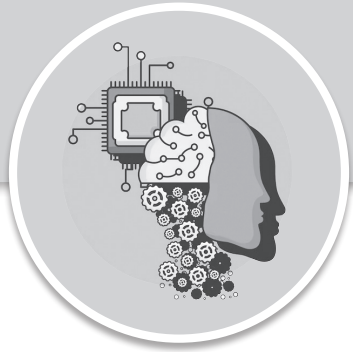
DATA SCIENCE & ARTIFICIAL INTELLIGENCE

Programming & Data Structures



Comprehensive Theory
with Solved Examples *and* Practice Questions

MADE EASY PUBLICATIONS

www.madeeasypublications.org

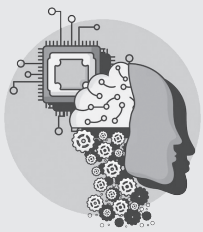
MADE EASY Publications Pvt. Ltd.

Corporate Office: 44-A/4, Kalu Sarai (Near Hauz Khas Metro Station), New Delhi-110016 | **Ph. :** 9021300500

Email : infomep@madeeasy.in | **Web :** www.madeeasypublications.org

Programming & Data Structures

© Copyright by MADE EASY Publications Pvt. Ltd.
All rights are reserved. No part of this publication may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photo-copying, recording or otherwise), without the prior written permission of the above mentioned publisher of this book.



MADE EASY Publications Pvt. Ltd. has taken due care in collecting the data and providing the solutions, before publishing this book. Inspite of this, if any inaccuracy or printing error occurs then **MADE EASY Publications Pvt. Ltd.** owes no responsibility. We will be grateful if you could point out any such error. Your suggestions will be appreciated.

EDITIONS

First Edition : 2025

Second Edition : 2026

CONTENTS

Programming & Data Structures

CHAPTER 5

CHAPTER 6

Programming and Data Structures

Goal of the Subject

Computer Science is not the study of programming. Programming, however, is an important part of what a computer scientist does. Programming is often the way that we create a representation for our solutions. Therefore, this language representation and the process of creating it becomes a fundamental part of the discipline.

A data structure is a specialized format for organizing and storing data. To manage the complexity of problems and the problem-solving process, computer scientists use abstractions to allow them to focus on the "big picture" without getting lost in the details. By creating models of the problem domain, we are able to utilize a better and more efficient problem-solving process. The implementation of an abstract data type, often referred to as a data structure, will require that we provide a physical view of the data using some collection of programming constructs and primitive data types.

General data structure types include the array, the file, the record, the table, the tree, and so on. Any data structure is designed to organize data to suit a specific purpose so that it can be accessed and worked with in appropriate ways. In computer programming, a data structure may be selected or designed to store data for the purpose of working on it with various algorithms.

Introduction

In this book we tried to keep the syllabus of Software Programming and Data structures around the GATE syllabus. Each topic required for GATE is crisply covered with illustrative examples and each chapter is provided with Student Assignment at the end of each chapter so that the students get the thorough revision of the topics that he/she had studied. This subject is carefully divided into seven chapters as described below.

1. **Programming Methodology:** In this chapter, we will study the different segments and their organization, variables, flow control statements, functions, recursion, lists, tuples, sets, and dictionaries.
2. **Stack:** In this chapter we will study the ADT of stack, operations on stack, applications and different types of notations evaluated by stack.
3. **Queue:** In this chapter we will study about the Queue, operations on queue, applications and finally we discuss different types of queues.
4. **Linked Lists:** In this chapter we will study types and applications of linked list, operations on linked list, priority queue and finally we discuss implementation of stack, queue and priority queue using lists.
5. **Trees:** In this chapter we introduce trees, their applications, types of trees (BST), different types tree traversals and finally we discuss operations on trees.
6. **Hashing Techniques:** In this chapter we introduce the Hash function, collision resolution techniques and comparisons of different collision techniques.

■■■■

CHAPTER

1 Programming Methodology

1.1 DATA SEGMENTS IN MEMORY

1. **Text (Code) Segment:** The Code Segment in Python contains compiled bytecode of the program. It includes function and class definitions, which are stored in memory for execution.

   Example:

```
def hello():
    print("MadeEasy, Delhi, World!") # Stored in the code segment
    hello()
```

   - The function definition is stored in the code segment, and only one copy is loaded into memory.

2. **Initialized Data Segment:** The Initialized Data Segment contains global and static variables that are initialized by the programmer. In Python, module-level variables act similarly.

   Example:

```
message = "MadeEasy" # Global variable (Stored in Initialized Data Segment)
readonly_message = ("MadeEasy",) # Tuple (Immutable, similar to read-only data)

def display():
    static_var = 1 # Local variable (Not stored in the data segment)
    print(message)
```

   - message is stored in memory as an initialized global variable.
   - Tuples `(readonly_message)` in Python act as read-only data.

3. **Uninitialized Data Segment (BSS Segment):** This segment stores global and static variables that are declared but not assigned an explicit value. In Python, an equivalent would be declaring a variable with none.

   Example:

```
uninit_var = None # Uninitialized global variable (Stored in BSS-Like
    segment)

class Example:
    static_var = None # Uninitialized class/static variable

print(uninit_var) # Output: None
```

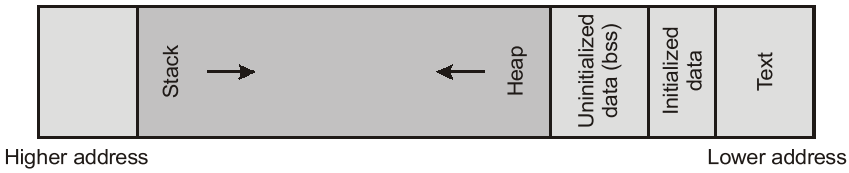
   - Variables declared but not initialized are stored with a default value `(None)`.

4. **Heap Segment:** The Heap Segment is used for dynamic memory allocation in Python. It grows and shrinks at runtime. Memory allocation in the heap is managed by Python's Garbage Collector (GC).
   Example:
```
import ctypes

ptr = ctypes.create_string_buffer(10) # Memory allocated in Heap
print("Memory allocated in Heap")
```
   - Python automatically manages heap memory using dynamic allocation.

5. **Stack Segment:** The Stack Segment is used for function calls and local variable storage. It follows LIFO (Last In, First Out) behavior.
   Example:
```
def recursive(n):
    if n == 0:
        return
    print(n)
    recursive(n - 1) # Recursive function calls create stack frames

recursive(5) # Function call stack grows
```
   - Each function call creates a new stack frame, and when recursion depth is too high, it causes a stack overflow.



1.2 IDENTIFIERS

An identifier is the name used to identify variables, functions, classes, modules, and objects in Python. It is essentially the user-defined name that helps in referring to different program elements.

1.2.1 Rules for Defining Identifiers in Python

To ensure valid and error-free code, Python enforces specific rules for naming identifiers:

1. Identifiers can contain letters (A-Z, a-z), digits (0-9), and underscores (_) only.
   - They cannot contain special characters like `@, $, %, &, *, -, +, !, etc.`

Example 1.1 What will be output of following program?

```
my_variable = 10
count1 = 5
_name = "Institute"
print("Made Easy")
```
Output: Made Easy

- `my-variable = 10 # Error: Hyphens are not allowed`
  `@name = "Institute" # Error: Special characters are not allowed`

Example 1.2 What will be output of following program?

```
my-variable = 10
@name = "Python"
print("Made Easy")
```

Output: Made Easy

2. Must Start with a Letter or an Underscore:
   - Identifiers must begin with a letter (A-Z or a-z) or an underscore (_), but not a digit (0-9).

Example 1.3 What will be output of following program?

```
name "Alice"
_var = 20
print("Made Easy")
```

Output: Made Easy

Example 1.4 What will be output of following program?

```
lst_name = "John X # Error: Cannot start with a digit
print("Made Easy")
```

Output:

```
ERROR!
Traceback (most recent call last):
    File "<main.py", line 1
    lst_name = "John" # Error: Cannot start with a digit
    ^

SyntaxError: invalid decimal literal
```

3. Cannot Be a Python Keyword:
   - Identifiers cannot be the same as Python's reserved keywords (e.g., if, for, class, import, etc.).

   Incorrect:

   ```
   class = "MadeEasy" # SyntaxError: 'class' is a reserved keyword
   def = 5000 # SyntaxError: 'def' is a reserved keyword
   ```

   Correct:

   ```
   class_name = "MadeEasy"
   define_value = 5000
   ```

   To check Python keywords:

   ```
   import keyword
   print(keyword.kwlist) # List of all reserved keywords in Python
   ```

4. Identifiers Are Case-Sensitive:
   - Python distinguishes between uppercase and lowercase letters.

   Correct:

   ```
   Name = "MadeEasy"
   name = "DA"
   print(Name) # Output: DA
   print(name) # Output: DA
   ```

   Here, Name and name are different identifiers.

5. No Spaces Allowed in Identifiers:
   - Identifiers cannot contain spaces. Use underscores (_) instead.

   Incorrect:
   ```
   user name = "MadeEasy" # Error: Spaces not allowed
   ```
   Correct:
   ```
   user_name = "MadeEasy" # Uses underscore instead of space
   ```
6. Cannot Use Special Characters:
   - Identifiers should not include special characters (`@, $, %, &, *, etc.`).

   Incorrect:
   ```
   price$ = 80000 # Error: Special character '$' not allowed
   my@var = 500 # Error: Special character '@' not allowed
   ```
   Correct:
   ```
   price_value = 80000
   my_var = 500
   ```
7. Use Meaningful Names:
   - Avoid using short or unclear variable names. Use descriptive names.

   Incorrect:
   ```
   x = 10 # Not descriptive
   y = 20
   ```
   Correct:
   ```
   age = 10 # Descriptive variable name
   salary = 20000
   ```
8. Use Proper Naming Conventions:
   - Variables and Functions: Use `snake_case (user_name, calculate_area)`.
   - Class Names: Use `PascalCase (Person, CarModel)`.
   - Constants: Use `UPPER_CASE (MAX_SPEED, PI_VALUE)`.

Correct Naming Examples:
```
# Variable (snake_case)
student_name = "Piyush"

# Function (snake_case)
def calculate_area(radius):
    return 3.14 * radius * radius

# Class Name (PascalCase)
class EmployeeDetails:
    pass

# Constant (UPPER_CASE)
MAX_LIMIT = 500
```

Student's Assignments

Q.1 Consider the following Python code:

```
def fun (a, * args, s = '!'):
print(a, s)
for i in args:
    print(i, s)
fun(10, 20, s= '+')
```

What is output of the above code?

(a) 10 + 20+ (b) 1020
(c) +10 +20 (d) +10 20+

Q.2 Consider the following Python code:

```
def print_it (i, a, s, * args):
    print( )
    print(i, a, s, end = ' ')
    for var in args:
        print(var, end = ' ')
    print_it (a = 10, s = 'Niraj', i = 30)
```

What will be output of the above code?

(a) Niraj 1030 (b) 3010 Niraj
(c) 1030 Niraj (d) Compile time error

Q.3 What will be the output of Python code :

```
i = 1
j = 1
for i in range(1, 11):
    if(i % 3 != 0):
        j += 2
        continue
    if(j % 3 == 0):
        break
print(i + j)
```

Q.4 Consider the following Python code:

```
x = 20
Class staticDemo:
    y = x
def main( )
    if x == staticDemo.y;
        print("Equal")
    else:
        print("Not Equal")
if_name_ == "_main_"
main( )
```

What will be the output above code?

(a) Equal (b) Not equal
(c) Run time error (d) Compile time error

Q.5 Consider the following Python Code:

```
det fun( ):
    if not hasattr(fun, 'i'):
        fun.i = 1
    j = 5
    print(fun.i, end = " ")
    print(j, end = " ")
    fun.i += 1
fun( )
fun( )
```

(a) 25 15 35 (b) 15 25
(c) 15 16 35 (d) 26 37 48

Q.6 What will be the out of the program:

```
def reverse (num):
    if num = = 0:
        return 0
    else:
        print (num, end = ' ')
            reverse (num – 1)
        num = 5
        reverse (num)
```

(a) 5, 4, 3, 2, 1 (b) 1, 2, 3, 4, 5
(c) 5, 4, 3, 2, 1, 0 (d) Infinite loop

Q.7 Consider the following Python code:

```
def main( )
    i = 10
    if i == 5:
        print("I am in case 5")
    else:
        print("I am in default case")
    print("I am in default case 3")
    print("I am in default case 1")
    print("I am in default case 2")
```

What is the output of above code?

(a) I am in default case
I am in case 3
I am in case 1
I am in case 2
(b) I am in default case

(c) I am in default case
I am in case 3
I am in case 2
I am in case 1
I am in case 5

(d) I am in case 3
I am in case 2
I am in case 1
I am in case 5
I am in default case

Q.8 Which of the following is true?

(a) Inorder traversal of a binary tree is always sorted order.

(b) Python programming does not support machine learning applications.

(c) Towers of Hanoi applications internally uses queue data structure.

(d) None of the above

Q.9 Consider the following Python code:

```
def call1(n):
    if n == 1:
        return
    print(n, end=" ")  # First print statement
    call2(n – 1) # Recursive call
    print(n, end=" ")  # Second print statement
def call2(n):
    if n == 1:
        return
    print(n, end=" ") # First print statement
    call1(n – 1) # Recursive call
    print(n, end=" ") # Second print statement
# Calling call2 with n = 6
call2(6)
```

(a) 5 6 4 3 2 2 3 4 5 6

(b) 6 5 4 3 2 2 3 4 5 6

(c) 6 5 4 3 3 2 2 4 5 6

(d) 5 6 3 4 3 2 2 4 5 6

Q.10 Consider the following Python code:

```
def f(a):
    print(a, end=" ") |
    a += 1
    return a + 1
b = 1
b = f(b)
b = f(b)
b = f(1 + f(b))
```

(a) 8 5 1 3 (b) 1 3 8 5

(c) 1 3 5 8 (d) 8 5 3 1

Answers Keys

1. (a)	**2.** (b)	**3.** (15)	**4.** (a)	**5.** (b)
6. (a)	**7.** (a)	**8.** (d)	**9.** (b)	**10.** (c)

Explanations

4. (a)

$x = 20$

StaticDemo.*y* is also set to 20 in the main( ) function (x) is compared to StaticDemo.*y*

Since, both are 20 then output will be "Equal".

5. (b)

The function with static variable 'fun.*i*' that retains its value across call, and a local variable '*j*' that reinitialized each time. In two calls, 'fun.*i*' starts at 1 and is incremented with each call, while *j* is always 5.

Output will be 15 25.

6. (a)

Here reverse function prints the current value of num and recursively calls itself with num – 1 until num reaches 0. For each recursive call, it prints num before making the next call. The base as termination the recursion without printing, the output 5, 4, 3, 2, 1.