

# Subject Name Solutions

1323203 – Summer 2023

Semester 1 Study Material

*Detailed Solutions and Explanations*

## Question 1(a) [3 marks]

Define algorithm. What are the advantages of Algorithm?

### Solution

An algorithm is a step-by-step procedure or a set of rules to solve a specific problem in a finite sequence of steps.

#### Advantages of Algorithm:

- **Clarity:** Provides clear, unambiguous instructions
- **Efficiency:** Helps in optimizing time and resources
- **Reusability:** Can be used repeatedly for similar problems
- **Verification:** Easy to test and debug before implementation
- **Communication:** Acts as a blueprint to communicate the solution

### Mnemonic

“CERVC” (Clarity, Efficiency, Reusability, Verification, Communication)

## Question 1(b) [4 marks]

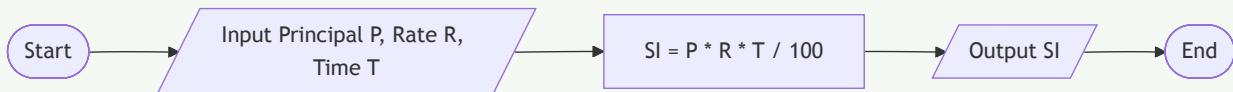
What are the rules for problem solving using flowchart? Design a flowchart to find simple interest.

### Solution

Rules for problem solving using flowchart:

- **Proper symbols:** Use standard symbols for different operations
- **Direction flow:** Always maintain clear top-to-bottom, left-to-right flow
- **Single entry/exit:** Have a clear start and end point
- **Clarity:** Keep steps clear and concise
- **Consistency:** Maintain consistent level of detail

#### Flowchart for Simple Interest Calculation:



### Mnemonic

“PDRSC” (Proper symbols, Direction flow, Required entry/exit, Simplicity, Consistency)

## Question 1(c) [7 marks]

List out assignment operator in python and build a python code to demonstrate an operation of any three assignment operators.

### Solution

Python assignment operators:

Operator	Example	Equivalent To
=	x = 5	x = 5
+=	x += 5	x = x + 5
-=	x -= 5	x = x - 5
*=	x *= 5	x = x * 5
/=	x /= 5	x = x / 5
%=	x %= 5	x = x % 5
//=	x //= 5	x = x // 5
**=	x **= 5	x = x ** 5
&=	x &= 5	x = x & 5
=	x  = 5	x = x   5
^=	x ^= 5	x = x ^ 5
»=	x »= 5	x = x » 5
«=	x «= 5	x = x « 5

#### Code demonstrating assignment operators:

```

1 # Demonstrating Assignment Operators
2 num = 10
3 print("Initial value:", num)
4
5 # Using += operator
6 num += 5
7 print("After += 5:", num) # Output: 15
8
9 # Using -= operator
0 num -= 3
1 print("After -= 3:", num) # Output: 12
2
3 # Using *= operator
4 num *= 2
5 print("After *= 2:", num) # Output: 24

```

#### Mnemonic

“VALUE” (Variable Assignment is Like Updating Existing values)

#### Question 1(c) OR [7 marks]

List out data types in python and Develop a Program to identify any three data types in python.

#### Solution

Python data types:

Data Type	Description	Example
int	Integer (whole numbers)	42
float	Floating point (decimal)	3.14
str	String (text)	“Hello”
bool	Boolean (True/False)	True
list	Ordered, mutable collection	[1, 2, 3]
tuple	Ordered, immutable collection	(1, 2, 3)
set	Unordered collection of unique items	{1, 2, 3}
dict	Key-value pairs	{"name": "John"}
complex	Complex numbers	2+3j
NoneType	Represents None	None

### Code to identify three data types:

```
1 # Program to identify data types
2 def identify_data_type(value):
3     data_type = type(value).__name__
4     print(f"Value: {value}")
5     print(f"Data Type: {data_type}")
6     print("-" * 20)
7
8 # Testing with 3 different data types
9 identify_data_type(42)          # Integer
10 identify_data_type(3.14)         # Float
11 identify_data_type("Hello World") # String
12
13 # Output:
14 # Value: 42
15 # Data Type: int
16 # -----
17 # Value: 3.14
18 # Data Type: float
19 # -----
20 # Value: Hello World
21 # Data Type: str
22 # -----
```

### Mnemonic

“TYPE-ID” (Tell Your Python Elements - Identify Data)

### Question 2(a) [3 marks]

Define pseudocode. Write pseudocode to find smallest of two number.

#### Solution

Pseudocode is a high-level description of an algorithm that uses structural conventions of a programming language but is designed for human reading rather than machine reading.

#### Pseudocode to find smallest of two numbers:

```
1 BEGIN
2     INPUT first_number, second_number
3     IF first_number < second_number THEN
4         smallest = first_number
5     ELSE
6         smallest = second_number
7     END IF
8     OUTPUT smallest
9 END
```

### Mnemonic

“RISE” (Read Input, Select smallest, Echo result)

### Question 2(b) [4 marks]

Develop a python code to read three numbers from the user and find the average of the numbers.

#### Solution

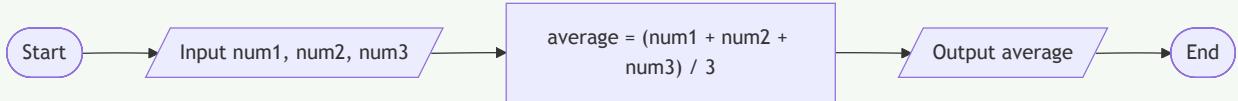
```
1 # Program to calculate average of three numbers
2 # Input three numbers from user
3 num1 = float(input("Enter first number: "))
```

```

4 num2 = float(input("Enter second number: "))
5 num3 = float(input("Enter third number: "))
6
7 # Calculate the average
8 average = (num1 + num2 + num3) / 3
9
10 # Display the result
11 print(f"The average of {num1}, {num2}, and {num3} is: {average}")

```

Diagram:



Mnemonic

“I-ADD-D” (Input three, ADD them up, Divide by 3)

### Question 2(c) [7 marks]

Write a python code to show whether the entered number is prime or not.

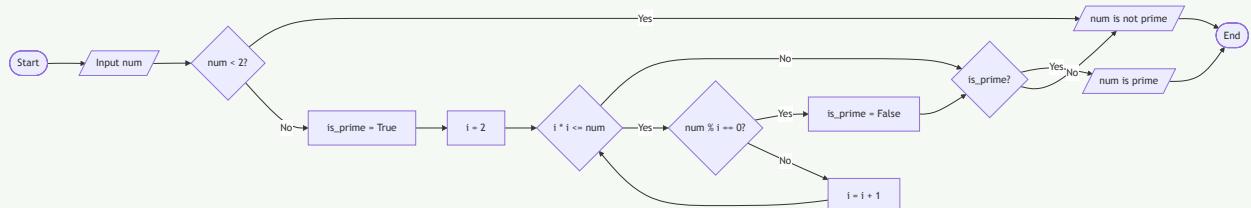
Solution

```

1 # Program to check if a number is prime
2 # Input number from user
3 num = int(input("Enter a number: "))
4
5 # Check if number is less than 2
6 if num < 2:
7     print(f"{num} is not a prime number")
8 else:
9     # Initialize is_prime as True
10    is_prime = True
11
12    # Check from 2 to sqrt(num)
13    for i in range(2, int(num**0.5) + 1):
14        if num %
15
16            i == 0:
17
18                is_prime = False
19                break
20
21    # Display result
22    if is_prime:
23        print(f"{num} is a prime number")
24    else:
25        print(f"{num} is not a prime number")

```

Diagram:



### Mnemonic

“PRIME” (Positive number, Range check from 2 to , If divisible it’s Multiple, Else it’s prime)

### Question 2(a) OR [3 marks]

Write down a difference between Flow chart and Algorithm.

#### Solution

##### Flow Chart

**Visual representation** using standard symbols and shapes  
**Easier to understand** due to graphical nature  
Shows **logical flow** and relationships clearly  
**Time-consuming to create** but easier to follow  
More difficult to modify or update

##### Algorithm

**Textual description** using structured language  
Requires knowledge of syntax and terminology  
Provides **detailed steps** in sequential order  
**Quicker to draft** but may be harder to interpret  
Easier to modify or update

### Mnemonic

“VITAL” (Visual vs Textual, Interpretation ease, Time to create, Alteration flexibility, Logical representation)

### Question 2(b) OR [4 marks]

What is the output of the following code:

```
1 x=10
2 y=2
3 print (x*y)
4 print (x ** y)
5 print (x//y)
6 print (x % y)
```

#### Solution

Operation	Explanation	Output
x*y	Multiplication: $10 \times 2$	20
x**y	Exponentiation: $10^2$	100
x//y	Integer division: $10 \div 2$	5
x%y	Modulus (remainder): $10 \div 2$	0

### Mnemonic

“MEMO” (Multiply, Exponent, Modulo, Operations)

### Question 2(c) OR [7 marks]

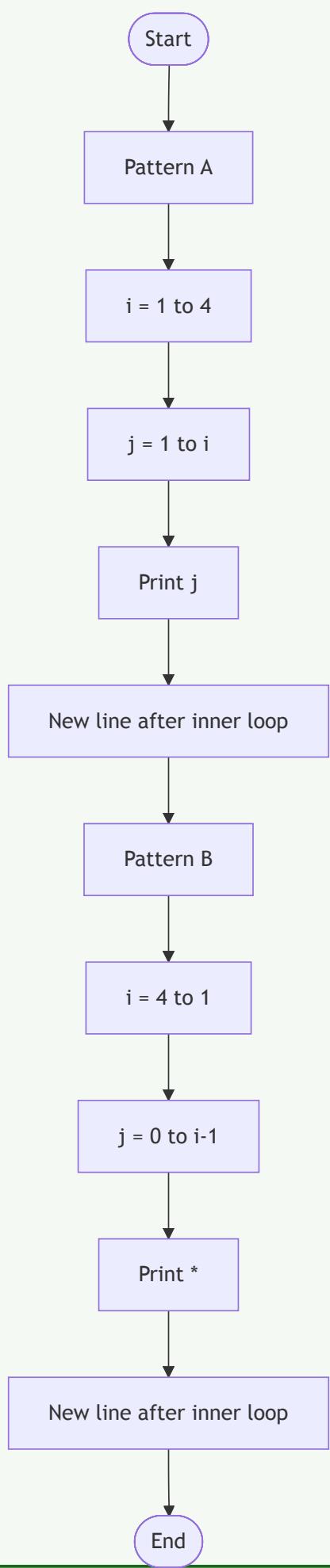
Write a python code to display the following patterns:

```
1 A)
2   1
3   1 2
4   1 2 3
5   1 2 3 4
       B)
          * * * *
          * * *
          * *
          *
```

## Solution

```
1 # Pattern A: Number pattern
2 print("Pattern A:")
3 for i in range(1, 5):
4     for j in range(1, i + 1):
5         print(j, end=" ")
6     print()
7
8 # Pattern B: Star pattern
9 print("\nPattern B:")
10 for i in range(4, 0, -1):
11     for j in range(i):
12         print("*", end=" ")
13     print()
```

Diagram:



## Mnemonic

“LOOP-NED” (Loop Outer, Order Pattern, Nested loops, End with newline, Display)

### Question 3(a) [3 marks]

With the necessary examples describe the use of break statement.

#### Solution

Break statement is used to exit or terminate a loop prematurely when a specific condition is met.

**Example:**

```
1 # Finding the first odd number in a list
2 numbers = [2, 4, 6, 7, 8, 10]
3 for num in numbers:
4     if num % 2 != 0:
5         print(f"Found odd number: {num}")
6         break
7     print(f"Checking {num}")
```

**Output:**

```
1 Checking 2
2 Checking 4
3 Checking 6
4 Found odd number: 7
```

## Mnemonic

“EXIT” (EXecute until condition, Immediately Terminate)

### Question 3(b) [4 marks]

Explain if...else statement with suitable example.

#### Solution

The if...else statement is a conditional statement that executes different blocks of code based on whether a specified condition evaluates to True or False.

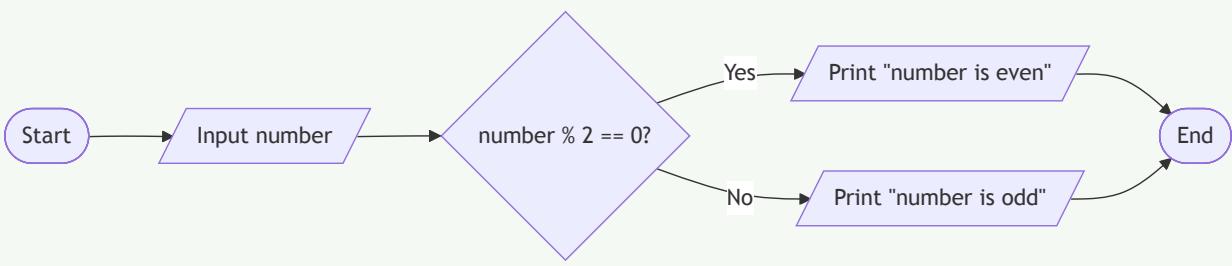
**Syntax:**

```
1 if condition:
2     # Code to be executed if condition is True
3 else:
4     # Code to be executed if condition is False
```

**Example:**

```
1 # Check if a number is even or odd
2 number = int(input("Enter a number: "))
3
4 if number % 2 == 0:
5     print(f"{number} is an even number")
6 else:
7     print(f"{number} is an odd number")
```

**Diagram:**



### Mnemonic

“CITE” (Check condition, If True Execute this, Else execute that)

### Question 3(c) [7 marks]

Create a User-defined function to print the Fibonacci series of 0 to N numbers where N is an integer number and passed as an argument.

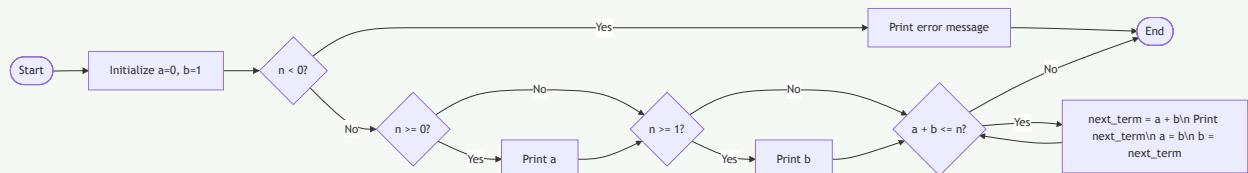
#### Solution

```

1 # Function to print Fibonacci series
2 def print_fibonacci(n):
3     """
4         Print Fibonacci series from 0 to n
5         Args:
6             n: Upper limit (inclusive)
7     """
8
9     # Initialize first two terms
10    a, b = 0, 1
11
12    # Check if n is valid
13    if n < 0:
14        print("Please enter a positive number")
15        return
16
17    # Print Fibonacci series
18    print("Fibonacci series up to", n, ":")
19
20    if n >= 0:
21        print(a, end=" ") # Print first term
22
23    if n >= 1:
24        print(b, end=" ") # Print second term
25
26    # Generate and print the rest of the series
27    while a + b <= n:
28        next_term = a + b
29        print(next_term, end=" ")
30        a, b = b, next_term
31
32 # Test the function
33 print_fibonacci(55)

```

#### Diagram:



## Mnemonic

“FIBER” (First terms set, Initialize variables, Build next term, Echo results, Repeat until limit)

### Question 3(a) OR [3 marks]

With the necessary examples describe the use of continue statement.

#### Solution

Continue statement is used to skip the current iteration of a loop and continue with the next iteration.

**Example:**

```
1 # Print only odd numbers from 1 to 10
2 for i in range(1, 11):
3     if i % 2 == 0:
4         continue # Skip even numbers
5     print(i)
```

**Output:**

```
1 1
2 3
3 5
4 7
5 9
```

## Mnemonic

“SKIP” (Skip current iteration, Keep looping, Ignore remaining statements, Proceed to next iteration)

### Question 3(b) OR [4 marks]

Explain For loop statement with example.

#### Solution

For loop is used to iterate over a sequence (like list, tuple, string) or other iterable objects and execute a block of code for each item in the sequence.

**Syntax:**

```
1 for variable in sequence:
2     # Code to be executed for each item
```

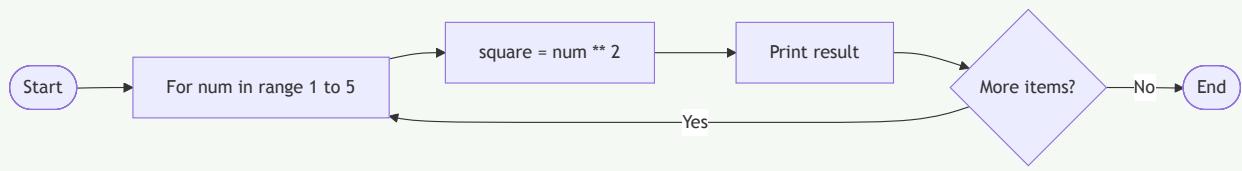
**Example:**

```
1 # Print squares of numbers from 1 to 5
2 for num in range(1, 6):
3     square = num ** 2
4     print(f"The square of {num} is {square}")
```

**Output:**

```
1 The square of 1 is 1
2 The square of 2 is 4
3 The square of 3 is 9
4 The square of 4 is 16
5 The square of 5 is 25
```

**Diagram:**



### Mnemonic

“FIRE” (For each Item, Run commands, Execute until end)

### Question 3(c) OR [7 marks]

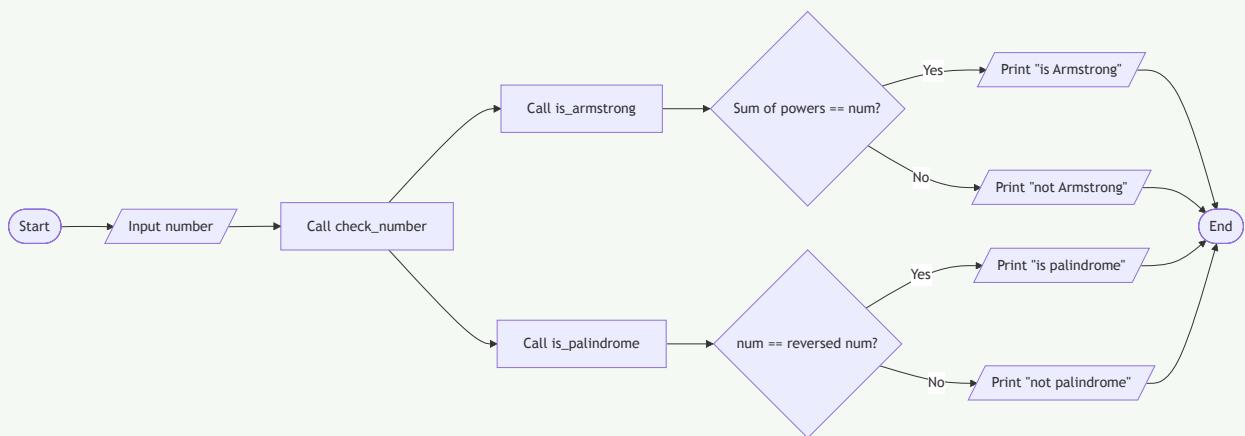
Write a python code that determines whether a given number is an ‘Armstrong number’ or a palindrome using a user-defined function.

#### Solution

```

1 # Function to check if a number is Armstrong number
2 def is_armstrong(num):
3     # Convert to string to count digits
4     num_str = str(num)
5     n = len(num_str)
6
7     # Calculate sum of each digit raised to power of total digits
8     sum_of_powers = sum(int(digit) ** n for digit in num_str)
9
10    # Check if sum equals the original number
11    return sum_of_powers == num
12
13 # Function to check if a number is a palindrome
14 def is_palindrome(num):
15     # Convert to string
16     num_str = str(num)
17
18     # Check if string equals its reverse
19     return num_str == num_str[::-1]
20
21 # Main function to check both conditions
22 def check_number(num):
23     if is_armstrong(num):
24         print(f"{num} is an Armstrong number")
25     else:
26         print(f"{num} is not an Armstrong number")
27
28     if is_palindrome(num):
29         print(f"{num} is a palindrome")
30     else:
31         print(f"{num} is not a palindrome")
32
33 # Test the function
34 number = int(input("Enter a number: "))
35 check_number(number)
  
```

Diagram:



### Mnemonic

“APC” (Armstrong check: Power sum of digits, Palindrome check: Compare with reverse)

### Question 4(a) [3 marks]

Develop a python code to identify whether the scanned number is even or odd and print an appropriate message.

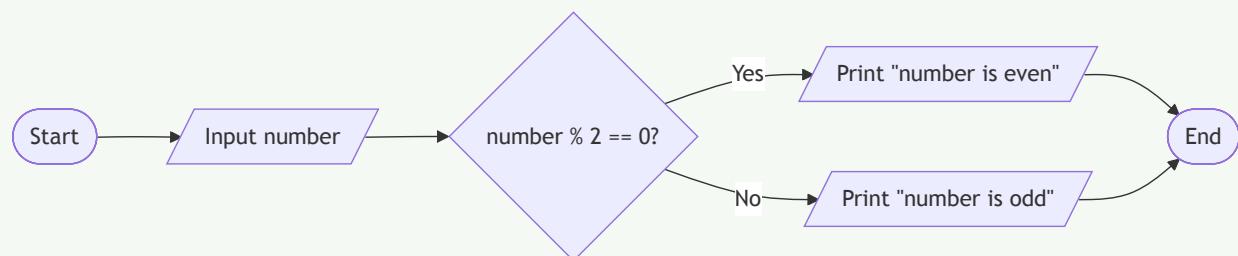
#### Solution

```

1 # Program to check if a number is even or odd
2 # Input number from user
3 number = int(input("Enter a number: "))
4
5 # Check if number is even or odd
6 if number % 2 == 0:
7     print(f"{number} is an even number")
8 else:
9     print(f"{number} is an odd number")

```

#### Diagram:



### Mnemonic

“MODE” (Modulo Operation Determines Even-odd)

### Question 4(b) [4 marks]

Define function. Explain user define function using suitable example.

#### Solution

A function is a block of organized, reusable code that performs a specific task. User-defined functions are functions created by the programmer to perform custom operations.

### Components of a User-defined Function:

- **def keyword:** Marks the start of function definition
- **Function name:** Identifier for the function
- **Parameters:** Input values (optional)
- **Docstring:** Description of the function (optional)
- **Function body:** Code to be executed
- **Return statement:** Output value (optional)

### Example:

```
1 # User-defined function to calculate area of rectangle
2 def calculate_area(length, width):
3     """
4         Calculate area of rectangle
5         Args:
6             length: Length of rectangle
7             width: Width of rectangle
8         Returns:
9             Area of rectangle
10        """
11    area = length * width
12    return area
13
14 # Call the function
15 result = calculate_area(5, 3)
16 print(f"Area of rectangle: {result}")
```

### Mnemonic

“DRAPE” (Define function, Receive parameters, Acquire result, Process data, End with return)

## Question 4(c) [7 marks]

List out various String operations and explain any three using example.

### Solution

String operations in Python:

Operation	Description
Concatenation	Joining strings together using +
Repetition	Repeating a string using *
Indexing	Accessing characters by position
Slicing	Extracting a portion of a string
Methods (len, upper, lower, etc.)	Built-in functions for string manipulation
Membership Testing	Check if a substring exists in a string
Formatting	Create formatted strings
Escape Sequences	Special characters preceded by \

### Three String Operations with Examples:

#### 1. String Concatenation:

```
1 first_name = "John"
2 last_name = "Doe"
3 full_name = first_name + " " + last_name
4 print(full_name) # Output: John Doe
```

#### 1. String Slicing:

```
1 message = "Python Programming"
2 print(message[0:6]) # Output: Python
3 print(message[7:]) # Output: Programming
4 print(message[-11:]) # Output: Programming
```

#### 1. String Methods:

```
1 text = "python programming"
2 print(text.upper()) # Output: PYTHON PROGRAMMING
3 print(text.capitalize()) # Output: Python programming
4 print(text.replace("python", "Java")) # Output: Java programming
```

### Mnemonic

“CSM” (Concatenate strings, Slice portions, Manipulate with methods)

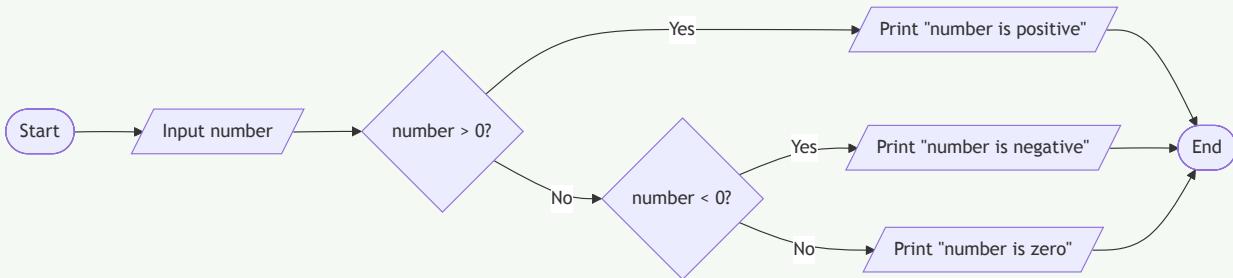
### Question 4(a) OR [3 marks]

Create a python code to check positive or negative number.

#### Solution

```
1 # Program to check if a number is positive or negative
2 # Input number from user
3 number = float(input("Enter a number: "))
4
5 # Check if number is positive, negative, or zero
6 if number > 0:
7     print(f"{number} is a positive number")
8 elif number < 0:
9     print(f"{number} is a negative number")
10 else:
11     print("The number is zero")
```

#### Diagram:



### Mnemonic

“SIGN” (See If Greater than 0, Negative otherwise)

### Question 4(b) OR [4 marks]

Explain local and global variables using suitable examples.

## Solution

In Python, variables can have different scopes:

Variable Type	Description
Local Variable	Defined within a function and accessible only inside that function
Global Variable	Defined outside functions and accessible throughout the program

### Example:

```
1 # Global variable
2 count = 0 # This is a global variable
3
4 def update_count():
5     # Local variable
6     local_var = 5 # This is a local variable
7
8     # Accessing global variable inside function
9     global count
10    count += 1
11
12    print(f"Local variable: {local_var}")
13    print(f"Global variable (inside function): {count}")
14
15 # Call the function
16 update_count()
17
18 # Accessing variables outside function
19 print(f"Global variable (outside function): {count}")
20
21 # This would cause an error if uncommented
22 # print(local_var) # NameError: name 'local_var' is not defined
```

### Output:

```
1 Local variable: 5
2 Global variable (inside function): 1
3 Global variable (outside function): 1
```

## Mnemonic

“SCOPE” (Some variables Confined to function Only, Program-wide Exposure for others)

## Question 4(c) OR [7 marks]

List out various List operations and explain any three using example.

## Solution

List operations in Python:

Operation	Description
Creating Lists	Using square brackets []
Indexing	Accessing elements by position
Slicing	Extracting portions of a list
Append	Adding elements to the end
Insert	Adding elements at specific positions
Remove	Removing specific elements
Pop	Removing and returning elements
Sort	Ordering list elements

Reverse	Reversing list order
Extend	Combining lists
List Comprehensions	Creating lists using expressions

### Three List Operations with Examples:

#### 1. List Indexing and Slicing:

```

1 fruits = ["apple", "banana", "cherry", "orange", "kiwi"]
2 print(fruits[1])      # Output: banana
3 print(fruits[-1])     # Output: kiwi
4 print(fruits[1:4])    # Output: ['banana', 'cherry', 'orange']

```

#### 1. List Methods (append, insert, remove):

```

1 numbers = [1, 2, 3]
2 numbers.append(4)      # Add 4 to the end
3 print(numbers)         # Output: [1, 2, 3, 4]
4
5 numbers.insert(0, 0)    # Insert 0 at position 0
6 print(numbers)         # Output: [0, 1, 2, 3, 4]
7
8 numbers.remove(2)      # Remove element with value 2
9 print(numbers)         # Output: [0, 1, 3, 4]

```

#### 1. List Comprehensions:

```

# Create a list of squares
squares = [x**2 for x in range(1, 6)]
print(squares) # Output: [1, 4, 9, 16, 25]

# Filter even numbers
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
evens = [x for x in numbers if x % 2 == 0]
print(evens) # Output: [2, 4, 6, 8, 10]

```

### Mnemonic

“AIM” (Access with index, Insert/modify elements, Make using comprehensions)

### Question 5(a) [3 marks]

Write python code to swap given two elements in a list.

#### Solution

```

1 # Program to swap two elements in a list
2 def swap_elements(my_list, pos1, pos2):
3     """
4         Swap elements at positions pos1 and pos2 in the list
5     """
6     # Check if positions are valid
7     if 0 <= pos1 < len(my_list) and 0 <= pos2 < len(my_list):
8         # Swap elements
9         my_list[pos1], my_list[pos2] = my_list[pos2], my_list[pos1]
10        return True
11    else:
12        return False
13
14 # Example usage
15 numbers = [10, 20, 30, 40, 50]
16 print("Original list:", numbers)
17
18 # Swap elements at positions 1 and 3
19 if swap_elements(numbers, 1, 3):

```

```

10     print("After swapping:", numbers)
11 else:
12     print("Invalid positions")

```

### Output:

```

1 Original list: [10, 20, 30, 40, 50]
2 After swapping: [10, 40, 30, 20, 50]

```

### Mnemonic

“SWAP” (Select positions, Watch boundaries, Assign simultaneously, Print result)

## Question 5(b) [4 marks]

Explain math module and random module in python using example.

### Solution

Math and random modules provide functions for mathematical operations and random number generation.

#### Math Module:

```

1 import math
2
3 # Constants
4 print(math.pi)          # Output: 3.141592653589793
5 print(math.e)            # Output: 2.718281828459045
6
7 # Mathematical functions
8 print(math.sqrt(16))    # Output: 4.0
9 print(math.ceil(4.2))   # Output: 5
0 print(math.floor(4.8))  # Output: 4
1 print(math.pow(2, 3))   # Output: 8.0

```

#### Random Module:

```

1 import random
2
3 # Random float between 0 and 1
4 print(random.random())      # Output: 0.123... (random)
5
6 # Random integer within range
7 print(random.randint(1, 10)) # Output: 7 (random between 1 and 10)
8
9 # Random choice from a sequence
10 colors = ["red", "green", "blue"]
11 print(random.choice(colors)) # Output: "green" (random)
12
13 # Shuffle a list
14 numbers = [1, 2, 3, 4, 5]
15 random.shuffle(numbers)
16 print(numbers)             # Output: [3, 1, 5, 2, 4] (random)

```

### Mnemonic

“MR-CS” (Math for Calculations, Random for Choice and Shuffling)

## Question 5(c) [7 marks]

Write a python code to demonstrate tuples functions and operations.

## Solution

```
1 # Demonstrating Tuple Functions and Operations
2
3 # Creating tuples
4 empty_tuple = ()
5 single_item_tuple = (1,) # Note the comma
6 mixed_tuple = (1, "Hello", 3.14, True)
7 nested_tuple = (1, 2, (3, 4))
8
9 # Accessing tuple elements
10 print("Accessing elements:")
11 print(mixed_tuple[0])      # Output: 1
12 print(mixed_tuple[-1])     # Output: True
13 print(nested_tuple[2][0])   # Output: 3
14
15 # Tuple slicing
16 print("\nTuple slicing:")
17 print(mixed_tuple[1:3])    # Output: ("Hello", 3.14)
18
19 # Tuple concatenation
20 tuple1 = (1, 2, 3)
21 tuple2 = (4, 5, 6)
22 tuple3 = tuple1 + tuple2
23 print("\nConcatenated tuple:", tuple3) # Output: (1, 2, 3, 4, 5, 6)
24
25 # Tuple repetition
26 repeated_tuple = tuple1 * 3
27 print("\nRepeated tuple:", repeated_tuple) # Output: (1, 2, 3, 1, 2, 3, 1, 2, 3)
28
29 # Tuple methods
30 numbers = (1, 2, 3, 2, 4, 2)
31 print("\nCount of 2:", numbers.count(2)) # Output: 3
32 print("Index of 3:", numbers.index(3))   # Output: 2
33
34 # Tuple unpacking
35 print("\nTuple unpacking:")
36 x, y, z = (10, 20, 30)
37 print(f"x={x},"
38
39 y={y},
40
41 z={z}") # Output:
42
43 x=10,
44
45 y=20,
46
47 z=30
48
49
50 # Check if an element exists in a tuple
51 print("\nMembership testing:")
52 print(3 in numbers)      # Output: True
53 print(5 in numbers)      # Output: False
54
55 # Converting list to tuple and vice versa
56 my_list = [1, 2, 3]
57 my_tuple = tuple(my_list)
58 print("\nList to tuple:", my_tuple)
59
60 back_to_list = list(my_tuple)
61 print("Tuple to list:", back_to_list)
```

### Diagram:



## Mnemonic

“CASC-RUMTC” (Create, Access, Slice, Concatenate, Repeat, Use methods, Membership test, Tuple conversion)

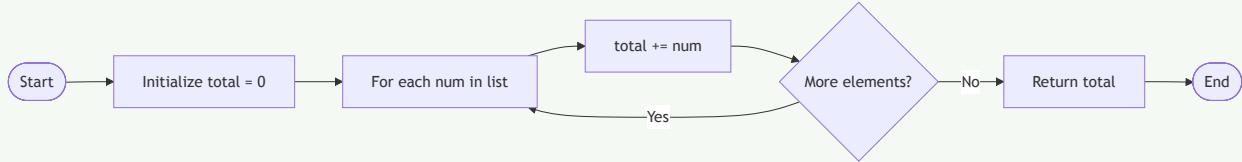
### Question 5(a) OR [3 marks]

Write a python code to find the sum of elements in a list.

#### Solution

```
1 # Program to find the sum of elements in a list
2 def sum_of_elements(numbers):
3     """
4         Calculate the sum of all elements in a list
5     """
6     total = 0
7     for num in numbers:
8         total += num
9     return total
10
11 # Example usage
12 my_list = [10, 20, 30, 40, 50]
13 print("List:", my_list)
14 print("Sum of elements:", sum_of_elements(my_list)) # Output: 150
15
16 # Alternative using built-in sum() function
17 print("Sum using built-in function:", sum(my_list)) # Output: 150
```

#### Diagram:



## Mnemonic

“SITE” (Sum Initialized To zero, Elements added one by one)

### Question 5(b) OR [4 marks]

Explain the usage of following built in functions: 1) Print() 2) Min() 3) Sum() 4) Input()

#### Solution

Function	Purpose	Example	Output
<code>print()</code>	Displays output to the console	<code>print("Hello World")</code>	Hello World
<code>min()</code>	Returns smallest item in an iterable	<code>min([5, 3, 8, 1])</code>	1
<code>sum()</code>	Returns sum of all items in an iterable	<code>sum([1, 2, 3, 4])</code>	10
<code>input()</code>	Reads input from the user	<code>name = input("Enter name: ")</code>	(waits for user input)

### Example Code:

```
1 # print() function
2 print("Hello, Python!") # Basic output
3 print("a", "b", "c", sep="-") # Output with separator: a-b-c
4 print("No newline", end=" ") # Custom end character
5 print("on same line") # Output: No newline on same line
6
7 # min() function
8 numbers = [15, 8, 23, 4, 42]
9 print("Minimum value:", min(numbers)) # Output: 4
10 print("Minimum of 5, 2, 9:", min(5, 2, 9)) # Output: 2
11 chars = "wxyz"
12 print("Minimum character:", min(chars)) # Output: w
13
14 # sum() function
15 print("Sum of numbers:", sum(numbers)) # Output: 92
16 print("Sum with start value:", sum(numbers, 10)) # Output: 102
17
18 # input() function
19 user_input = input("Enter something: ") # Prompts user for input
20 print("You entered:", user_input) # Displays user's input
```

### Mnemonic

“PMSI” (Print to display, Min for smallest, Sum for total, Input for reading)

### Question 5(c) OR [7 marks]

Write a python code to demonstrate the set functions and operations.

### Solution

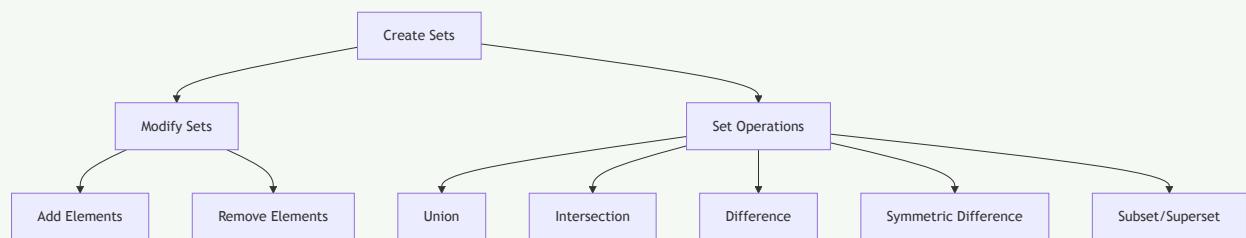
```
1 # Demonstrating Set Functions and Operations
2
3 # Creating sets
4 empty_set = set() # Empty set
5 numbers = {1, 2, 3, 4, 5}
6 duplicates = {1, 2, 2, 3, 4, 4, 5} # Duplicates removed automatically
7 print("Original set:", numbers)
8 print("Set with duplicates:", duplicates) # Output: {1, 2, 3, 4, 5}
9
10 # Adding elements
11 numbers.add(6)
12 print("\nAfter adding 6:", numbers) # Output: {1, 2, 3, 4, 5, 6}
13
14 # Updating with multiple elements
15 numbers.update([7, 8, 9])
16 print("After updating:", numbers) # Output: {1, 2, 3, 4, 5, 6, 7, 8, 9}
17
18 # Removing elements
19 numbers.remove(5) # Raises error if element not found
20 print("\nAfter removing 5:", numbers)
21
22 numbers.discard(10) # No error if element not found
23 print("After discarding 10:", numbers) # No change
24
25 popped = numbers.pop() # Removes and returns arbitrary element
26 print("Popped element:", popped)
27 print("After pop:", numbers)
28
29 # Set operations
30 set1 = {1, 2, 3, 4, 5}
31 set2 = {4, 5, 6, 7, 8}
```

```

42
43 # Union
44 union_set = set1 | set2 # or set1.union(set2)
45 print("\nUnion:", union_set) # Output: {1, 2, 3, 4, 5, 6, 7, 8}
46
47 # Intersection
48 intersection_set = set1 & set2 # or set1.intersection(set2)
49 print("Intersection:", intersection_set) # Output: {4, 5}
50
51 # Difference
52 difference_set = set1 - set2 # or set1.difference(set2)
53 print("Difference (set1 - set2):", difference_set) # Output: {1, 2, 3}
54
55 # Symmetric Difference
56 symmetric_diff = set1 ^ set2 # or set1.symmetric_difference(set2)
57 print("Symmetric difference:", symmetric_diff) # Output: {1, 2, 3, 6, 7, 8}
58
59 # Subset and Superset
60 subset = {1, 2}
61 print("\nIs {1, 2} subset of set1?", subset.issubset(set1)) # Output: True
62 print("Is set1 superset of {1, 2}?", set1.issuperset(subset)) # Output: True

```

Diagram:



### Mnemonic

“CARDS-UI” (Create, Add, Remove, Discard elements, Set operations - Union, Intersection)