

Python Programming (4311601) - Winter 2024 Solution

Milav Dabgar

January 13, 2025

Question Question 1(a) [03 marks]

Define Problem Solving, Algorithm and Pseudo Code.

Solution

Definitions:

Table 1. Core Concepts

Term	Definition
Problem Solving	Systematic process of finding solutions to complex issues using logical thinking
Algorithm	Step-by-step procedure to solve a problem with finite operations
Pseudo Code	Informal description of program logic using plain English-like syntax

Key Points:

- **Problem Solving:** Breaking down complex problems into manageable steps
- **Algorithm:** Must be finite, definite, effective, and produce correct output
- **Pseudo Code:** Bridge between human language and programming code

Mnemonic

“PAP - Problem, Algorithm, Pseudo”

Question Question 1(b) [04 marks]

Explain various Flowchart Symbols. Design a Flowchart to find maximum number out of two given numbers

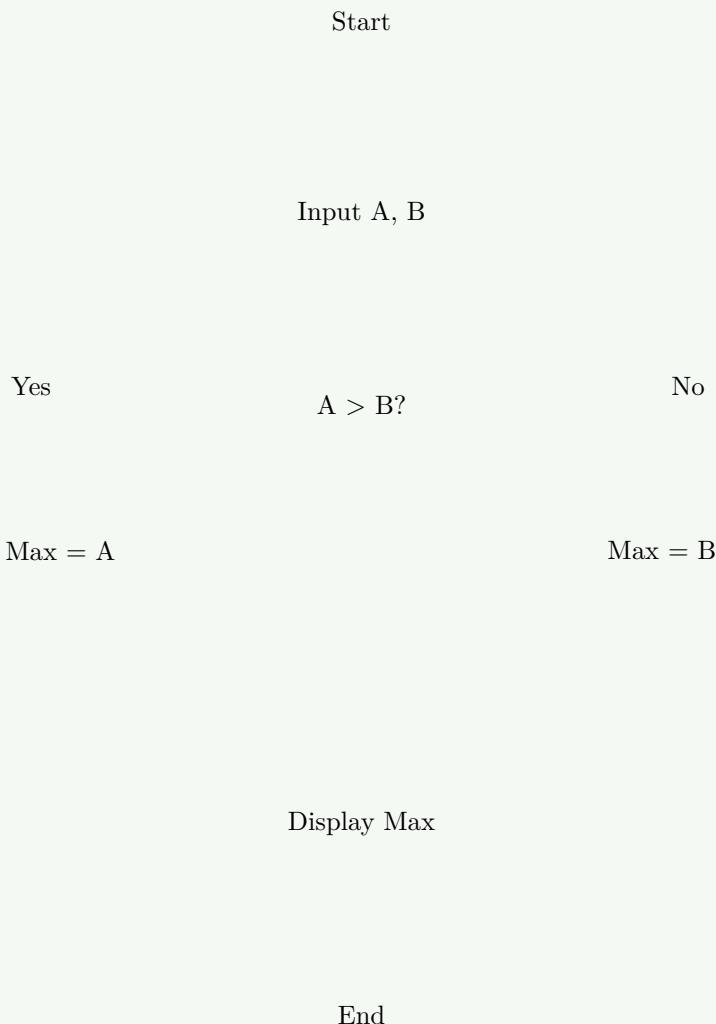
Solution

Flowchart Symbols:

Table 2. Symbols

Symbol	Shape	Purpose
Oval		Start/End
Rectangle		Process/Action
Diamond		Decision
Parallelogram		Input/Output

Flowchart for Maximum of Two Numbers:

**Explanation:**

- **Start/End:** Entry and exit points
- **Input/Output:** Data flow operations
- **Decision:** Conditional branching
- **Process:** Computational steps

Mnemonic

“SIPO - Start, Input, Process, Output”

Question Question 1(c) [07 marks]

List out various arithmetic operators of python. Write Python Code that performs various arithmetic operations.

Solution**Arithmetic Operators:**

Table 3. Arithmetic Operators

Operator	Symbol	Example	Result
Addition	+	5 + 3	8
Subtraction	-	5 - 3	2
Multiplication	*	5 * 3	15
Division	/	5 / 3	1.667
Floor Division	//	5 // 3	1
Modulus	%	5 % 3	2
Exponentiation	**	5 ** 3	125

Code:

```

1 a = 10
2 b = 3
3 print(f"Addition: {a + b}")
4 print(f"Subtraction: {a - b}")
5 print(f"Multiplication: {a * b}")
6 print(f"Division: {a / b}")
7 print(f"Floor Division: {a // b}")
8 print(f"Modulus: {a % b}")
9 print(f"Power: {a ** b}")

```

Mnemonic

“Add-Sub-Mul-Div-Floor-Mod-Pow”

Question Question 1(c OR) [07 marks]

List out various comparison operators of python. Write Python Code which performs various comparison operations.

Solution**Comparison Operators:****Table 4.** Comparison Operators

Operator	Symbol	Purpose	Example
Equal	==	Check equality	5 == 3 → False
Not Equal	!=	Check inequality	5 != 3 → True
Greater Than	>	Check greater	5 > 3 → True
Less Than	<	Check smaller	5 < 3 → False
Greater Equal	>=	Check greater/equal	5 >= 3 → True
Less Equal	<=	Check smaller/equal	5 <= 3 → False

Code:

```

1 x = 8
2 y = 5
3 print(f"Equal: {x == y}")
4 print(f"Not Equal: {x != y}")
5 print(f"Greater: {x > y}")
6 print(f"Less: {x < y}")
7 print(f"Greater Equal: {x >= y}")
8 print(f"Less Equal: {x <= y}")

```

Mnemonic

“Equal-Not-Greater-Less-GreaterEqual-LessEqual”

Question Question 2(a) [03 marks]

Write short note on membership operators.

Solution

Membership Operators:

Table 5. Membership Operators

Operator	Purpose	Example
in	Check if element exists	'a' in 'apple' → True
not in	Check if element doesn't exist	'z' not in 'apple' → True

Key Points:

- **in operator:** Returns True if element found in sequence
- **not in operator:** Returns True if element not found in sequence
- **Usage:** Lists, strings, tuples, dictionaries

Mnemonic

“In-Not-In for membership testing”

Question Question 2(b) [04 marks]

Define Python. Write down various applications of Python Programming.

Solution

Python Definition: High-level, interpreted programming language known for simplicity and readability.

Applications:

Table 6. Applications

Application Area	Examples
Web Development	Django, Flask frameworks
Data Science	NumPy, Pandas, Matplotlib
AI/ML	TensorFlow, Scikit-learn
Desktop Apps	Tkinter, PyQt
Game Development	Pygame library

Features:

- **Interpreted:** No compilation needed
- **Cross-platform:** Runs on multiple OS
- **Large libraries:** Extensive standard library

Mnemonic

“Web-Data-AI-Desktop-Games”

Question Question 2(c) [07 marks]

Write python program which calculates electricity bill using following details.

Solution

Table of Rates:

Table 7. Electricity Rates

Unit Range	Rate per Unit
≤ 100	Rs 5.00
101-200	Rs 7.50
201-300	Rs 10.00
≥ 301	Rs 15.00

Code:

```

1 units = int(input("Enter consumed units: "))

2
3 if units <= 100:
4     bill = units * 5.00
5 elif units <= 200:
6     bill = units * 7.50
7 elif units <= 300:
8     bill = units * 10.00
9 else:
10    bill = units * 15.00
11
12 print(f"Total Bill: Rs {bill}")

```

Explanation:

- **Conditional logic:** if-elif-else structure
- **Rate calculation:** Based on unit slabs
- **User input:** Interactive billing system

Mnemonic

“Input-Check-Calculate-Display”

Question Question 2(a OR) [03 marks]

Write short note on identity operators.

Solution

Identity Operators:

Table 8. Identity Operators

Operator	Purpose	Example
<code>is</code>	Check same object	<code>a is b</code>
<code>is not</code>	Check different object	<code>a is not b</code>

Key Points:

- **is operator:** Compares object identity, not values
- **is not operator:** Checks if objects are different
- **Memory comparison:** Checks same memory location

Mnemonic

“Is-IsNot for object identity”

Question Question 2(b OR) [04 marks]

What is indentation in Python? Explain various features of Python.

Solution

Indentation: Whitespace at line beginning to define code blocks.

Features:

Table 9. Python Features

Feature	Description
Simple Syntax	Easy to read and write
Interpreted	No compilation step
Object-Oriented	Supports OOP concepts
Cross-Platform	Runs on multiple OS
Large Library	Extensive standard library

Importance of Indentation:

- **Indentation:** Replaces curly braces
- **Consistent:** Usually 4 spaces per level
- **Mandatory:** Creates code structure

Mnemonic

“Simple-Interpreted-Object-Cross-Large”

Question Question 2(c OR) [07 marks]

Write a python program that calculates Student’s class/grade using following details.

Solution

Grading Table:

Table 10. Grading Scheme

Percentage	Grade
≥ 70	Distinction
60-69	First Class
50-59	Second Class
35-49	Pass Class
< 35	Fail

Code:

```

1 percentage = float(input("Enter percentage: "))
2
3 if percentage >= 70:

```

```

4     grade = "Distinction"
5 elif percentage >= 60:
6     grade = "First Class"
7 elif percentage >= 50:
8     grade = "Second Class"
9 elif percentage >= 35:
10    grade = "Pass Class"
11 else:
12     grade = "Fail"
13
14 print(f"Grade: {grade}")

```

Explanation:

- **Multiple conditions:** Nested if-elif structure
- **Grade assignment:** Based on percentage ranges
- **Float input:** Handles decimal percentages

Mnemonic

"Distinction-First-Second-Pass-Fail"

Question Question 3(a) [03 marks]

What is Selection Control Statement? List it out.

Solution**Selection Control Statements:**

Table 11. Selection Statements

Statement Type	Purpose
if	Single condition check
if-else	Two-way branching
if-elif-else	Multi-way branching
nested if	Conditions within conditions

Key Concepts:

- **Selection statements:** Control program flow based on conditions
- **Boolean evaluation:** Uses True/False logic
- **Branching:** Different paths of execution

Mnemonic

"If-IfElse-IfElif-Nested"

Question Question 3(b) [04 marks]

Write short note on nested loops.

Solution**Nested Loops:**

Table 12. Loop Structure

Loop Type	Structure
Outer Loop	Controls iterations
Inner Loop	Executes completely for each outer iteration
Total Iterations	Outer \times Inner

Key Points:

- **Nested structure:** Loop inside another loop
- **Complete execution:** Inner loop finishes before outer continues
- **Pattern creation:** Useful for 2D structures

Code Example:

```

1 for i in range(3):
2     for j in range(2):
3         print(f"i={i}, j={j}")

```

Mnemonic

“Outer-Inner-Complete-Pattern”

Question Question 3(c) [07 marks]

Write a user-defined function that displays all numbers, which are divisible by 4 from 1 to 100.

Solution**Code:**

```

1 def display_divisible_by_4():
2     print("Numbers divisible by 4 from 1 to 100:")
3     for num in range(1, 101):
4         if num % 4 == 0:
5             print(num, end=" ")
6     print()
7
8 # Function call
9 display_divisible_by_4()

```

Alternative with return:

```

1 def get_divisible_by_4():
2     return [num for num in range(1, 101) if num % 4 == 0]
3
4 result = get_divisible_by_4()
5 print(result)

```

Key Concepts:

- **Function definition:** def keyword usage
- **Range function:** 1 to 100 iteration
- **Modulus check:** num % 4 == 0 condition
- **List comprehension:** Alternative approach

Mnemonic

“Define-Range-Check-Display”

Question Question 3(a OR) [03 marks]

What is Repetition Control Statement? List it out.

Solution

Repetition Control Statements:

Table 13. Loops

Statement Type	Purpose
for loop	Known number of iterations
while loop	Condition-based repetition
nested loop	Loop within loop

Key Concepts:

- **Repetition statements:** Execute code blocks repeatedly
- **Iteration control:** Different methods of looping
- **Loop variables:** Track iteration progress

Mnemonic

“For-While-Nested”

Question Question 3(b OR) [04 marks]

Differentiate break and continue statements.

Solution

Difference:

Table 14. Break vs Continue

Aspect	break	continue
Purpose	Exit loop completely	Skip current iteration
Execution	Jumps out of loop	Jumps to next iteration
Usage	Terminate loop early	Skip specific conditions
Effect	Loop ends	Loop continues

Code Example:

```

1 # break example
2 for i in range(5):
3     if i == 3:
4         break
5     print(i) # Output: 0, 1, 2
6
7 # continue example
8 for i in range(5):
9     if i == 2:
10        continue
11    print(i) # Output: 0, 1, 3, 4

```

Mnemonic

“Break-Exit, Continue-Skip”

Question Question 3(c OR) [07 marks]

Write a user-defined function which displays all even numbers from 1 to 100.

Solution

Code:

```

1 def display_even_numbers():
2     print("Even numbers from 1 to 100:")
3     for num in range(2, 101, 2):
4         print(num, end=" ")
5     print()
6
7 # Alternative method
8 def display_even_alt():
9     even_nums = []
10    for num in range(1, 101):
11        if num % 2 == 0:
12            even_nums.append(num)
13    print(even_nums)
14
15 # Function call
16 display_even_numbers()
```

Explanation:

- Efficient range: `range(2, 101, 2)` for even numbers
- Modulus method: Alternative checking with `% 2 == 0`
- Function design: Reusable code block

Mnemonic

“Range-Step-Even-Display”

Question Question 4(a) [03 marks]

Define Function. List out various types of Functions available in Python.

Solution

Function: Reusable block of code that performs specific task.

Function Types:

Table 15. Types

Function Type	Description
Built-in	Pre-defined functions (<code>print</code> , <code>len</code>)
User-defined	Created by programmer
Lambda	Anonymous single-line functions
Recursive	Functions calling themselves

Benefits:

- Code reusability: Write once, use many times
- Modularity: Breaking complex problems into smaller parts
- Parameters: Input values to functions

Mnemonic

“Built-User-Lambda-Recursive”

Question Question 4(b) [04 marks]

Write short note on Scope of a variable.

Solution

Variable Scope:

Table 16. Scope Types

Scope Type	Description	Example
Local	Inside function only	Function variables
Global	Throughout program	Module-level variables
Built-in	Python keywords	print, len, type

Code Example:

```

1 x = 10 # Global variable
2
3 def my_function():
4     y = 20 # Local variable
5     print(x) # Access global
6     print(y) # Access local
7
8 my_function()
9 # print(y) # Error: y not accessible

```

Key Concepts:

- **Variable accessibility:** Where variables can be used
- **LEGB rule:** Local, Enclosing, Global, Built-in

Mnemonic

“Local-Global-Builtin”

Question Question 4(c) [07 marks]

Write Python code which asks user for Main string and Substring and checks membership of a Substring in the Main String.

Solution

Code:

```

1 def check_substring():
2     main_string = input("Enter main string: ")
3     substring = input("Enter substring: ")
4
5     if substring in main_string:
6         print(f'{substring} found in {main_string}')
7         print(f"Position: {main_string.find(substring)}")

```

```

8     else:
9         print(f'{substring} not found in {main_string}')
10
11 # Enhanced version with case handling
12 def check_substring_enhanced():
13     main_string = input("Enter main string: ")
14     substring = input("Enter substring: ")
15
16     if substring.lower() in main_string.lower():
17         print("Substring found (case-insensitive)")
18     else:
19         print("Substring not found")
20
21 check_substring()

```

Explanation:

- User interaction: `input()` for string collection
- Membership testing: `in` operator usage
- Case sensitivity: Optional case handling

Mnemonic

“Input-Check-Report-Position”

Question Question 4(a OR) [03 marks]

What is Local variable and Global variable?

Solution**Comparison:**

Table 17. Local vs Global

Variable Type	Scope	Lifetime	Access
Local	Function only	Function execution	Limited
Global	Entire program	Program execution	Widespread

Example:

```

1 global_var = 100 # Global
2
3 def function():
4     local_var = 50 # Local
5     print(global_var) # Accessible
6     print(local_var) # Accessible
7
8     print(global_var) # Accessible
9     # print(local_var) # Error

```

Mnemonic

“Local-Limited, Global-Everywhere”

Question Question 4(b OR) [04 marks]

Explain any four built-in functions of Python.

Solution

Built-in Functions:

Table 18. Functions

Function	Purpose	Example
len()	Returns length	len("hello") → 5
type()	Returns data type	type(10) → <class 'int'>
input()	Gets user input	name = input("Name: ")
print()	Displays output	print("Hello")

Additional Examples:

```

1 # len() function
2 print(len([1, 2, 3, 4])) # Output: 4
3
4 # type() function
5 print(type(3.14)) # Output: <class 'float'>
6
7 # input() function
8 age = input("Enter age: ")
9
10 # print() function
11 print("Your age is:", age)

```

Mnemonic

“Length-Type-Input-Print”

Question Question 4(c OR) [07 marks]

Write Python code which locates a substring in a given string.

Solution

Code:

```

1 def locate_substring():
2     main_string = input("Enter main string: ")
3     substring = input("Enter substring to find: ")
4
5     # Method 1: Using find()
6     position = main_string.find(substring)
7     if position != -1:
8         print(f"Found at index: {position}")
9     else:
10        print("Substring not found")
11
12     # Method 2: Using index() with exception handling
13     try:
14         position = main_string.index(substring)
15         print(f"Located at index: {position}")

```

```

16     except ValueError:
17         print("Substring not found")
18
19 # Method 3: Find all occurrences
20 positions = []
21 start = 0
22 while True:
23     pos = main_string.find(substring, start)
24     if pos == -1:
25         break
26     positions.append(pos)
27     start = pos + 1
28
29 if positions:
30     print(f"All positions: {positions}")
31
32 locate_substring()

```

Key Methods:

- **find()** method: Returns index or -1
- **index()** method: Returns index or raises exception
- **Multiple occurrences:** Loop to find all positions

Mnemonic

“Find-Index-Exception-Multiple”

Question Question 5(a) [03 marks]

Define String. List out various string operations.

Solution

String: Sequence of characters enclosed in quotes.

Operations:

Table 19. String Operations

Operation	Method	Example
Concatenation	+	"Hello" + "World"
Repetition	*	"Hi" * 3
Slicing	[start:end]	"Hello"[1:4]
Length	len()	len("Hello")
Case	upper(), lower()	"hello".upper()

Characteristics:

- **Immutable:** Strings cannot be changed after creation
- **Indexing:** Access individual characters
- **Methods:** Built-in functions for manipulation

Mnemonic

“Concat-Repeat-Slice-Length-Case”

Question Question 5(b) [04 marks]

How can we identify whether an element is a member of a list or not? Explain with a suitable example.

Solution

Methods:

Table 20. Membership Check

Method	Operator	Returns
in	element in list	True/False
not in	element not in list	True/False
count()	list.count(element)	Number of occurrences

Example:

```

1 fruits = ["apple", "banana", "orange", "mango"]
2
3 # Using 'in' operator
4 if "apple" in fruits:
5     print("Apple is available")
6
7 # Using 'not in' operator
8 if "grapes" not in fruits:
9     print("Grapes not available")
10
11 # Using count() method
12 count = fruits.count("apple")
13 if count > 0:
14     print(f"Apple found {count} times")

```

Mnemonic

“In-NotIn-Count for membership”

Question Question 5(c) [07 marks]

Write Python code that replaces a substring with another substring of a given string. Consider the given string as 'Welcome to GTU' and replace the substring 'GTU' with 'Gujarat Technological University'.

Solution

Code:

```

1 def replace_substring():
2     # Given string
3     original = "Welcome to GTU"
4     old_substring = "GTU"
5     new_substring = "Gujarat Technological University"
6
7     # Method 1: Using replace()
8     result1 = original.replace(old_substring, new_substring)
9     print(f"Original: {original}")
10    print(f"Modified: {result1}")
11

```

```

12 # Method 2: Manual replacement
13 if old_substring in original:
14     index = original.find(old_substring)
15     result2 = original[:index] + new_substring + original[index + len(old_substring):]
16     print(f"Manual method: {result2}")
17
18 # Method 3: Replace all occurrences
19 test_string = "GTU offers GTU degree from GTU"
20 result3 = test_string.replace("GTU", "Gujarat Technological University")
21 print(f"Multiple replacements: {result3}")
22
23 replace_substring()

```

Output:

Original: Welcome to GTU
 Modified: Welcome to Gujarat Technological University

Key Points:

- **replace()** method: Built-in string function
- **Slicing method:** Manual string manipulation
- **All occurrences:** Replaces every instance

Mnemonic

“Find-Replace-Slice-All”

Question Question 5(a OR) [03 marks]

Define List. List out various list operations.

Solution

List: Ordered collection of items that can be modified.

Operations:

Table 21. List Operations

Operation	Method	Example
Add	append(), insert()	list.append(item)
Remove	remove(), pop()	list.remove(item)
Access	[index]	list[0]
Slice	[start:end]	list[1:3]
Sort	sort()	list.sort()

Features:

- **Mutable:** Lists can be changed after creation
- **Indexed:** Elements accessed by position
- **Dynamic:** Size can grow or shrink

Mnemonic

“Add-Remove-Access-Slice-Sort”

Question Question 5(b OR) [04 marks]

Write short note on String Slicing. Explain with suitable example.

Solution

String Slicing: Extracting parts of string using `[start:end:step]`.

Syntax:

Table 22. Slicing Syntax

Syntax	Description	Example
<code>[start:]</code>	From start to end	<code>"Hello"[1:]</code> → "ello"
<code>[:end]</code>	From beginning to end	<code>"Hello":[3]</code> → "Hel"
<code>[start:end]</code>	Specific range	<code>"Hello"[1:4]</code> → "ell"
<code>[::-1]</code>	Reverse string	<code>"Hello"[::-1]</code> → "olleH"

Example:

```

1 text = "Python Programming"
2
3 print(text[0:6])      # "Python"
4 print(text[7:])        # "Programming"
5 print(text[:6])        # "Python"
6 print(text[::-2])      # "Pto rgamm"
7 print(text[::-1])      # "gnimmargorP nohtyP"

```

Mnemonic

“Start-End-Step-Reverse”