

Python Programming (1323203) - Winter 2023 Solution

Milav Dabgar

January 20, 2024

Question 1(a) [3 marks]

Write a pseudocode to check the given number is positive or negative.

Solution

Listing 1. Pseudocode for Number Check

```
1 BEGIN
2   Input number
3   IF number > 0 THEN
4     Display "Number is positive"
5   ELSE IF number < 0 THEN
6     Display "Number is negative"
7   ELSE
8     Display "Number is zero"
9   END IF
10  END
```

Mnemonic

“Compare Zero”

Question 1(b) [4 marks]

Define Algorithm and Design it for Finding maximum from given three Numbers.

Solution

Algorithm Definition: An algorithm is a step-by-step procedure or set of rules designed to solve a specific problem or perform a computation.

Algorithm for Finding Maximum of Three Numbers:

Listing 2. Algorithm for Maximum of Three Numbers

```
1 BEGIN
2   Input num1, num2, num3
3   Set max = num1
4   IF num2 > max THEN
5     Set max = num2
6   END IF
7   IF num3 > max THEN
8     Set max = num3
9   END IF
10  Display max
```

11 | END

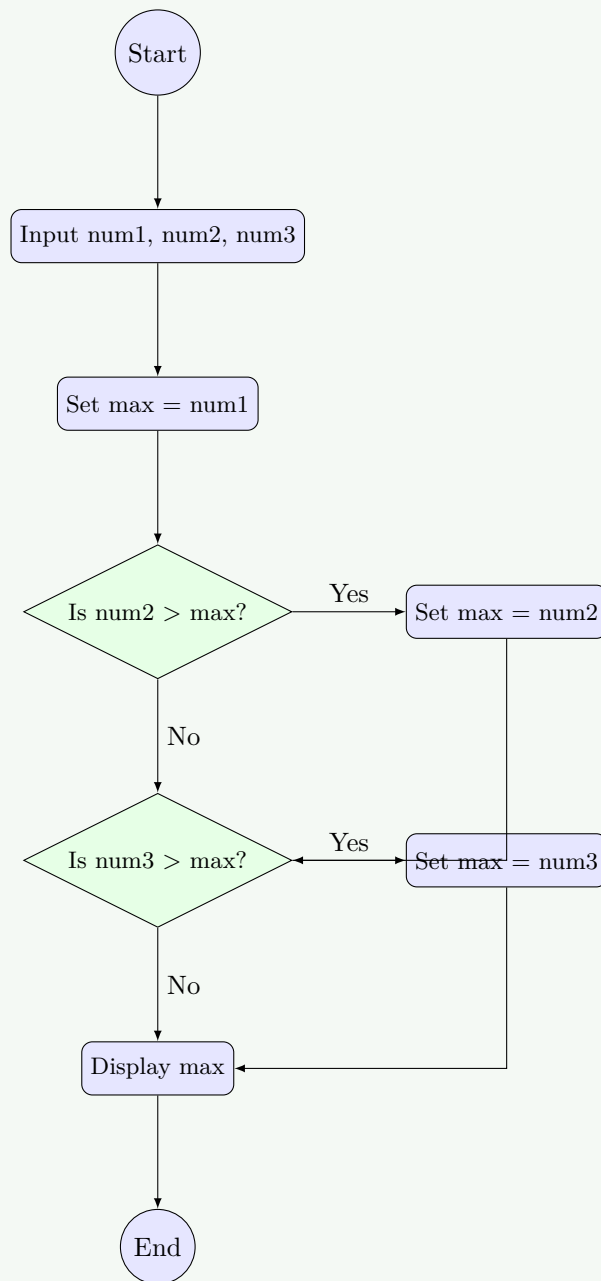


Figure 1. Flowchart to Find Maximum of Three Numbers

Mnemonic

"Compare and Replace"

Question 1(c) [7 marks]

Develop a Python code to convert Temperature parameter from Celsius to Fahrenheit.

Solution

Listing 3. Celsius to Fahrenheit Conversion

```

1 # Program to convert Celsius to Fahrenheit
2
3 # Get the Celsius temperature from user
4 celsius = float(input("Enter temperature in Celsius: "))
5
6 # Convert to Fahrenheit using the formula: F = (C * 9/5) + 32
7 fahrenheit = (celsius * 9/5) + 32
8
9 # Display the result
10 print(f"{celsius}\u00B0C is equal to {fahrenheit}\u00B0F")

```

Table 1. Temperature Conversion

Component	Description
Input	Temperature in Celsius
Formula	$F = (C \times 9/5) + 32$
Output	Temperature in Fahrenheit

Mnemonic

“Multiply by 9, divide by 5, add 32”

Question 1(c OR) [7 marks]

List out all comparison operators and explain each by giving python code example.

Solution

Table 2. Python Comparison Operators

Operator	Description	Example	Result
==	Equal to	5 == 5	True
!=	Not equal to	5 != 6	True
>	Greater than	6 > 3	True
<	Less than	3 < 6	True
>=	Greater than or equal to	5 >= 5	True
<=	Less than or equal to	5 <= 5	True

Code Example:

Listing 4. Comparison Operators Example

```

1 # Python comparison operators example
2 a = 10
3 b = 5
4
5 # Equal to
6 print(f"{a} == {b}: {a == b}") # False
7
8 # Not equal to
9 print(f"{a} != {b}: {a != b}") # True
10

```

```

11 # Greater than
12 print(f"{a} > {b}: {a > b}")    # True
13
14 # Less than
15 print(f"{a} < {b}: {a < b}")    # False
16
17 # Greater than or equal to
18 print(f"{a} >= {b}: {a >= b}")  # True
19
20 # Less than or equal to
21 print(f"{a} <= {b}: {a <= b}")  # False

```

Mnemonic

“CLEAN: Compare, Less than, Equal to, Above, Not equal”

Question 2(a) [3 marks]

Describe data types in python with its examples.

Solution

Table 3. Python Data Types

Data Type	Description	Example
int	Integer values	x = 10
float	Decimal point values	y = 10.5
str	Text or character values	name = "Python"
bool	Logical values (True/False)	is_valid = True
list	Ordered, mutable collection	nums = [1, 2, 3]
tuple	Ordered, immutable collection	point = (5, 10)
dict	Key-value pairs	student = {"name": "John"}

Mnemonic

“NIFTY SLD: Numbers, Integers, Floats, Text, Yes/No, Sequences, Lists, Dictionaries”

Question 2(b) [4 marks]

Explain Nested if in python with python code example.

Solution

Nested if: A conditional statement inside another conditional statement is called a nested if. It allows checking for multiple conditions in sequence.

Listing 5. Nested If Example

```

1 # Nested if example to check if a number is positive, negative, or zero
2 # And if positive, check if it's even or odd
3
4 num = int(input("Enter a number: "))
5

```

```

6  if num > 0:
7      print("Positive number")
8      # Nested if to check if the positive number is even or odd
9      if num % 2 == 0:
10         print("Even number")
11     else:
12         print("Odd number")
13 elif num < 0:
14     print("Negative number")
15 else:
16     print("Zero")

```

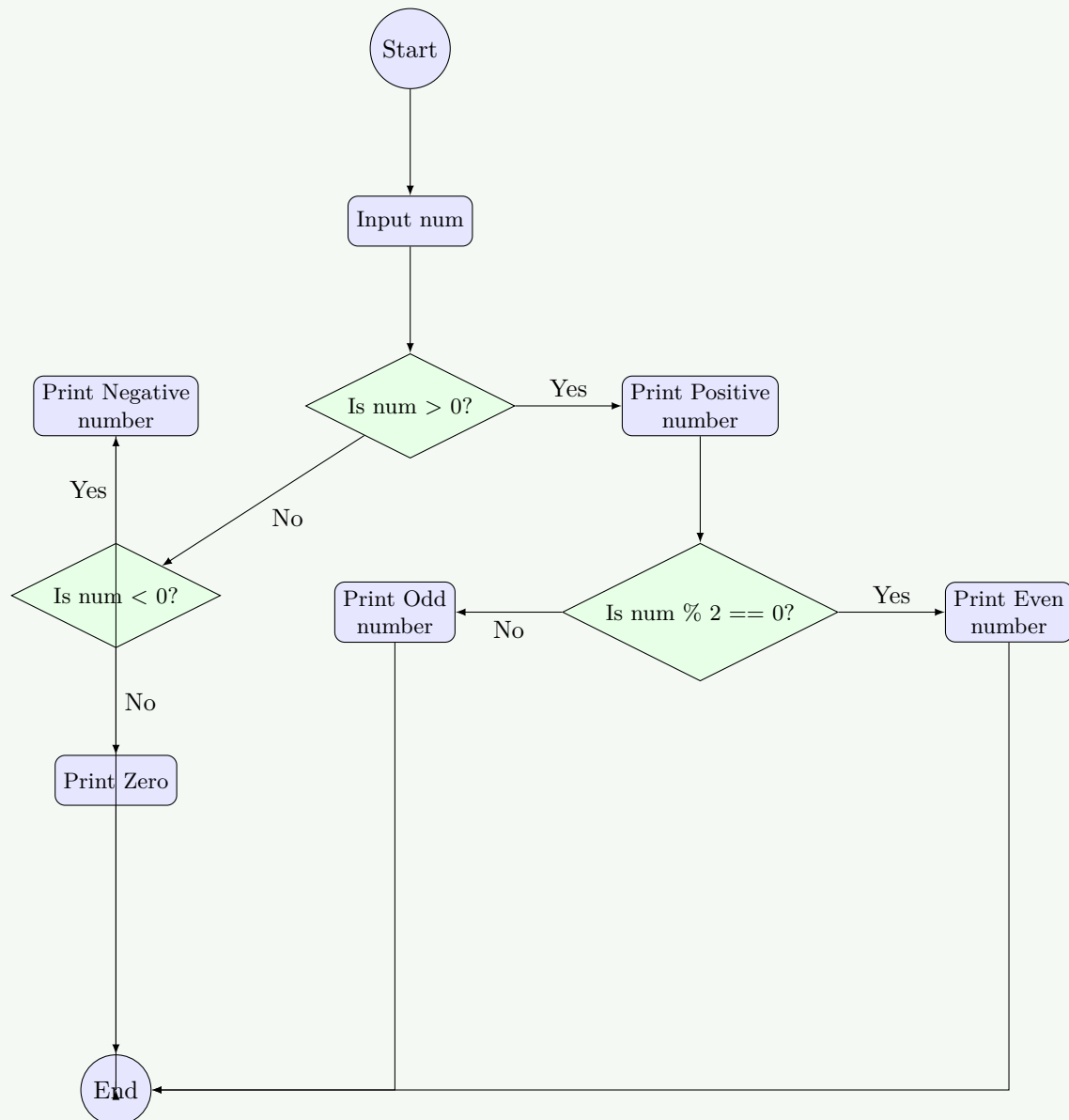


Figure 2. Nested If Flowchart

Mnemonic

“Check Inside Check”

Question 2(c) [7 marks]

Write use of different types of selection / decision making flow of control structures with example.

Solution

Table 4. Selection Control Structures in Python

Structure	Purpose	Use Case
if	Execute code when condition is true	Simple condition check
if-else	Execute one code for true condition, another for false	Binary decision making
if-elif-else	Multiple condition checking	Multiple possible outcomes
Nested if	Condition checking inside another condition	Complex hierarchical decisions
Ternary operator	One-line if-else	Simple conditional assignment

Code Example:

Listing 6. Selection Structures Example

```

1  # Example of different selection structures
2  score = int(input("Enter your score: "))
3
4  # Simple if
5  if score >= 90:
6      print("Excellent!")
7
8  # if-else
9  if score >= 60:
10     print("You passed.")
11 else:
12     print("You failed.")
13
14 # if-elif-else
15 if score >= 90:
16     grade = "A"
17 elif score >= 80:
18     grade = "B"
19 elif score >= 70:
20     grade = "C"
21 elif score >= 60:
22     grade = "D"
23 else:
24     grade = "F"
25 print(f"Your grade is {grade}")
26
27 # Ternary operator
28 result = "Pass" if score >= 60 else "Fail"
29 print(result)

```

Mnemonic

“SCENE: Simple if, Conditions with else, Elif for multiple, Nested for complex, Express with ternary”

Question 2(a OR) [3 marks]

List out rules for defining variables in python.

Solution

Table 5. Rules for Defining Variables in Python

Rule	Description	Example
Start with letter or underscore	First character must be a letter or underscore	<code>name = "John", _count = 10</code>
No special characters	Only letters, numbers, and underscores allowed	<code>user_name</code> (valid), <code>user-name</code> (invalid)
Case sensitive	Uppercase and lowercase are different	<code>age</code> and <code>Age</code> are different variables
No reserved keywords	Cannot use Python keywords as variable names	Cannot use <code>if</code> , <code>for</code> , <code>while</code> , etc.
No spaces	Use underscores instead of spaces	<code>first_name</code> instead of <code>first name</code>

Mnemonic

"SILKS: Start properly, Ignore special chars, Look at case, Keywords avoided, Spaces not allowed"

Question 2(b OR) [4 marks]

Explain For loop in python with necessary python code example.

Solution

For Loop in Python: A for loop is used to iterate over a sequence (list, tuple, string) or other iterable objects. It executes a block of code for each item in the sequence.

Listing 7. For Loop Example

```

1  # Example of for loop in Python
2  # Printing each element in a list
3  fruits = ["apple", "banana", "cherry"]
4  for fruit in fruits:
5      print(fruit)
6
7  # Using range function with for loop
8  print("Numbers from 1 to 5:")
9  for i in range(1, 6):
10     print(i)
11
12 # Using for loop with string
13 name = "Python"
14 for char in name:
15     print(char)

```

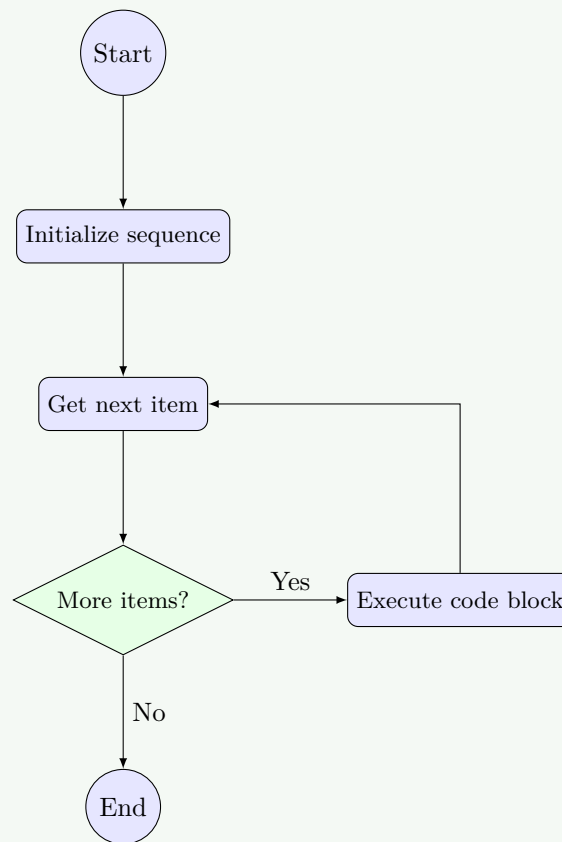


Figure 3. For Loop Flowchart

Mnemonic

“ITEM: Iterate Through Each Member”

Question 2(c OR) [7 marks]

Describe Break and continue statement in python in brief.

Solution**Table 6.** Break and Continue Statements

State-ment	Purpose	Effect
break	Exit the loop immediately	Terminates the current loop and transfers control to the statement following the loop
continue	Skip the current iteration	Jumps to the next iteration of the loop, skipping any code after the continue statement

Code Example:**Listing 8.** Break and Continue Example

```

1 # Break statement example
2 print("Break example:")
3 for i in range(1, 11):
4     if i == 6:

```



```

5     print("Breaking the loop at i =", i)
6     break
7     print(i, end=" ")
8     print("\nLoop ended")
9
10    # Continue statement example
11    print("\nContinue example:")
12    for i in range(1, 11):
13        if i % 2 == 0:
14            continue
15        print(i, end=" ")
16    print("\nOnly odd numbers were printed")

```

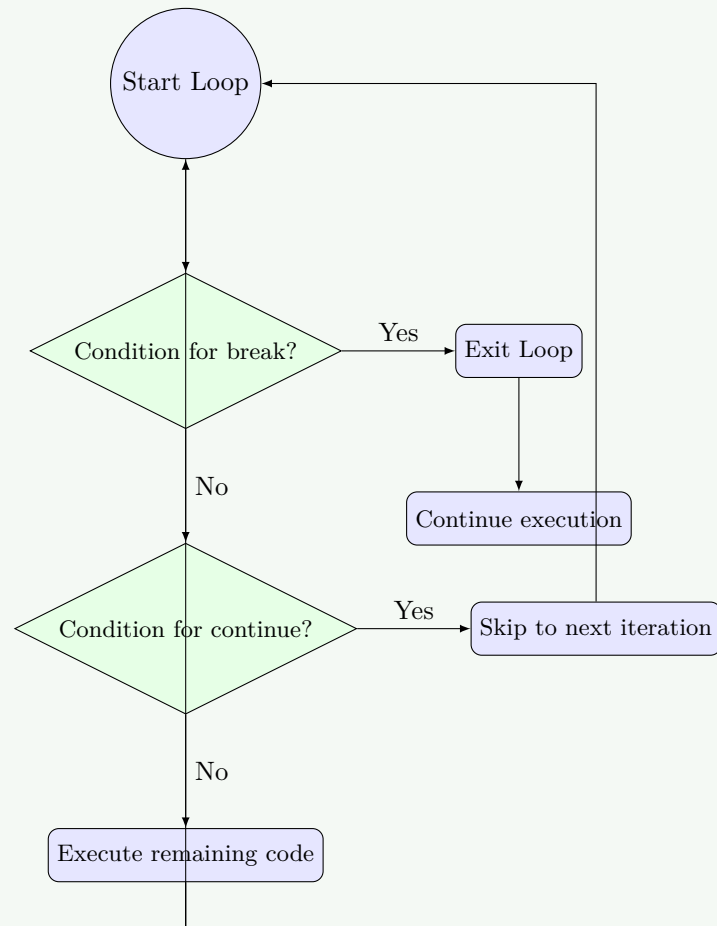


Figure 4. Break and Continue Flowchart

Mnemonic

“EXIT SKIP: EXIT with break, SKIP with continue”

Question 3(a) [3 marks]

Develop a python program to print 1 to 10 numbers using loops.

Solution

Listing 9. Printing 1 to 10

```

1 # Using for loop to print numbers from 1 to 10
2 print("Using for loop:")
3 for i in range(1, 11):
4     print(i, end=" ")
5
6 print("\n\nUsing while loop:")
7 # Using while loop to print numbers from 1 to 10
8 counter = 1
9 while counter <= 10:
10     print(counter, end=" ")
11     counter += 1

```

Table 7. Loop Approaches

Approach	Advantage
For loop with range	Simple, concise, automatically manages counter
While loop	More flexible for complex conditions

Mnemonic

“COUNT UP: Counter Updates in each iteration”

Question 3(b) [4 marks]

Develop a python program to print following pattern using loop:

```

*
**
***
****
*****

```

Solution

Listing 10. Star Pattern Program

```

1 # Print star pattern using for loop
2 rows = 5
3
4 for i in range(1, rows + 1):
5     # Print i stars in each row
6     print("*" * i)

```

Alternative solution with nested loops:

Listing 11. Star Pattern Nested Loop

```

1 # Print star pattern using nested loops
2 rows = 5
3
4 for i in range(1, rows + 1):
5     for j in range(1, i + 1):
6         print("*", end="")
7     print() # New line after each row

```

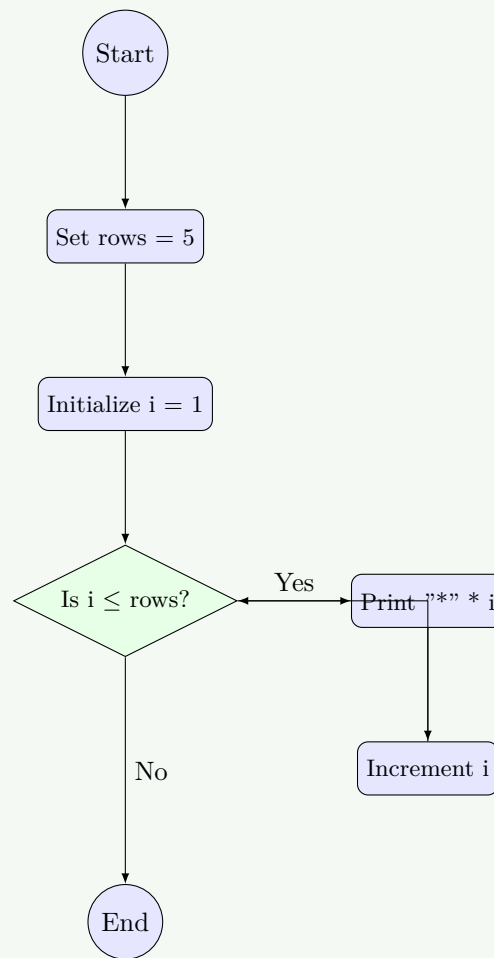


Figure 5. Pattern Printing Flowchart

Mnemonic

“RISE UP: Row Increases, Stars Expand Upward Progressively”

Question 3(c) [7 marks]

Create a user define function to find factorial of the given number.

Solution

Listing 12. Factorial Function

```

1 # Function to find factorial of a given number
2 def factorial(n):
3     # Check if input is valid
4     if not isinstance(n, int) or n < 0:
5         return "Invalid input. Please enter a non-negative integer."
6
7     # Base case: factorial of 0 or 1 is 1
8     if n == 0 or n == 1:
9         return 1
10
11    # Calculate factorial using iteration
  
```

```
12     result = 1
13     for i in range(2, n + 1):
14         result *= i
15
16     return result
17
18 # Test the function
19 number = int(input("Enter a number to find its factorial: "))
20 print(f"Factorial of {number} is {factorial(number)}")
```

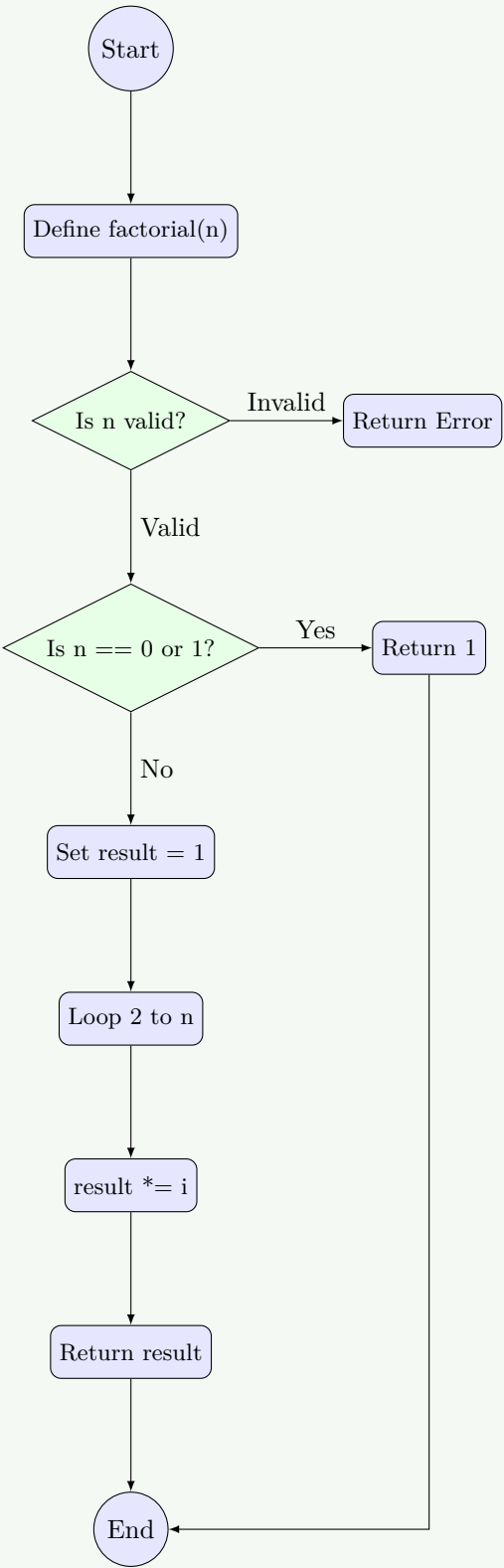


Figure 6. Factorial Function Flowchart

Table 8. Factorial Examples

Number	Calculation	Factorial
0	$0! = 1$	1
1	$1! = 1$	1
3	$3! = 3 \times 2 \times 1$	6
5	$5! = 5 \times 4 \times 3 \times 2 \times 1$	120

Mnemonic

“Multiply Down To One: Multiply all integers down to 1”

Question 3(a OR) [3 marks]

Develop a python code to find odd and even numbers from 1 to N using loops.

Solution**Listing 13.** Odd and Even Numbers Loop

```

1 # Program to find odd and even numbers from 1 to N
2
3 # Get input from user
4 N = int(input("Enter the value of N: "))
5
6 print("Even numbers from 1 to", N, "are:")
7 for i in range(1, N + 1):
8     if i % 2 == 0:
9         print(i, end=" ")
10
11 print("\nOdd numbers from 1 to", N, "are:")
12 for i in range(1, N + 1):
13     if i % 2 != 0:
14         print(i, end=" ")

```

Table 9. Even and Odd Check

Number	Check	Type
Even numbers	<code>number % 2 == 0</code>	2, 4, 6, ...
Odd numbers	<code>number % 2 != 0</code>	1, 3, 5, ...

Mnemonic

“MOD-2: Modulo 2 determines odd or even”

Question 3(b OR) [4 marks]

Develop a code to create nested list and display elements.

Solution**Listing 14.** Nested List Example

```

1 # Program to create and display nested list

```

```

2
3 # Create a nested list
4 nested_list = [
5     [1, 2, 3],
6     [4, 5, 6],
7     [7, 8, 9]
8 ]
9
10 # Display the nested list
11 print("Nested List:", nested_list)
12
13 # Display each element using nested loops
14 print("\nElements of the nested list:")
15 for i in range(len(nested_list)):
16     for j in range(len(nested_list[i])):
17         print(f"nested_list[{i}][{j}] = {nested_list[i][j]}")
18
19 # Alternative way to display using enumerate
20 print("\nUsing enumerate:")
21 for i, inner_list in enumerate(nested_list):
22     for j, value in enumerate(inner_list):
23         print(f"Position ({i}, {j}): {value}")

```

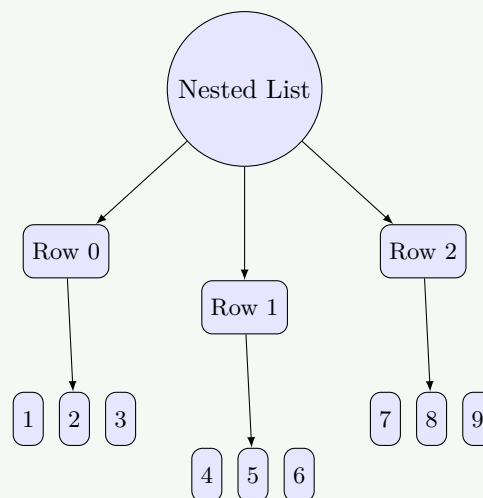


Figure 7. Nested List Structure

Mnemonic

“ROWS COLS: Rows and Columns form the structure”

Question 3(c OR) [7 marks]

Explain local and global variables using examples.

Mnemonic

“GLOBAL SEES ALL: Global variables are visible everywhere”

Question 4(a) [3 marks]

List out Python standard library mathematical functions.

Solution

Table 11. Python Math Module Functions

Function	Description	Example
<code>abs()</code>	Returns absolute value	<code>abs(-5) → 5</code>
<code>pow()</code>	Returns x to power y	<code>pow(2, 3) → 8</code>
<code>max()</code>	Returns largest value	<code>max(5, 10, 15) → 15</code>
<code>min()</code>	Returns smallest value	<code>min(5, 10, 15) → 5</code>
<code>round()</code>	Rounds to nearest integer	<code>round(4.6) → 5</code>
<code>math.sqrt()</code>	Square root	<code>math.sqrt(16) → 4.0</code>
<code>math.sin()</code>	Sine function	<code>math.sin(math.pi/2) → 1.0</code>

Mnemonic

“PEARS Math: Power, Exponents, Arithmetic, Roots, Sine functions in Math”

Question 4(b) [4 marks]

Explain Module in python with example python code of it.

Solution

Module: A module in Python is a file containing Python definitions and statements. The file name is the module name with the suffix `.py` added.

Listing 16. Module Usage Example

```

1 # Example of using math module
2 import math
3
4 # Using mathematical functions from math module
5 radius = 5
6 area = math.pi * math.pow(radius, 2)
7 print(f"Area of circle with radius {radius} is {area:.2f}")
8
9 # Using different import techniques
10 from math import sqrt, sin
11 angle = math.pi / 4
12 print(f"Square root of 25 is {sqrt(25)}")
13 print(f"Sine of {angle} radians is {sin(angle):.4f}")
14
15 # Importing with alias
16 import random as rnd
17 random_number = rnd.randint(1, 100)
18 print(f"Random number between 1 and 100: {random_number}")

```

Table 12. Module Import Techniques

Method	Syntax	Example
Import entire module	import module_name	import math
Import specific items	from module_name import item1, item2	from math import sqrt, sin
Import with alias	import module_name as alias	import random as rnd

Mnemonic

“CODE-LIB: Code Libraries for reuse”

Question 4(c) [7 marks]

Write a Program that determines whether a given number is an 'Armstrong number' or a palindrome using a user-defined function.

Solution

Listing 17. Armstrong and Palindrome Check

```

1  # Function to check if a number is an Armstrong number
2  def is_armstrong(num):
3      # Convert number 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 number of digits
8      armstrong_sum = 0
9      for digit in num_str:
10         armstrong_sum += int(digit) ** n
11
12     # Check if sum equals the original number
13     return armstrong_sum == num
14
15 # Function to check if a number is a palindrome
16 def is_palindrome(num):
17     # Convert number to string and check if it reads the same forwards and backwards
18     num_str = str(num)
19     return num_str == num_str[::-1]
20
21 # Main program
22 number = int(input("Enter a number: "))
23
24 # Check if the number is an Armstrong number
25 if is_armstrong(number):
26     print(f"{number} is an Armstrong number")
27 else:
28     print(f"{number} is not an Armstrong number")
29
30 # Check if the number is a palindrome
31 if is_palindrome(number):
32     print(f"{number} is a palindrome")
33 else:
34     print(f"{number} is not a palindrome")

```

Table 13. Examples

Number	Armstrong Check	Palindrome Check
153	$1^3 + 5^3 + 3^3 = 1 + 125 + 27 = 153 \checkmark$	$153 \neq 351 \times$
121	$1^3 + 2^3 + 1^3 = 1 + 8 + 1 = 10 \neq 121 \times$	$121 = 121 \checkmark$
1634	$1^4 + 6^4 + 3^4 + 4^4 = 1 + 1296 + 81 + 256 = 1634 \checkmark$	$1634 \neq 4361 \times$

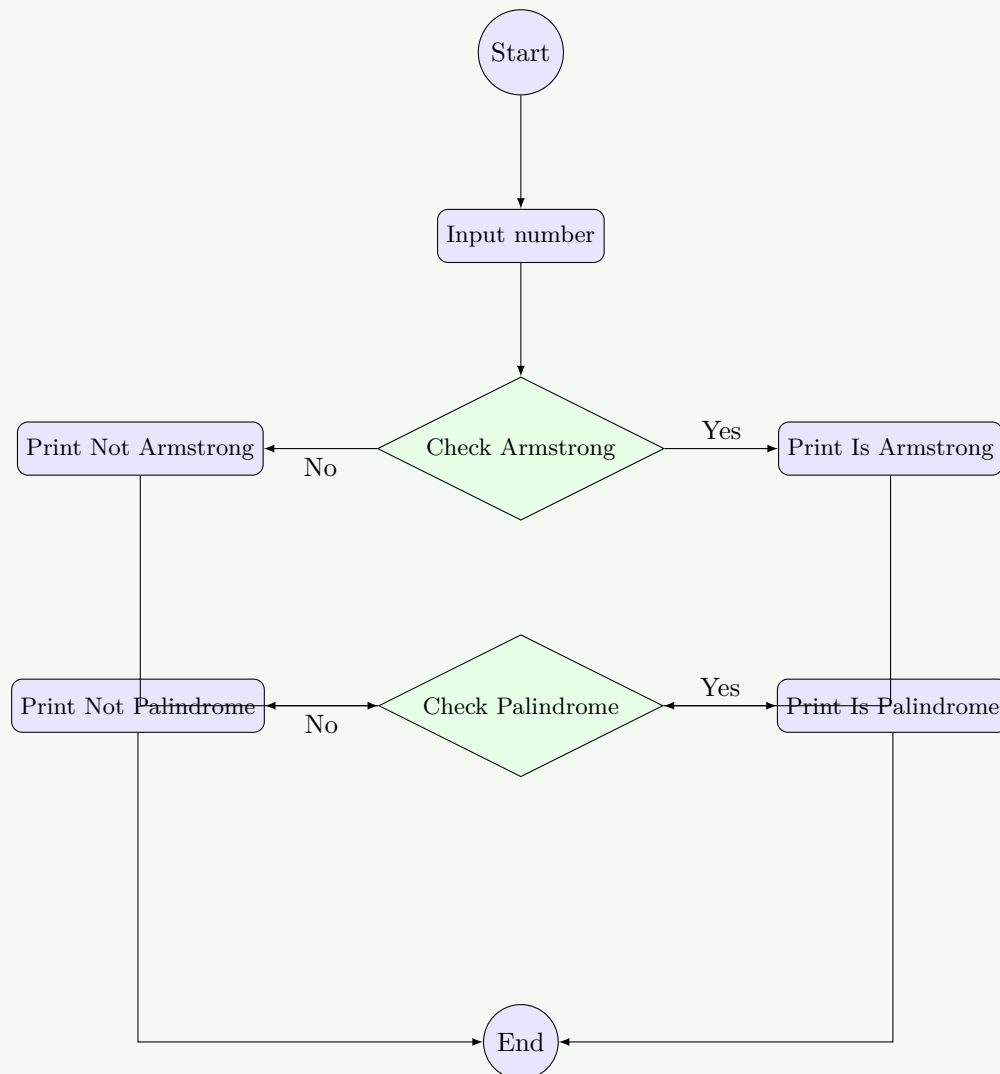


Figure 9. Validation Flowchart

Mnemonic

“SAME SUM: SAME forwards and backwards for palindrome, SUM of powered digits for Armstrong”

Question 4(a OR) [3 marks]

Explain built in functions in python.

Solution

Built-in Functions: These are functions that are part of Python’s standard library and available without importing any module.

Table 14. Common Python Built-in Functions

Function	Purpose	Example
<code>print()</code>	Display output	<code>print("Hello")</code>
<code>input()</code>	Get user input	<code>name = input("Name: ")</code>
<code>len()</code>	Return object length	<code>len([1, 2, 3]) → 3</code>
<code>type()</code>	Return object type	<code>type(5) → <class 'int'></code>
<code>int(), float(), str()</code>	Convert to specific type	<code>int("5") → 5</code>
<code>range()</code>	Generate sequence	<code>list(range(3)) → [0, 1, 2]</code>
<code>sum()</code>	Calculate sum	<code>sum([1, 2, 3]) → 6</code>

Mnemonic

“PITS LCR: Print, Input, Type, Sum, Len, Convert, Range”

Question 4(b OR) [4 marks]

Describe python math module by giving one python code example.

Solution

Python Math Module: The math module provides access to mathematical functions defined by the C standard.

Listing 18. Math Module Example

```

1  # Example using math module
2  import math
3
4  # Basic constants
5  print(f"Value of pi: {math.pi}")
6  print(f"Value of e: {math.e}")
7
8  # Trigonometric functions (argument in radians)
9  angle = math.pi / 3 # 60 degrees
10 print(f"Sine of {angle:.2f} radians: {math.sin(angle):.4f}")
11 print(f"Cosine of {angle:.2f} radians: {math.cos(angle):.4f}")
12 print(f"Tangent of {angle:.2f} radians: {math.tan(angle):.4f}")
13
14 # Logarithmic and exponential functions
15 x = 10
16 print(f"Natural logarithm of {x}: {math.log(x):.4f}")
17 print(f"Logarithm base 10 of {x}: {math.log10(x):.4f}")
18 print(f"e raised to power {x}: {math.exp(x):.4f}")
19
20 # Other functions
21 print(f"Square root of 25: {math.sqrt(25)}")
22 print(f"Ceiling of 4.3: {math.ceil(4.3)}")
23 print(f"Floor of 4.7: {math.floor(4.7)}")

```

Table 15. Math Module Categories

Category	Functions
Constants	<code>math.pi</code> , <code>math.e</code>
Trigonometric	<code>sin()</code> , <code>cos()</code> , <code>tan()</code>
Logarithmic	<code>log()</code> , <code>log10()</code> , <code>exp()</code>
Numeric	<code>sqrt()</code> , <code>ceil()</code> , <code>floor()</code>

Mnemonic

“PENT: Pi/constants, Exponents, Numbers, Trigonometry”

Question 4(c OR) [7 marks]

Explain concept of scope of variable in Python and Apply global and local variable concepts in python program.

Solution

Scope of Variables in Python: The scope of a variable determines where in the program a variable is accessible or visible.

Table 16. Variable Scope Types

Scope	Description	Access
Local	Variables defined inside a function	Only within the function
Global	Variables defined at the top level	Throughout the program
Enclosing	Variables in outer function of nested functions	In the outer and inner function
Built-in	Pre-defined variables in Python	Throughout the program

Listing 19. Variable Scope Example

```

1  # Variable scope demonstration
2
3  # Global variable
4  count = 0
5
6  def outer_function():
7      # Enclosing scope variable
8      name = "Python"
9
10     def inner_function():
11         # Local variable
12         age = 30
13         # Accessing global variable
14         global count
15         count += 1
16         # Accessing enclosing variable
17         print(f"Inside inner_function: name is {name}")
18         print(f"Inside inner_function: age is {age}")
19         print(f"Inside inner_function: count is {count}")
20
21     # Local variable to outer_function
22     language = "Programming"
23     print(f"Inside outer_function: name is {name}")
24     print(f"Inside outer_function: language is {language}")
25     print(f"Inside outer_function: count is {count}")
26
27     # Call inner function
28     inner_function()
29
30 # Main program
31 print(f"Global scope: count is {count}")
32 outer_function()
33 print(f"Global scope after function call: count is {count}")

```

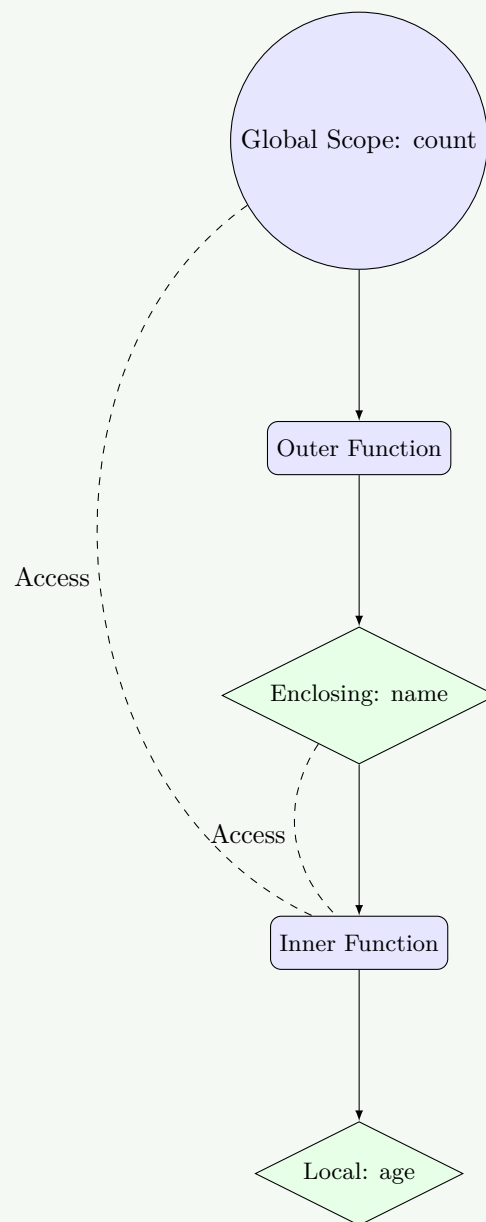


Figure 10. Rule of LEGB

Mnemonic

“LEGB: Local, Enclosing, Global, Built-in - order of scope lookup”

Question 5(a) [3 marks]

Develop a python program to swap two elements in given list

Solution**Listing 20. Swapping List Elements**

```

1 # Program to swap two elements in a list
2

```

```

3 # Create a list
4 my_list = [10, 20, 30, 40, 50]
5 print("Original list:", my_list)
6
7 # Get positions to swap
8 pos1 = int(input("Enter first position (index starts from 0): "))
9 pos2 = int(input("Enter second position (index starts from 0): "))
10
11 # Swap elements using a temporary variable
12 if 0 <= pos1 < len(my_list) and 0 <= pos2 < len(my_list):
13     # Swapping
14     temp = my_list[pos1]
15     my_list[pos1] = my_list[pos2]
16     my_list[pos2] = temp
17
18     print(f"List after swapping elements at positions {pos1} and {pos2}:", my_list)
19 else:
20     print("Invalid positions! Positions should be within list range.")

```

Alternative method:

Listing 21. Pythonic Swap

```

1 # Swap using Python's tuple unpacking (more pythonic)
2 if 0 <= pos1 < len(my_list) and 0 <= pos2 < len(my_list):
3     my_list[pos1], my_list[pos2] = my_list[pos2], my_list[pos1]
4     print(f"List after swapping elements at positions {pos1} and {pos2}:", my_list)

```

Table 17. Swapping Methods

Method	Code
Using temp variable	temp = a; a = b; b = temp
Python tuple unpacking	a, b = b, a

Mnemonic

“TEMP SWAP: Temporary variable helps safe swapping”

Question 5(b) [4 marks]

Explain nested list by giving example.

Solution

Nested List: A nested list is a list that contains other lists as its elements, creating a multi-dimensional data structure.

Listing 22. Nested List Operations

```

1 # Creating a nested list (3x3 matrix)
2 matrix = [
3     [1, 2, 3],
4     [4, 5, 6],
5     [7, 8, 9]
6 ]
7
8 # Accessing elements
9 print("Complete matrix:", matrix)

```

```

10 print("First row:", matrix[0])
11 print("Element at row 1, column 2:", matrix[0][1]) # Output: 2
12
13 # Modifying elements
14 matrix[1][1] = 50
15 print("Matrix after modification:", matrix)
16
17 # Iterating through a nested list
18 print("\nPrinting the matrix:")
19 for row in matrix:
20     for element in row:
21         print(element, end=" ")
22     print() # New line after each row

```

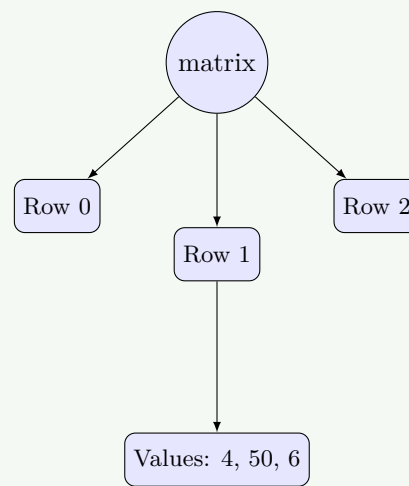


Figure 11. Nested List Structure

Table 18. Nested List Operations

Operation	Syntax	Example
Access element	<code>list[row][col]</code>	<code>matrix[0][1]</code>
Modify element	<code>list[row][col] = new_value</code>	<code>matrix[1][1] = 50</code>
Add new row	<code>list.append([...])</code>	<code>matrix.append([10, 11, 12])</code>

Mnemonic

“MARS: Matrix Access with Row and column Structure”

Question 5(c) [7 marks]

Explain string operations with examples.

Solution

Table 19. String Operations in Python

Operation	Description	Example
Concatenation	Joining strings	"Hello" + " World" → "Hello World"
Repetition	Repeating strings	"Python" * 3 → "PythonPythonPython"
Slicing	Extract substring	"Python"[1:4] → "yth"
Indexing	Access character	"Python"[0] → "P"
Length	Count characters	len("Python") → 6
Membership	Check if present	"P" in "Python" → True
Comparison	Compare strings	"apple" < "banana" → True

Listing 23. String Operations Example

```

1 # String operations demonstration
2 text = "Python Programming"
3
4 # Indexing
5 print("First character:", text[0])
6 print("Last character:", text[-1])
7
8 # Slicing
9 print("First word:", text[:6])
10 print("Second word:", text[7:])
11 print("Middle characters:", text[3:10])
12 print("Reverse:", text[::-1])
13
14 # String methods
15 print("Uppercase:", text.upper())
16 print("Lowercase:", text.lower())
17 print("Replace 'P' with 'J':", text.replace("P", "J"))
18 print("Split by space:", text.split())
19 print("Count 'm':", text.count('m'))
20 print("Find 'gram':", text.find("gram"))
21
22 # Check operations
23 print("Is alphanumeric?", text.isalnum())
24 print("Starts with 'Py'?", text.startswith("Py"))
25 print("Ends with 'ing'?", text.endswith("ing"))

```

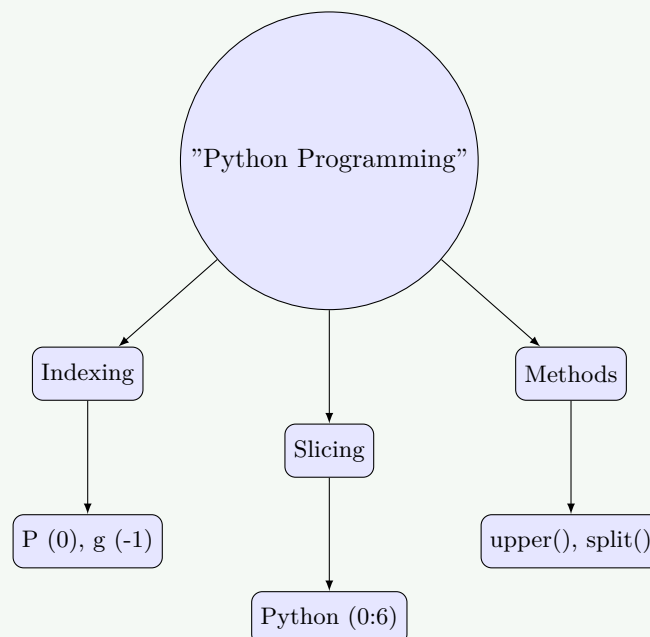


Figure 12. String Operations**Mnemonic**

“SCREAM: Slice, Concat, Replace, Extract, Access, Methods”

Question 5(a OR) [3 marks]

Develop a python program to find sum of all elements in given list

Solution**Listing 24.** Sum of List Elements

```

1  # Program to find sum of all elements in a list
2
3  # Method 1: Using built-in sum() function
4  def sum_list_builtin(numbers):
5      return sum(numbers)
6
7  # Method 2: Using a loop
8  def sum_list_loop(numbers):
9      total = 0
10     for num in numbers:
11         total += num
12     return total
13
14 # Create a sample list
15 my_list = [10, 20, 30, 40, 50]
16 print("List:", my_list)
17
18 # Calculate sum using built-in function
19 print("Sum using built-in function:", sum_list_builtin(my_list))
20
21 # Calculate sum using loop
22 print("Sum using loop:", sum_list_loop(my_list))

```

Table 20. Sum Methods Comparison

Method	Advantage
Built-in sum()	Simple, efficient, fast
Loop approach	Works for custom summing logic

Mnemonic

“ADD ALL: Add All elements in sequence”

Question 5(b OR) [4 marks]

Explain indexing and slicing operations in python list

Solution

Table 21. Indexing and Slicing Operations

Operation	Syntax	Description	Example
Positive Indexing	list[i]	Access item at position i	fruits[0]
Negative Indexing	list[-i]	Access item from end	fruits[-1]
Basic Slicing	list[s:e]	Items from start to end-1	fruits[1:3]
Slice with Step	list[s:e:st]	Items with interval of step	nums[1:6:2]
Reverse	list[::-1]	Reverse the list	fruits[::-1]

Listing 25. Indexing and Slicing Demo

```

1 # Indexing and slicing demonstration
2 fruits = ["apple", "banana", "cherry", "date", "elderberry", "fig"]
3 print("Original list:", fruits)
4
5 # Indexing
6 print("\nIndexing examples:")
7 print("First item:", fruits[0]) # apple
8 print("Last item:", fruits[-1]) # fig
9
10 # Slicing
11 print("\nSlicing examples:")
12 print("First three items:", fruits[:3])
13 print("Last three items:", fruits[-3:])
14 print("Middle items:", fruits[2:4])
15 print("Every second item:", fruits[::2])
16 print("Reversed list:", fruits[::-1])

```

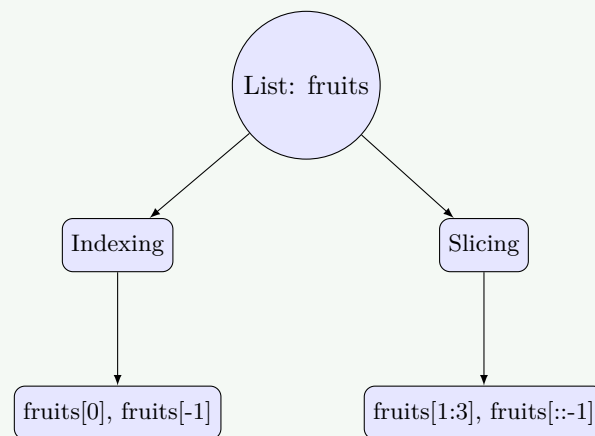


Figure 13. Indexing and Slicing

Mnemonic

“START-END-STEP: Slicing syntax: [start:end:step]”

Question 5(c OR) [7 marks]

Explain tuple in brief with necessary example.

Solution

Tuple: A tuple is an ordered, immutable collection of elements. Once created, the elements cannot be changed.

Table 22. Tuple vs List

Feature	Tuple	List
Syntax	(item1, item2)	[item1, item2]
Mutability	Immutable (cannot change)	Mutable (can change)
Performance	Faster	Slower
Use Case	Fixed data, keys	Dynamic data

Listing 26. Tuple Example

```

1 # Creating tuples
2 empty_tuple = ()
3 single_item_tuple = (1,) # Comma is necessary for single item
4 mixed_tuple = (1, "Hello", 3.14, True)
5 nested_tuple = (1, 2, (3, 4), 5)
6
7 # Accessing tuple elements
8 print("First item:", mixed_tuple[0]) # 1
9 print("Nested tuple element:", nested_tuple[2][0]) # 3
10
11 # Tuple operations
12 combined_tuple = mixed_tuple + nested_tuple
13 print("Combined tuple:", combined_tuple)
14
15 # This will cause error as tuples are immutable
16 # mixed_tuple[0] = 100 # TypeError

```

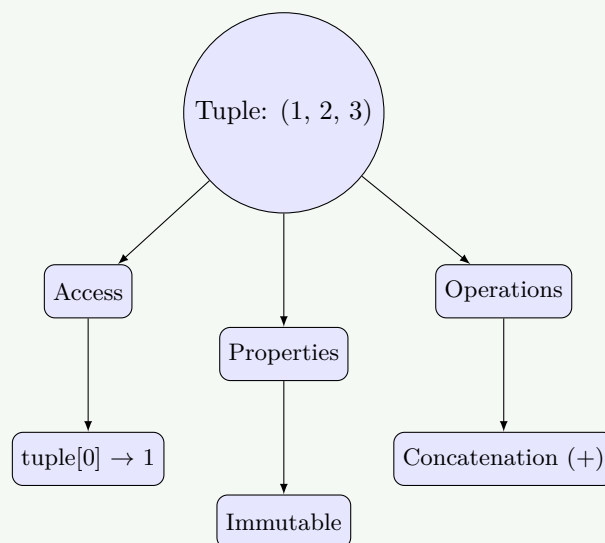


Figure 14. Tuple Concepts

Mnemonic

“IPAC: Immutable, Parentheses, Access only, Cannot modify”

