	148
1.4	Sequence Data Types in Python	152
1.5	Worked Out Example	159
1.6	Creating and Using Arrays in Python	168
2	Decomposition and Modularization in Programming	182
2.1	Decomposition	183
2.2	Modularization	185

2.3	Basic Properties of Modularity	187
2.4	Defining and Using Functions in Python	189
2.5	Functions with Multiple Return Values	193
3	Recursion	197
3.1	Recursion Defined	197
3.2	Reasons for Using Recursion	197
3.3	The Call Stack	198
3.4	Recursion and the Stack	198
3.5	Avoiding Circularity in Recursion	200
3.6	Overhead of Recursion	201
3.7	Sample Problems	201
4	Questions and Answers	208
4	Computational Approaches To Problem-Solving	219
1	Brute-Force Approach	220
1.1	Understanding the Brute-Force Approach	220
1.2	Example 1: Unlocking a Padlock	220
1.3	Example 2: Password Guessing	221
1.4	Limitations of the Brute-Force Approach	223
2	Divide-and-Conquer Approach in Python Programming	223
2.1	Understanding the Divide-and-Conquer Approach	223
2.2	Example: The Merge Sort Algorithm	224
2.3	Advantages of Divide-and-Conquer Approach	226
2.4	Disadvantages of Divide-and-Conquer Approach	226
3	Generating Fibonacci Series in Python	227
3.1	Recursive Approach	227
3.2	Iterative Approach	228
3.3	Dynamic Programming Approach	229
3.4	Comparison of Approaches	231
3.5	Recursion vs Dynamic Programming	232
4	Greedy Algorithm Approach	233
4.1	Example: Task Completion Problem	234
4.2	Motivations for the Greedy Approach	235
4.3	Characteristics of the Greedy Algorithm	235
4.4	Greedy Algorithms vs Dynamic Programming	236
5	Randomized Approach	236
5.1	Example 1: Coupon Collector's Problem	236
5.2	Example 2: Hat Check Problem	238

5.3	Motivations for the Randomized Approach	239
6	Questions and Answers	240
EXERCISES		251
Index		307

INDEX

- abs() function, 69
- Algorithm, 15, 98
- Array module, 170
- Arrays, 168
- Backtracking, 16
- Basic Operators, 40
 - Arithmetic Operators, 40
 - Assignment Operators, 52
 - Bitwise Operators, 45
 - Comparison Operators, 42
 - Logical Operators, 44
 - Membership Operators, 54
- Basic Symbols in Flowchart, 121
- Brute-Force Approach, 220
- Call Stack, 198
- Combining if statements with loop, 148
- Comments, 32
- Comments in Pseudocode, 110
- Conditional Statements, 144
 - elif statement, 145
 - if statement, 144
 - if-else statement, 144
- Conditional Statements in Pseudocode, 104
 - Case or Switch, 106
 - If-Then-Else, 104
 - Nested If-Then-Else, 105
- Constructs in Pseudocode, 104
- Coupon Collector's Problem, 236
- Decomposition, 183
- Divide-and-Conquer, 223
- Dynamic Programming Approach, 229
- Errors in Python, 35
 - Runtime errors, 35
- Fibonacci Series, 227
- Flowchart, 100, 121
- Functions in Python, 189
- Google Colab, 28
- Greedy Algorithm, 233
- Hat Check Problem, 238
- Heuristic, 15
- IDE, 27
- IDLE, 31
- input() function, 33
- Keywords, 38
- Loops, 146
 - for loop, 146
 - while loop, 148
- Loops in Pseudocode, 107
 - For Loop, 107
 - Repeat-Until-Loop, 108
 - While Loop, 107
- math module, 69

- Basic Mathematical Functions, 70
- Hyperbolic Functions, 75
- Mathematical Constant, 70
- Rounding and Remainder Functions, 76
- Special Functions, 77
- Trigonometric Functions, 73
- max() function, 69
- Means-Ends Analysis, 16
- Merge Sort Algorithm, 224
- min() function, 69
- Modularization, 185
- Naming variables, 36
- Natural Language, 101
- Numeric Type Conversion, 60
- Numeric Types, 56
 - Complex Numbers, 58
 - Floating-Point Numbers, 57
 - Integers, 56
- numpy arrays, 172
- Padlock unlocking, 220
- Password Guessing, 221
- Precedence Table for Operators, 55
- print() function, 32
- Problem Solving Process, 21
- Problem Solving Strategies, 10
- Pseudocode, 100, 102
- Pseudocode Function call, 108
- Python, 25
- Randomized Approach, 236
- Recursion, 197
- Rounding Decimals, 59
- Sequence Data Types, 152
 - Dictionaries, 157
 - Lists, 152
 - Sets, 156
 - Strings, 155
 - Tuples, 154
- Stack Overflow, 199
- String Operations, 62
- Strings, 61
- Trial and Error, 15
- Types of Flowchart, 127
- Types of problems, 10, 13, 14
 - Algorithmic Problems, 12
 - Analytical Problems, 10
 - Complex Problems, 11
 - Convergent Problems, 12
 - Creative Problems, 11
 - Divergent Problems, 12
 - Ethical Problems, 13
 - Heuristic Problems, 12
 - Ill-Defined Problems, 10
 - Simple Problems, 11
 - Well-Defined Problems, 10
- Variable, 36
- ZeroDivisionError, 36

SYLLABUS		
Module No.	Syllabus Description	Contact Hours
1	<p>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).</p> <p>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.</p> <p>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.</p>	7
2	<p>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*</p> <p>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.</p>	9

	<p><i>* - 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).</i></p> <p><i>** 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</i></p>	
3	<p>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).</p> <p>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</p> <p>RECURSION:- Recursion Defined, Reasons for using Recursion, The Call Stack, Recursion and the Stack, Avoiding Circularity in Recursion, <i>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</i> **.</p>	10

	<p><i>* The idea should be introduced and demonstrated using Merge sort, the problem of returning the top three integers from a list of $n \geq 3$ integers as examples. (Not to be limited to these two exercises. More can be worked out if time permits).</i></p> <p><i>** Not to be limited to these exercises. More can be worked out if time permits.</i></p>	
4	<p>COMPUTATIONAL APPROACHES TO PROBLEM-SOLVING (<i>Introductory diagrammatic/algorithmic explanations only. Analysis not required</i>):-</p> <p>Brute-force Approach – <i>Example: Padlock, Password guessing</i></p> <p>Divide-and-conquer Approach – <i>Example: The Merge Sort Algorithm</i></p> <p>Advantages of Divide and Conquer Approach - Disadvantages of the Divide and Conquer Approach</p> <p>Dynamic Programming Approach <i>Example: Fibonacci series</i></p> <p>- Recursion vs Dynamic Programming</p> <p>Greedy Algorithm Approach - <i>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.</i></p> <p>- Motivations for the Greedy Approach - Characteristics of the Greedy Algorithm - Greedy Algorithms vs Dynamic Programming</p> <p>Randomized Approach</p>	10

	<p>- <i>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?</i></p> <p>- <i>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?</i></p> <p>- Motivations for the Randomized Approach</p>	
--	---	--