Question Number - 1
Max Marks - 3
Answer/Marking Scheme:

Sample Answer:

```
class Pair:
    def __init__(self, int_value, str_value):
        self.o1 = int_value  # Integer member variable
        self.o2 = str_value  # String member variable

# Create an object p1 with the given values
p1 = Pair(7, "Okay!")

# (Optional) Print or test to confirm values
print(p1.o1)  # 7
print(p1.o2)  # "Okay!"
```

Marking Scheme:

Class Definition (1 point): Correctly naming the class `Pair` and creating it in Python.

Constructor Definition (1 point): Providing an `__init__` method that accepts two parameters (for integer and string).Correctly assigning these parameters to instance variables (e.g., `self.o1`, `self.o2`).

Object Instantiation (1 point): Creating an object `p1` of the `Pair` class with the given values (`7` and `"Okay!"`).Correctly demonstrating that `p1.o1` holds the integer `7` and `p1.o2` holds the string `"Okay!"`.

Question Number - 2
Max Marks - 3
Answer/Marking Scheme:

Part a)
Sample Answer:
Assume an **empty** BST at the start.
1. Insert 7

```
 7
```

7 becomes the root (since the tree was empty).
2. *Insert 5*

```
 7
/
5
```

Compare 5 to 7: 5 < 7 → goes to the left of 7.
3. *Insert 9*

```
  7
 / \
5   9
```

Compare 9 to 7: 9 > 7 → goes to the right of 7.
4. *Insert 2*

```
  7
 / \
5   9
/
2
```

Compare 2 to 7: 2 < 7 → go left to 5.

Compare 2 to 5: 2 < 5 → go left to (empty) ⇒ place 2 there.

5. *Insert 4*

```
  7
 / \
5   9
/
2
 \
  4
```

Compare 4 to 7: 4 < 7 → go left to 5.

Compare 4 to 5: 4 < 5 → go left to 2.

Compare 4 to 2: 4 > 2 → go right to (empty) ⇒ place 4 there

6. Insert 11

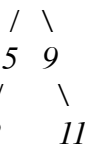
```
    7
   / \
  5   9
 /     \
2       11
 \
```

4

Compare 11 to 7: 11 > 7 → go right to 9.

Compare 11 to 9: 11 > 9 → go right to (empty) ⇒ place 11 there.

7. *Delete 4*



Node 4 is the right child of 2 and is a leaf, so it is removed directly.

That is the **final BST** after all operations.

Marking Scheme:

Correct Insertion Steps (1 point): Showing the tree structure after each insertion: 7, 5, 9, 2, 4, 11 in the right places.

Correct Deletion Step (1 point): Demonstrating the removal of 4 accurately (removing it from its parent 2).

Part b)

Sample Answer:

"balanced" means "height-balanced" (as in AVL): The heights of the subtrees differ by at most 1 for every node, so **"Yes"** it is height-balanced.

Marking Scheme:

Answer to "Is the final tree balanced?" (1 point): Yes

Question Number - 3
Max Marks - 4
Answer/Marking Scheme:

AWARD ZERO.

Question Number - 4
Max Marks - 4
Answer/Marking Scheme:

Sample Answer:
Part a) Contains

1. Sorted array: O(logn)
2. Singly linked list: O(n)

Part b) Insert

1. Unsorted array: O(n)
2. Sorted singly linked list: O(n)

Marking Scheme:
Each correct complexity is worth **1 point**.

Question Number - 5
Max Marks - 5
Answer/Marking Scheme:

Sample Answer:
(a) Option (b)

(b) Option (b)

(c) Option (a)

(d) Option (b)

(e) Option (b)

Marking Scheme:
Each correct option chosen is worth **1 point**.

Question Number - 6
Max Marks - 2
Answer/Marking Scheme:

Sample Answer:
Stacks

Depth-First Search (DFS)/Backtracking: When doing a DFS (e.g., in graph traversal), you push nodes onto a stack and pop them as you explore.

Function Call Stack (Recursion): Programming languages use an internal call stack to track function calls, return addresses, and local variables.

(Other valid examples include undo operations in text editors, parsing expressions, etc.)

Queues

Breadth-First Search (BFS) in Graphs: Nodes/vertices are enqueued as they are discovered and dequeued in the order they were visited.

Scheduling / Buffering: OS job scheduling, printer spooling, and CPU task queues typically use queues to process tasks in FIFO (first-in-first-out) order.

(Other valid examples include handling requests on web servers, event queue systems, etc.)

Marking Scheme:
Two applications for Stacks (1 point)

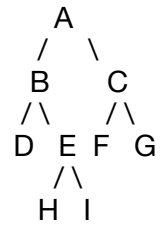
- 0.5 points per correct/appropriate application.
- Examples: DFS/backtracking, function-call/recursion stack, undo functionality, etc.

Two applications for Queues (1 point)

- 0.5 points per correct/appropriate application.
- Examples: BFS, scheduling/printing spooling, CPU task queues, etc.

Question Number - 7
Max Marks - 4
Answer/Marking Scheme:

Sample Answer:



Marking Scheme:
Identify the root from preorder (1 point): Correctly picking `A` as the root and splitting the inorder sequence.

Construct the left subtree correctly (1 point): Correctly dividing preorder (`B D E H I`) & inorder (`D B H E I`) and showing the subtree structure for `B-D-E-H-I`.

Construct the right subtree correctly (1 point): Correctly dividing preorder (`C F G`) & inorder (`F C G`) and showing the subtree structure for `C-F-G`.

Final Tree (1 point): Final structure (with all correct placements) matching the given traversals.

Question Number - 8
Max Marks - 4
Answer/Marking Scheme:

AWARD ZERO.

Question Number - 9
Max Marks - 3
Answer/Marking Scheme:

Sample Answer:

```
function SumLeaf(root):
    # 1. Base Case: If the tree (or subtree) is empty
    if root == null:
        return 0

    # 2. If the node is a leaf (no children)
    if root.left == null and root.right == null:
        return root.value

    # 3. Otherwise, recurse on the left and right subtrees
    leftSum = SumLeaf(root.left)
    rightSum = SumLeaf(root.right)

    return leftSum + rightSum
```

Marking Scheme:

Base Case Handling (1 point): Correctly returns 0 if `root == null`.

Leaf Check (1 point): Correctly identifies a leaf node (`root.left == null` and `root.right == null`) and returns the node's value.

Recursive Summation (1 point): Correctly returns SumLeaf(root.left) + SumLeaf(root.right) for non-leaf nodes.

Question Number - 10
Max Marks - 3
Answer/Marking Scheme:

Sample Answer:

Part a) Converting to Postfix Notation

Expression: 5+3×(2+4)

Step-by-Step Conversion to Postfix:

1. Identify sub-expression in parentheses:(2+4): Postfix of 2+4 is 24+.
2. Multiply that result by 3: So 3×(2+4) in postfix becomes 324+×.
3. Finally, add 5 to that product: The entire expression 5+[3×(2+4)] in postfix is: 5324+×+

Thus, the **postfix** (also called **Reverse Polish notation**) for **5+3×(2+4) is: 5324+∗+**

Part b) Evaluating the Postfix Expression with a Stack

Postfix to evaluate: 5324+∗+

We process tokens **from left to right**:

Token	Action	Stack After
5	**Operand** → push onto stack	[5]
3	**Operand** → push onto stack	[5, 3]
2	**Operand** → push onto stack	[5, 3, 2]
4	**Operand** → push onto stack	[5, 3, 2, 4]
+	**Operator** → pop top 2 (4, 2), compute 2+4=6, push 6	[5, 3, 6]
*	**Operator** → pop top 2 (6, 3), compute 3*6=18, push 18	[5, 18]
+	**Operator** → pop top 2 (18, 5), compute 5+18=23, push 23	[23]

- **Final stack** after reading all tokens is [23]
- The single value **23** is the result of the expression 5+3×(2+4).

Marking Scheme:

Postfix Conversion (1 point): Correctly writing 5321+∗+.

Stack Evaluation:

Push/Pop Steps (1 point)**:** Demonstrating the correct pushing of operands (5, 3, 2, 4). Correctly popping the top two for each operator (+, *, +).

Final Result (1 point)**:** Arriving at the correct final answer of 23.

Question Number - 11
Max Marks - 4
Answer/Marking Scheme:

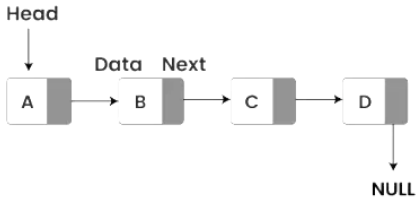
Sample Answer:

Part a) A linked list is a linear data structure where each element (called a node) contains:

1. A **value** (data).

2. A **reference** (or pointer) to the **next** node in the list (in a singly linked list).

Unlike an array, a linked list does **not** store its nodes in contiguous memory locations. Instead, each node can be stored **anywhere** in memory, and nodes are connected through references/pointers.



Part b) One Advantage and One Disadvantage of Linked Lists Over Arrays

Advantage:
Dynamic Size and Easier Insertions/Deletions – Linked lists can grow or shrink at runtime without needing contiguous memory. Also, inserting or deleting a node can be O(1) if you already have the pointer to the node (or if you are inserting at the head).

Disadvantage:
No Random Access – Accessing an element by index requires traversing from the start (or a known node) until you reach that element, resulting in O(n) time complexity. Arrays support direct indexing in O(1).

Marking Scheme:

Definition and Diagram (1 point): Clear explanation of linked-list structure (nodes with pointers). Simple diagram showing nodes connected by references.

Advantage and Disadvantage (1 point): One valid advantage (e.g., dynamic size and simpler insertions/deletions). One valid disadvantage (e.g., no random access, thus slower access time).

Question Number - 12
Max Marks - 4
Answer/Marking Scheme:

AWARD ZERO.

Question Number - 13
Max Marks - 3
Answer/Marking Scheme:

Sample Answer:

Part a) Difference Between Mutable and Immutable Data Types in Python

Mutable data types **can be** modified in place without creating a new object.Common examples: **lists, dictionaries, sets.**

Immutable data types **cannot** be changed in place; any "change" creates a new object with a new memory address.Common examples: **integers**, **strings**, **tuples**.

Example Code Demonstration

1. List (Mutable):

Code Snippet:-

```
my_list = [1, 2, 3]

print("Before append:", my_list, "| Memory Address:", hex(id(my_list)))



my_list.append(4)

print("After append:", my_list, "| Memory Address:", hex(id(my_list)))
```

Output might look like:
Before append: [1, 2, 3] | Memory Address: 0x109b5b080
After append: [1, 2, 3, 4] | Memory Address: 0x109b5b080
Notice the **same** memory address (ID) before and after, indicating the list was changed **in place**.

2. Tuple (Immutable):

Code Snippet:-

```
my_tuple = (1, 2, 3)

print("Original tuple:", my_tuple, "| Memory Address:", hex(id(my_tuple)))



# Attempt to 'add' an element creates a NEW tuple

my_tuple += (4,)

print("Modified tuple:", my_tuple, "| Memory Address:", hex(id(my_tuple)))
```

Output might look like:
Original tuple: (1, 2, 3) | Memory Address: 0x10c3e3aa0
Modified tuple: (1, 2, 3, 4) | Memory Address: 0x10c3e3b60
The **memory address changes**, showing that a new tuple object has been created.

Part b) How Mutability of Function Arguments Affects Function Behavior

In Python, **function arguments** are passed by object reference (sometimes colloquially described as "pass by assignment"). If the argument is **mutable**, the function can directly modify the same object. If the argument is **immutable**, any attempt to change it in the function results in a new object being created—leaving the original unchanged.

Marking Scheme:

Mutable vs. Immutable (2 points)

1. Definition and Examples (1 point):States what mutable vs. immutable means.
2. Code/Illustration (1 point): Demonstrates how changes in mutable objects do not alter the memory address, while immutable changes create new objects.

Effect on Function Arguments (1 points)

Explanation of Pass-by-Object-Reference (1 point): Correctly showing how a mutable object can be changed in place, while an immutable object remains unchanged outside the function.

Question Number - 14
Max Marks - 4
Answer/Marking Scheme:

Sample Answer:

Part a)

Public Members

- **Naming Convention**: no underscores (e.g., `myvar`)
- **Meaning**: Accessible from anywhere (inside or outside the class).
- **Usage**:

```
class MyClass:
    def __init__(self, val):
        self.value = val  # Public member
obj = MyClass(10)
print(obj.value)  # Accessible directly
```

Protected Members

- **Naming Convention**: single leading underscore (`_myvar`)
- **Meaning**: By convention, it's meant for "internal use" within the class or its subclasses. It is **not** strictly enforced, but the underscore signals to other developers that "this is semi-private; do not access unless you really know what you're doing."
- **Usage:**

```
class Parent:
    def __init__(self, val):
        self._protected = val
```

```
class Child(Parent):
    def show_protected(self):
        print("Protected value:", self._protected)

c = Child(42)
c.show_protected()        # Allowed, but it's by convention only
print(c._protected)       # Possible, but not recommended
```

Private Members

- **Naming Convention**: **double** leading underscores (`__myvar`)
- **Meaning**: Triggers **name mangling** in Python; the interpreter changes the attribute name internally, making it harder to access from outside. It is still possible to access (using the mangled name), but indicates stronger intent that it's private to the class implementation.
- **Usage**:

```
class Demo:
    def __init__(self, val):
        self.__private = val   # Name-mangled to _Demo__private internally

    def get_private(self):
        return self.__private

d = Demo(100)
print(d.get_private())        # 100
# print(d.__private)  # AttributeError if you do this directly

# But can access with: d._Demo__private (not recommended)
```

Part b)

Inheritance is a mechanism allowing a class (called the **child class** or **subclass**) to **reuse** and **extend** the attributes and behaviors of another class (called the **parent class** or **superclass**).
Two Advantages of Inheritance:

Code Reusability: Common functionality can reside in a parent class so that child classes can inherit and reuse it without duplication**.**

Improved Organization and Maintenance: Related classes can share a logical hierarchy, making the code more organized and easier to maintain.

(Other advantages could include polymorphism, easier extension, etc.)

Marking Scheme:

Access Modifiers in Python (2 points)

Public: Explanation, example code (0.66 points)

Protected: Explanation, example code (0.66 points)

Private: Explanation, example code(0.66 points)

Inheritance in Python (2 points)

Definition: Explains that a subclass inherits properties/methods from a superclass (1 point)

Two Advantages: Lists and briefly explains at least two (0.5 point each)