Unit 04: Computational Structures

Exercise Short Answer Questions.

1. **Explain how the ' extend() ' function works in python lists. Provide an example.**
   The extend() function in Python is used to add multiple elements from another iterable, such as a list, tuple, or string, to the end of the current list. Unlike the append() method, which adds the entire iterable as a single element, extend() takes each item from the iterable and adds it individually to the list. This is useful when you want to combine two lists into one without creating nested lists. For example, if you have a list a = [1, 2, 3] and you call a.extend([4, 5]), the list a will become [1, 2, 3, 4, 5]. This function modifies the original list in place and does not return a new list.

2. **Explain the potential issues which could arise when two variables reference the same list in a program? Provide an example.**
   If two variables point to the same list, changes in one will affect the other—because both refer to the same memory.

   Example:
   a = [1, 2, 3]
   b = a
   b.append(4)
   print(a) # Output: [1, 2, 3, 4]
   Problem: You might accidentally change data in both places.
   Use copy() to avoid this:
   b = a.copy()

3. **Define a stack and explain the Last-In, First-Out (LIFO) principle.**
   A stack is a simple data structure where you can only add or remove items from one end, known as the "top". Both insertion and deletion of elements occur at this top end. A stack operates on the Last-In, First-Out (LIFO) principle, meaning that the most recently added element is the first one to be removed.
   LIFO Principle: A stack works by the Last-In, First-Out rule. The last item added is the first one removed.
   • Single Access Point: You can only add or remove items from the top of the stack.
   • Order Preservation: The order in which you add items is preserved, with the last item added being the first to leave.
   • Size Flexibility: A stack can either have a fixed size or grow as needed. Sequential Access: You can only access items in the order they were added to, starting from the top.

4. **How does the stack help in balancing parentheses in an expression? Describe the process.**
   When dealing with infix expressions, parentheses are used to show which operations should be done first. Balancing parentheses is important to ensure that every opening parenthesis has a matching closing parenthesis.
   Steps to Check Parentheses:
   1. Use a stack to keep track of opening parentheses.

2. When you find an opening parenthesis (, push it onto the stack.
3. When you find a closing parenthesis), pop the top of the stack.
4. If the stack is empty after processing all parentheses, they are balanced.
5. If the stack is not empty, or if you try to pop from an empty stack, the parentheses are not balanced.

Example: For the expression (3 + 4) * (5 - 2), the parentheses are balanced. Each (has a matching).

5. **Differentiate between the Enqueue and Dequeue operations of queue.**

Enqueue (Add an Item): This is like adding a person to the end of the line. In a queue, you add items to the back.

Example:

queue.append(10)

Dequeue (Remove an Item): This is like serving the person at the front of the line. In a queue, you take items out from the front.

Additional operations might include checking if the queue is empty, retrieving the element at the front without removing it, and determining the size of the queue.

Example:

queue.pop(0)

6. **Explain how the concept of a queue is applied in a computer job scheduling system?**

In job scheduling:

Jobs are queued in order of arrival.

The first job added is the first to be processed (FIFO).

Example: Print queue

- You click "Print" on several files
- They are printed in order, one by one.

7. **How can lists be utilized to implement other data structures?**

Python lists are versatile and can be used to implement a variety of other data structures due to their dynamic nature and built-in methods.

For example, a **stack** can be implemented using a list by using the append() method to push elements and the pop() method to remove the last element, following the Last-In, First-Out (LIFO) principle.

Similarly, a **queue** can be implemented using append() to add elements at the end and pop(0) to remove elements from the front, following the First-In, First-Out (FIFO) principle.

Lists can also be used to represent **2D arrays or matrices** using nested lists, where each inner list acts as a row. With the help of custom logic or classes, lists can even simulate more complex structures like **linked lists, trees, and graphs**, making them a foundational tool in Python for learning and building data structures.

8. **Differentiate between the Depth-First Search (DFS) and Breadth-First Search (BFS).**

Feature	DFS	BFS

Search Type	Goes **deep** into each path first	Explores **level by level**
Data Structure Used	Stack (or recursion)	Queue
Memory Usage	Less (usually)	More
Best For	Solving puzzles, maze-like problems	Finding shortest paths

Example:

- DFS is like exploring a cave deep before going sideways.
- BFS is like searching all rooms on one floor before going upstairs.