Python Programming (4311601)

Milav Dabgar

Winter 2023

Question Question 1(a) [03 marks]

What is Flow chart? List out symbols used in Flow chart.

Solution

A **flowchart** is a graphical representation of an algorithm that shows the sequence of steps and decision points in a process using standardized symbols.

Table 1. Flowchart Symbols Table

Symbol	Name	Purpose
Oval	Terminal	Start/End of program
Rectangle	Process	Processing/Calculation steps
Diamond	Decision	Conditional statements
Parallelogram	Input/Output	Data input or output
Circle	Connector	Connect flowchart parts
Arrow	Flow line	Direction of flow

Key Points:

- **Visual representation**: Shows program logic graphically
- **Step-by-step**: Displays sequential flow of operations
- **Decision making**: Diamond symbols show conditional branches

Mnemonic

"Flow Charts Show Program Steps Visually"

Question Question 1(b) [04 marks]

Write a short note on for loop.

Solution

The **for loop** is used to iterate over a sequence (list, tuple, string, range) in Python.

Table 2. For Loop Table

Component	Syntax	Example
Basic	`for variable in sequence:`	`for i in range(5):`
Range	`range(start, stop, step)`	`range(1, 10, 2)`
List	`for item in list:`	`for x in [1,2,3]:`
String	`for char in string:`	`for c in "hello":`

Simple Code Example:

```
for i in range(3):
    print(i)
# Output: 0, 1, 2
```

Key Features:

- **Automatic iteration**: No manual counter needed
- **Sequence traversal**: Works with any iterable object
- **Range function**: Creates number sequences easily

Mnemonic

"For Loops Iterate Through Sequences"

Question Question 1(c) [07 marks]

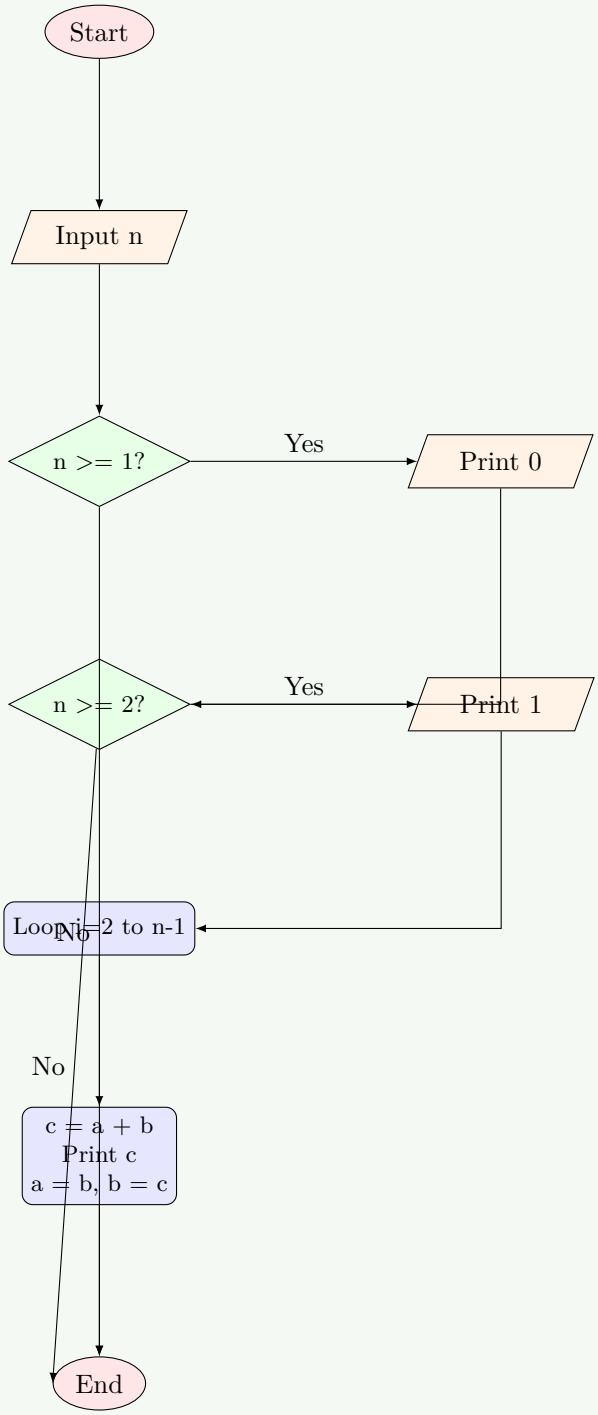
Write a program to display Fibonacci series up to nth term where n is provided by the user.

Solution

Fibonacci Series Program:

```
# Get number of terms from user
n = int(input("Enter number of terms: "))

# Initialize first two terms
a, b = 0, 1

# Display first term
if n >= 1:
    print(a, end=" ")

# Display second term
if n >= 2:
    print(b, end=" ")

# Generate remaining terms
for i in range(2, n):
    c = a + b
    print(c, end=" ")
    a, b = b, c
```

Algorithm Flow:



Key Concepts:

- **Sequential generation**: Each term = sum of previous two
- **Variable swapping**: Update a, b values efficiently
- **User input**: Dynamic series length

Mnemonic

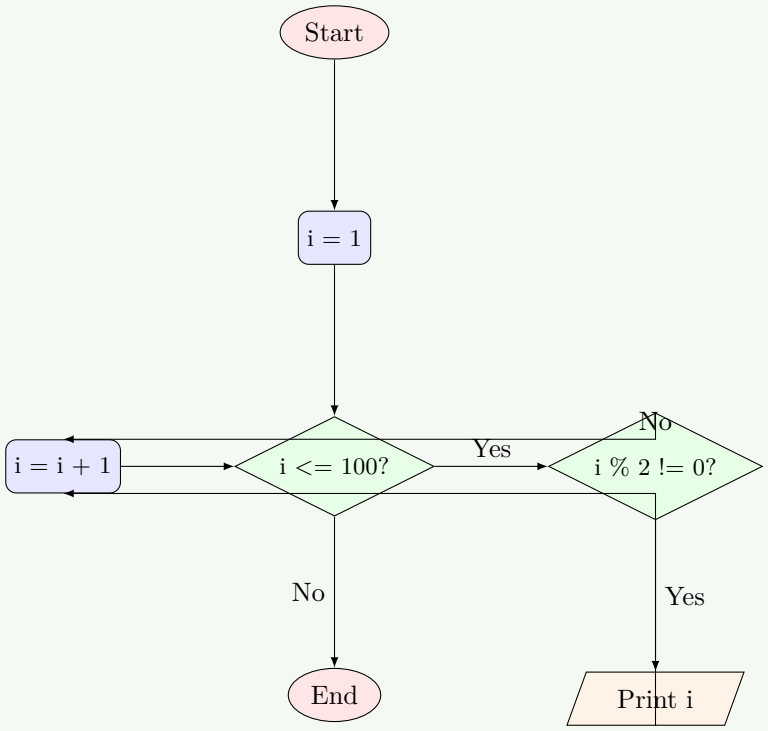
"Fibonacci: Add Previous Two Numbers"

Question Question 1(c OR) [07 marks]

Draw a flow chart to print ODD numbers from 1 to 100.

Solution

Flowchart for ODD Numbers 1 to 100:



Corresponding Python Code:

```
for i in range(1, 101):
    if i % 2 != 0:
        print(i, end=" ")
```

Alternative Method:

```
for i in range(1, 101, 2):
    print(i, end=" ")
```

Key Elements:

- **Loop control**: i from 1 to 100
- **Odd check**: i % 2 != 0 condition
- **Step increment**: Move to next number

Mnemonic

"Odd Numbers: Remainder 1 When Divided by 2"

Question Question 2(a) [03 marks]

Write a Program to find whether a number is Palindrome or not.

Solution

Palindrome Check Program:

```python
# Input number
num = int(input("Enter a number: "))
temp = num
reverse = 0

# Reverse the number
while temp > 0:
    reverse = reverse * 10 + temp % 10
    temp = temp // 10

# Check palindrome
if num == reverse:
    print(f"{num} is palindrome")
else:
    print(f"{num} is not palindrome")
```

Algorithm Table:

Table 3. Algorithm Steps

Step	Operation	Example (121)
1	Get last digit	121 % 10 = 1
2	Build reverse	0*10 + 1 = 1
3	Remove last digit	121 // 10 = 12
4	Repeat until 0	Continue process

Key Points:

- **Digit extraction**: Use modulo (%) operator
- **Reverse building**: Multiply by 10 and add digit
- **Comparison**: Original equals reversed

Mnemonic

"Palindrome Reads Same Forward Backward"

Question Question 2(b) [04 marks]

Explain features of Python Programming.

Solution

Python Features Table:

Table 4. Python Features

Feature	Description	Benefit
Easy Syntax	Simple, readable code	Faster development
Interpreted	No compilation needed	Quick testing
Object-Oriented	Classes and objects support	Code reusability
Open Source	Free to use	No licensing cost
Cross-Platform	Runs on multiple OS	Wide compatibility
Large Libraries	Extensive built-in modules	Rich functionality

Key Advantages:

- **Beginner-friendly**: Easy to learn and understand
- **Versatile**: Web development, AI, data science
- **Community support**: Large developer community
- **Dynamic typing**: No variable type declaration needed

Mnemonic

"Python: Easy, Powerful, Popular Programming"

Question Question 2(c) [07 marks]

Explain basic structure of Python Program.

Solution

Python Program Structure:

```python
#!/usr/bin/env python3
# Shebang line (optional)

"""
Documentation string (docstring)
Describes program purpose
"""

# Import statements
import math
from datetime import date

# Global variables
PI = 3.14159
count = 0

# Function definitions
def calculate_area(radius):
    """Calculate circle area"""
    return PI * radius * radius

# Class definitions
class Calculator:
    def __init__(self):
        self.result = 0

# Main program execution
if __name__ == "__main__":
    # Program logic here
    radius = 5
    area = calculate_area(radius)
    print(f"Area: {area}")
```

Structure Components Table:

Table 5. Program Components

Component	Purpose	Example
Shebang	System interpreter	`#!/usr/bin/env python3`
Docstring	Program documentation	`"""Program description"""`
Imports	External modules	`import math`
Variables	Global data storage	`PI = 3.14159`
Functions	Reusable code blocks	`def function_name():`

Key Principles:

- **Indentation**: Defines code blocks (4 spaces recommended)
- **Comments**: Use # for single line, """ """ for multi-line
- **Modularity**: Organize code in functions and classes

Mnemonic

"Structure: Import, Define, Execute"

Question Question 2(a OR) [03 marks]

Write a Program to reverse a string.

Solution

String Reversal Program:

```
# Method 1: Using slicing
string = input("Enter a string: ")
reversed_string = string[::-1]
print(f"Reversed: {reversed_string}")

# Method 2: Using loop
string = input("Enter a string: ")
reversed_string = ""
for char in string:
    reversed_string = char + reversed_string
print(f"Reversed: {reversed_string}")
```

Reversal Methods Table:

Table 6. Reversal Methods

Method	Syntax	Example
Slicing	`string[::-1]`	"hello" → "*olleh*"
Loop	Build character by character	Add each char to front
Built-in	`"".join(reversed(string))`	Join reversed sequence

Mnemonic

"Reverse: Last Character First"

Question Question 2(b OR) [04 marks]

Explain Logical Operators with example.

Solution

Python Logical Operators:

Table 7. Logical Operators

Operator	Symbol	Description	Example	Result
AND	`and`	Both conditions true	`True and False`	`False`
OR	`or`	At least one condition true	`True or False`	`True`
NOT	`not`	Opposite of condition	`not True`	`False`

Example Code:

```
a = 10
b = 5

# AND operator
if a > 5 and b < 10:
    print("Both conditions true")

# OR operator
if a > 15 or b < 10:
    print("At least one condition true")

# NOT operator
if not (a < 5):
    print("a is not less than 5")
```

Truth Table:

Table 8. Truth Table

A	B	A and B	A or B	not A
T	T	T	T	F
T	F	F	T	F
F	T	F	T	T
F	F	F	F	T

Mnemonic

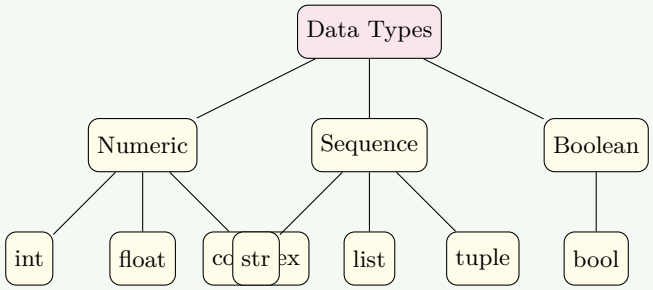
"AND needs All, OR needs One, NOT reverses"

Question Question 2(c OR) [07 marks]

Explain different Data Types in Python Programming language

Solution

Python Data Types Classification:



Data Types Table:

Table 9. Data Types

Type	Example	Description	Mutable
int	`42`	Whole numbers	No
float	`3.14`	Decimal numbers	No
str	`"hello"`	Text data	No
list	`[1,2,3]`	Ordered collection	Yes
tuple	`(1,2,3)`	Ordered immutable	No
dict	`{"a":1}`	Key-value pairs	Yes
bool	`True/False`	Boolean values	No
set	`{1,2,3}`	Unique elements	Yes

Example Code:

```
# Numeric types
age = 25          # int
price = 99.99     # float
complex_num = 3+4j # complex

# Sequence types
name = "Python"          # string
numbers = [1,2,3,4]      # list
coordinates = (10,20)    # tuple

# Other types
is_active = True         # boolean
unique_items = {1,2,3}   # set
student = {"name":"John", "age":20}  # dict
```

Mnemonic

"Python Types: Numbers, Sequences, Collections"

Question Question 3(a) [03 marks]

What is flow control in Python? Explain with example

Solution

Flow control manages the execution order of program statements using conditional and loop structures.
Flow Control Types Table:

Table 10. Flow Control Types

Type	Statement	Purpose	Example
Sequential	Normal execution	Line by line	`print("Hello")`
Selection	if, elif, else	Decision making	`if x > 0:`
Iteration	for, while	Repetition	`for i in range(5):`
Jump	break, continue	Loop control	`break`

Example Code:

```python
# Selection example
age = 18
if age >= 18:
    print("Adult")
else:
    print("Minor")

# Iteration example
for i in range(3):
    print(f"Count: {i}")
```

Mnemonic

"Flow Control: Decide, Repeat, Jump"

Question Question 3(b) [04 marks]

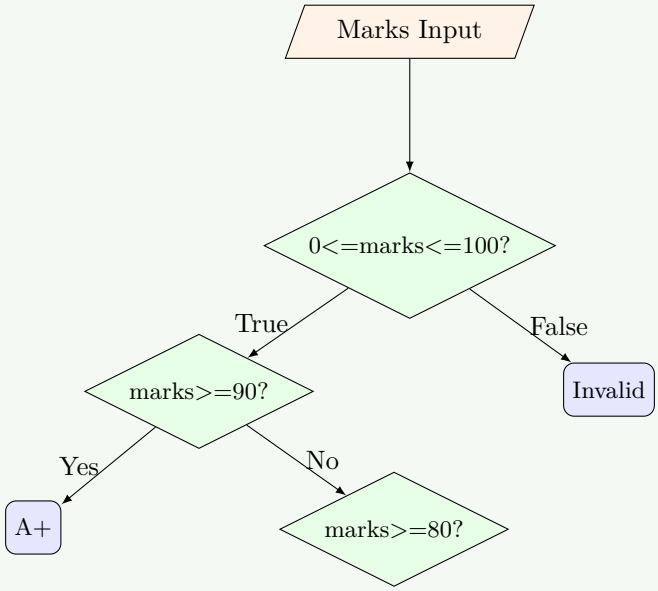
Write a program to explain nested if statement.

Solution

Nested If Statement Program:

```python
# Grade calculation using nested if
marks = int(input("Enter marks: "))

if marks >= 0 and marks <= 100:
    if marks >= 90:
        grade = "A+"
    elif marks >= 80:
        if marks >= 85:
            grade = "A"
        else:
            grade = "B+"
    elif marks >= 70:
        grade = "B"
    elif marks >= 60:
        grade = "C"
    else:
        grade = "F"
    print(f"Grade: {grade}")
else:
    print("Invalid marks")
```

Nested Structure Diagram:



Mnemonic

"Nested If: Decisions Within Decisions"

Question Question 3(c) [07 marks]

Write a program to Explain types of Arguments and Parameters.

Solution

Types of Arguments and Parameters:

```
# 1. Positional Arguments
def greet(name, age):
    print(f"Hello {name}, you are {age} years old")

greet("John", 25)  # Positional arguments

# 2. Keyword Arguments
greet(age=30, name="Alice")  # Keyword arguments

# 3. Default Parameters
def introduce(name, city="Unknown"):
    print(f"{name} lives in {city}")

introduce("Bob")  # Uses default value
introduce("Carol", "NYC")  # Override default

# 4. Variable-length Arguments (*args)
def sum_all(*numbers):
    return sum(numbers)

result = sum_all(1, 2, 3, 4, 5)
print(f"Sum: {result}")

# 5. Keyword Variable Arguments (**kwargs)
def display_info(**info):
    for key, value in info.items():
```

```
        print(f"{key}: {value}")

display_info(name="David", age=28, city="Boston")
```

Parameters Types Table:

Table 11. Parameter Types

Type	Syntax	Example	Description
Positional	`def func(a, b):`	`func(1, 2)`	Order matters
Keyword	`def func(a, b):`	`func(b=2, a=1)`	Name specified
Default	`def func(a, b=10):`	`func(5)`	Default value
*args	`def func(*args):`	`func(1,2,3)`	Variable positional
**kwargs	`def func(**kwargs):`	`func(a=1, b=2)`	Variable keyword

Mnemonic

"Parameters: Position, Keywords, Defaults, Variables"

Question Question 3(a OR) [03 marks]

Explain break and continue statement with example.

Solution

Break and Continue Statements:
Break Statement:

```
# Break example - exit loop
for i in range(10):
    if i == 5:
        break
    print(i)
# Output: 0, 1, 2, 3, 4
```

Continue Statement:

```
# Continue example - skip iteration
for i in range(5):
    if i == 2:
        continue
    print(i)
# Output: 0, 1, 3, 4
```

Comparison Table:

Table 12. Break vs Continue

Statement	Purpose	Action	Example Use
break	Exit loop	Terminates entire loop	Exit on condition
continue	Skip iteration	Jump to next iteration	Skip specific values

Mnemonic

"Break Exits, Continue Skips"

Question Question 3(b OR) [04 marks]

Create a program to display the following pattern

Solution

Pattern:

1
12
123
1234
12345

Number Pattern Program:

```python
# Method 1: Using nested loops
rows = 5
for i in range(1, rows + 1):
    for j in range(1, i + 1):
        print(j, end="")
    print()  # New line

# Method 2: Using string manipulation
for i in range(1, 6):
    line = ""
    for j in range(1, i + 1):
        line += str(j)
    print(line)
```

Mnemonic

"Pattern: Row Number Determines Column Count"

Question Question 3(c OR) [07 marks]

Explain the following mathematical functions by writing a code for each: 1. abs() 2. max() 3. pow() 4. sum()

Solution

Mathematical Functions in Python:

```python
# 1. abs() - Absolute value
numbers = [-5, 3.7, -10.2, 0]
print("abs() function examples:")
for num in numbers:
    print(f"abs({num}) = {abs(num)}")

# 2. max() - Maximum value
list1 = [4, 7, 2, 9, 1]
print(f"\nmax() function examples:")
print(f"max({list1}) = {max(list1)}")
print(f"max(10, 25, 5) = {max(10, 25, 5)}")
print(f"max('hello') = {max('hello')}")  # Alphabetically

# 3. pow() - Power function
print(f"\npow() function examples:")
```

```python
print(f"pow(2, 3) = {pow(2, 3)}")       # 2^3 = 8
print(f"pow(5, 2) = {pow(5, 2)}")       # 5^2 = 25
print(f"pow(8, 1/3) = {pow(8, 1/3)}")  # Cube root of 8

# 4. sum() - Sum of sequence
numbers = [1, 2, 3, 4, 5]
print(f"\nsum() function examples:")
print(f"sum({numbers}) = {sum(numbers)}")
print(f"sum({numbers}, 10) = {sum(numbers, 10)}")  # With start value
```

Functions Summary Table:

Table 13. Math Functions

Function	Syntax	Purpose	Example	Result
abs()	`abs(x)`	Absolute value	`abs(-5)`	5
max()	`max(iterable)`	Maximum value	`max([1,5,3])`	5
pow()	`pow(x, y)`	x raised to power y	`pow(2, 3)`	8
sum()	`sum(iterable)`	Sum of values	`sum([1,2,3])`	6

Mnemonic

"Math Functions: Absolute, Maximum, Power, Sum"

Question Question 4(a) [03 marks]

Explain scope of variables.

Solution

Variable Scope refers to the region where a variable can be accessed in a program.
Scope Types Table:

Table 14. Variable Scope

Scope	Description	Lifetime	Access
Local	Inside function	Function execution	Function only
Global	Outside functions	Program execution	Entire program
Built-in	Python keywords	Python session	Everywhere

Mnemonic

"Scope: Local Lives in Functions, Global Lives Everywhere"

Question Question 4(b) [04 marks]

Develop a program to create nested LOOP and display numbers.

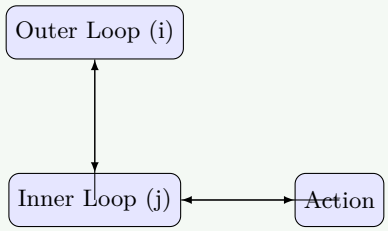
Solution

Nested Loop Program:

```python
# Example 1: Number grid
```

```
print("Number Grid Pattern:")
for i in range(1, 4):
    for j in range(1, 5):
        print(f"{i}{j}", end=" ")
    print()  # New line after each row
```

Nested Loop Structure:



Mnemonic

"Nested Loops: Outer Controls Inner"

Question Question 4(c) [07 marks]

Write a program to create a list of ODD and EVEN numbers in range of 1 to 50.

Solution

ODD and EVEN Numbers Program:

```
# Method 1: Using loops and conditions
odd_numbers = []
even_numbers = []

for i in range(1, 51):
    if i % 2 == 0:
        even_numbers.append(i)
    else:
        odd_numbers.append(i)

print("ODD Numbers (1-50):")
print(odd_numbers)
print(f"Count: {len(odd_numbers)}")

print("\nEVEN Numbers (1-50):")
print(even_numbers)
print(f"Count: {len(even_numbers)}")
```

Number Classification Table:

Table 15. Number Classification

Type	Condition	Range 1-10	Count (1-50)
ODD	`n % 2 != 0`	1,3,5,7,9	25
EVEN	`n % 2 == 0`	2,4,6,8,10	25

Mnemonic

"Odd/Even: Remainder 1/0 When Divided by 2"

Question Question 4(a OR) [03 marks]

Explain String Slicing with example.

Solution

String Slicing extracts parts of a string using `[start:stop:step]` syntax.
Slicing Syntax Table:

Table 16. Slicing Syntax

Syntax	Description	Example	Result
`s[start:stop]`	From start to stop-1	`"hello"[1:4]`	"ell"
`s[start:]`	From start to end	`"hello"[2:]`	"llo"
`s[:stop]`	From beginning to stop-1	`"hello"[:3]`	"hel"
`s[::step]`	Every step character	`"hello"[::2]`	"hlo"
`s[::-1]`	Reverse string	`"hello"[::-1]`	"olleh"

Mnemonic

"Slice: Start, Stop, Step"

Question Question 4(b OR) [04 marks]

Write a program using user defined function to find the factorial of a given number.

Solution

Factorial Function Program:

```python
def factorial(n):
    """Calculate factorial using recursion"""
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial(n - 1)

def factorial_iterative(n):
    """Calculate factorial using loop"""
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result

# Main program
number = int(input("Enter a number: "))
if number < 0:
    print("Factorial not defined for negative numbers")
else:
    result1 = factorial(number)
    print(f"Factorial of {number} = {result1}")
```

Mnemonic

"Factorial: Multiply All Numbers Below"

Question Question 4(c OR) [07 marks]

Write a user defined function to check whether a sub string is present in a given string.

Solution

Substring Check Function:

```
def find_substring(main_string, sub_string):
    """Check if substring exists in main string"""
    if sub_string in main_string:
        index = main_string.find(sub_string)
        return True, index
    else:
        return False, -1

# Main program
text = input("Enter main string: ")
search = input("Enter substring to search: ")

found, position = find_substring(text, search)
if found:
    print(f"Substring '{search}' found at position {position}")
else:
    print(f"Substring '{search}' not found")
```

String Methods Table:

Table 17. String Methods

Method	Purpose	Example	Result
`find()`	Find first position	`"hello".find("ll")`	2
`count()`	Count occurrences	`"hello".count("l")`	2
`in`	Check existence	`"ll" in "hello"`	True
`index()`	Find position (error if not found)	`"hello".index("e")`	1

Mnemonic

"Substring: Search, Find, Count, Position"

Question Question 5(a) [03 marks]

Explain how to create and access a List with example.

Solution

List Creation and Access:

```
# Creating lists
numbers = [1, 2, 3, 4, 5]

# Accessing elements
print(f"First element: {numbers[0]}")      # 1
print(f"Last element: {numbers[-1]}")      # 5
print(f"Slice: {numbers[1:4]}")           # [2, 3, 4]
```

List Access Methods:

Table 18. List Access

Method	Syntax	Example	Result
Index	`list[i]`	`[1,2,3][1]`	2
Negative	`list[-i]`	`[1,2,3][-1]`	3
Slice	`list[start:stop]`	`[1,2,3,4][1:3]`	[2,3]

Mnemonic

"Lists: Create, Index, Access"

Question Question 5(b) [04 marks]

List out the operations that can be performed on a LIST. Write a program to create and copy one List into another List.

Solution

List Operations and Copy Program:

```
# Original list
original = [1, 2, 3, 4, 5]
print(f"Original list: {original}")

# Copying methods
shallow_copy = original.copy()
slice_copy = original[:]
list_copy = list(original)

# Modify original
original.append(6)
print(f"After append: {original}")
print(f"Shallow copy: {shallow_copy}")
```

List Operations Table:

Table 19. List Operations

Operation	Method	Example	Result
Add	`append()`	`[1,2].append(3)`	[1,2,3]
Insert	`insert()`	`[1,3].insert(1,2)`	[1,2,3]
Remove	`remove()`	`[1,2,3].remove(2)`	[1,3]
Pop	`pop()`	`[1,2,3].pop()`	[1,2]

Mnemonic

"List Operations: Add, Insert, Remove, Pop, Copy"

Question Question 5(c) [07 marks]

List and give use of various Built in methods of LIST

Solution

Built-in List Methods:

```
# Sample list for demonstrations
fruits = ['apple', 'banana', 'cherry', 'apple']

# Modification methods
fruits.append('date')              # Add to end
fruits.insert(1, 'avocado')       # Insert at index
fruits.remove('apple')            # Remove first occurrence
last_fruit = fruits.pop()         # Remove and return last

# Search and count methods
count = fruits.count('apple')     # Count occurrences
index = fruits.index('banana')    # Find first index

# Sorting and reversing
fruits.sort()                     # Sort in place
fruits.reverse()                  # Reverse in place
```

List Methods Summary:

Table 20. List Methods

Category	Method	Purpose	Returns	Modifies Original
Add	`append(x)`	Add item to end	None	Yes
Add	`insert(i,x)`	Insert at position	None	Yes
Remove	`remove(x)`	Remove first x	None	Yes
Remove	`pop(i)`	Remove at index	Removed item	Yes
Search	`index(x)`	Find position	Index	No
Sort	`sort()`	Sort in place	None	Yes
Sort	`reverse()`	Reverse order	None	Yes
Copy	`copy()`	Shallow copy	New list	No

Mnemonic

"List Methods: Add, Remove, Search, Sort, Copy"

Question Question 5(a OR) [03 marks]

Explain how to create and traverse a string by giving an example.

Solution

String Creation and Traversal:

```
# String creation methods
string1 = "Hello World"        # Double quotes
string2 = 'Python'             # Single quotes

# String traversal methods
text = "Python"

# Method 1: Using for loop
for char in text:
    print(char, end=" ")
```

Traversal Methods Table:

Table 21. Traversal Methods

Method	Syntax	Use Case
Direct	`for char in string:`	Simple character access
Index	`for i in range(len(s)):`	Need position info
Enumerate	`for i, char in enumerate(s):`	Both index and character

Mnemonic

"Strings: Create, Loop, Access"

Question Question 5(b OR) [04 marks]

List out the operations that can be performed on a String. Write a code for any 2 operations

Solution

String Operations:

```
# Operation 1: String concatenation
first_name = "John"
last_name = "Doe"
full_name = first_name + " " + last_name
print(f"Concatenation: {full_name}")

# Operation 2: String case conversion
sentence = "learn python"
title_case = sentence.title()
print(f"Title case: {title_case}")
```

String Operations Table:

Table 22. String Operations

Category	Operation	Example	Result
Join	Concatenation	`"Hello" + " World"`	"Hello World"
Case	`upper()`	`"hello".upper()`	"HELLO"
Case	`lower()`	`"HELLO".lower()`	"hello"
Split	`split()`	`"a,b".split(",")`	['a','b']
Replace	`replace()`	`"hi".replace("i","o")`	"ho"

Mnemonic

"String Operations: Join, Case, Split, Find"

Question Question 5(c OR) [07 marks]

List and give use of various built – in methods of String.

Solution

Built-in String Methods:

```
# Sample string for demonstration
text = "  Python Programming  "

# Case conversion methods
print(f"upper(): {text.upper()}")
print(f"lower(): {text.lower()}")

# Whitespace methods
print(f"strip(): '{text.strip()}'")

# Search and check methods
print(f"find('Python'): {text.find('Python')}")
print(f"count('P'): {text.count('P')}")

# Split and join methods
words = text.split()
print(f"split(): {words}")
```

String Methods Classification:

Table 23. String Methods Class

Category	Methods	Purpose	Example
Case	`upper(), lower()`	Change case	`"hi".upper()` $\rightarrow$ "HI"
Whitespace	`strip()`	Remove spaces	`" h ".strip()` $\rightarrow$ "h"
Search	`find(), count()`	Find substrings	`"hi".find("i")` $\rightarrow 1$
Check	`startswith()`	Test string ends	`"hi".startswith("h")` $\rightarrow True$
Type Check	`isdigit()`	Character types	`"1".isdigit()` $\rightarrow True$
Replace	`replace()`	Substitute text	`"sub".replace("u","o")` $\rightarrow$ "sob"

Mnemonic

"String Methods: Case, Clean, Check, Change"