CHAPTER -9

Lists

2MARK QUESTIONS

1Q: Define a list in the context of computer science.

Answer:

A list is a data structure that stores an ordered collection of elements, allowing for easy access, modification, and manipulation of data.

2Q: Differentiate between an array and a list.

Answer:

An array is a fixed-size collection of elements of the same data type, while a list is a dynamic collection that can grow or shrink, and its elements can be of different data types.

3Q: Explain the concept of indexing in a list.

Answer:

Indexing in a list refers to the position of an element within the list. It usually starts from 0, with the first element at index 0, the second at index 1, and so on.

4Q: How is memory allocation managed in lists compared to arrays?

Answer:

Lists dynamically manage memory allocation, automatically adjusting their size as elements are added or removed. In contrast, arrays have a fixed size that needs to be defined during initialization.

5Q: Discuss the significance of the append() method in Python lists.

Answer:

The append() method is used to add an element to the end of a Python list. It is crucial for dynamically expanding lists.

6Q: Define the term "list comprehension" in Python.

Answer:

List comprehension is a concise way to create lists in Python by specifying the elements and conditions in a single line, using a compact syntax.

7Q: Explain the difference between the insert() and append() methods in Python lists.

Answer:

The insert() method is used to add an element at a specific index in a Python list, while append() adds an element to the end of the list.

8Q: How is a list different from a set?

Answer:

A list is an ordered collection with duplicate elements allowed, while a set is an unordered collection of unique elements.

9Q: Illustrate the concept of slicing in Python lists.

Answer:

Slicing in Python lists involves extracting a portion of the list using the syntax list[start:stop]. It creates a new list containing elements from index start to stop-1.

10Q: Discuss the role of the remove() method in Python lists.

Answer:

The remove() method in Python lists is used to remove the first occurrence of a specified element. If the element is not found, it raises a ValueError.

4MARK QUESTIONS

1Q: Explain the concept of dynamic memory allocation in Python lists.

Answer:

In Python, lists dynamically manage memory allocation, allowing them to resize as elements are added or removed. The list class automatically handles memory allocation, ensuring that users do not need to explicitly manage memory. This dynamic feature makes Python lists versatile for various applications.

2Q: Compare and contrast the extend() and append() methods in Python lists.

Answer:

Both extend() and append() methods are used to add elements to Python lists, but they behave differently. The append() method adds a single element at the end of the list, while the extend() method appends the elements of an iterable (e.g., another list) to the end of the list. This key distinction makes extend() suitable for adding multiple elements.

3Q: Discuss the role of list comprehension in Python and provide an example.

Answer:

List comprehension is a concise way to create lists in Python. It allows the definition of lists in a single line using a compact syntax. For example:

```
squares = [x**2 for x in range(1, 6)]
```

This code creates a list squares containing the squares of numbers from 1 to 5.

4Q: Explain the concept of shallow copy and deep copy in the context of Python lists.

Answer:

In Python, a shallow copy creates a new list, but instead of copying the elements, it copies references to the original objects. This means that changes to nested objects are reflected in both the original and copied lists. On the other hand, a deep copy creates a new list and recursively copies all nested objects, resulting in independent copies. The copy module provides functions copy() for shallow copy and deepcopy() for deep copy.

5Q: How does the pop() method work in Python lists? Provide an example.
Answer:

The pop() method removes and returns the last element from a Python list. If an index is provided, it removes and returns the element at that index. Example:

numbers = [1, 2, 3, 4, 5]

popped_element = numbers.pop()

print("Popped Element:", popped_element)

6Q: Differentiate between the remove() and del statements in Python lists.

Answer:

The remove() method removes the first occurrence of a specified element from the list, raising a ValueError if the element is not found. The del statement, on the other hand, deletes elements or slices from a list using indices. It can also delete the entire list or variables. Unlike remove(), del does not raise an error if the element is not found.

7Q: Explain the concept of list aliasing and its potential implications.
Answer:

List aliasing occurs when two or more variables refer to the same list object. Modifying the list through one variable will affect the other variables pointing to the same list. This can lead to unintended side effects and bugs if not handled carefully. To avoid aliasing, consider creating a copy of the list using the copy() method or slicing.

8Q: Discuss the purpose of the count() method in Python lists. Provide an example.

Answer:

The count() method in Python lists is used to count the number of occurrences of a specific element in the list. Example:

numbers = [1, 2, 2, 3, 4, 2, 5]

count_of_twos = numbers.count(2)

print("Count of Twos:", count_of_twos)

9Q: How does the reverse() method work in Python lists?

Answer:

The reverse() method in Python lists reverses the order of elements in the list in-place. It modifies the original list without creating a new one. Example:

```
numbers = [1, 2, 3, 4, 5]

numbers.reverse()

print("Reversed List:", numbers)
```

10Q: Explain the purpose of the index() method in Python lists. Provide an example.

Answer:

The `index()` method is used to find the index of the first occurrence of a specified element in a Python list. Example:

```python
fruits = ["apple", "orange", "banana", "apple"]

index_of_apple = fruits.index("apple")

print("Index of 'apple':", index_of_apple)
```

7MARK QUESTIONS

1Q: Explain the concept of a linked list and how it differs from an array.
Answer:

A linked list is a data structure consisting of nodes, where each node points to the next node in the sequence. Unlike arrays, linked lists do not require contiguous memory locations, allowing dynamic allocation and easier insertion or deletion of elements. Arrays have fixed sizes, and elements are stored in adjacent memory locations.

2Q: Discuss the advantages and disadvantages of using arrays over linked lists and vice versa.

Answer:

Advantages of Arrays:

Constant time access to elements using indexing.

Efficient memory usage for fixed-size collections.

Disadvantages of Arrays:

Fixed size, requiring additional memory management for resizing.

Inefficient for insertions and deletions, especially in the middle.

Advantages of Linked Lists:

Dynamic size, allowing easy addition or removal of elements.

Efficient for insertions and deletions at any position.

Disadvantages of Linked Lists:

Sequential access required for searching.

Additional memory overhead due to storing pointers.

3Q: Write a Python program to implement a stack using a list and explain how the stack operations work.

Answer:

```
class Stack:
    def __init__(self):
        self.items = []
def push(self, item):
        self.items.append(item)
  def pop(self):
        return self.items.pop()
def peek(self):
        return self.items[-1] if self.items else None
 def is_empty(self):
        return not bool(self.items)
# Example usage
stack = Stack()
stack.push(1)
stack.push(2)
stack.push(3)
print(stack.pop())  # Output: 3
print(stack.peek()) # Output: 2
```

4Q: Elaborate on the concept of a doubly linked list and its advantages over a singly linked list.

Answer:

A doubly linked list is a linked list in which each node contains a reference to both the next and the previous node. This bidirectional linking allows traversal in both directions, enhancing flexibility. Advantages include efficient backward traversal and easier removal of nodes without iterating from the beginning.

5 Q: Describe the process of merging two sorted lists into a single sorted list. Provide a Python implementation.

Answer:

```
def merge_sorted_lists(list1, list2):
    merged_list = []
    i = j = 0
while i < len(list1) and j < len(list2):
        if list1[i] < list2[j]:
            merged_list.append(list1[i])
            i += 1
        else:
            merged_list.append(list2[j])
            j += 1

    merged_list.extend(list1[i:])
    merged_list.extend(list2[j:])
 return merged_list
# Example usage
list1 = [1, 3, 5]
list2 = [2, 4, 6]
result = merge_sorted_lists(list1, list2)
print(result)  # Output: [1, 2, 3, 4, 5, 6]
```

6Q: Discuss the time and space complexity of common list operations such as insertion, deletion, and searching.

Answer:

Insertion: Time complexity depends on the position. For an array, it's O(n) for middle insertions, while linked lists can be O(1) for head insertions.

Deletion: Similar to insertion, O(n) for arrays and O(1) for linked lists at the head.

Searching: O(n) for both arrays and linked lists, as a sequential search may be required.

7 Q: Write a Python program to implement a queue using a list and explain how the queue operations work.

Answer:

```
class Queue:
    def __init__(self):
        self.items = []
def enqueue(self, item):
        self.items.append(item)
 def dequeue(self):
        return self.items.pop(0) if self.items else None
  def front(self):
        return self.items[0] if self.items else None
def is_empty(self):
        return not bool(self.items)
# Example usage
queue = Queue()
queue.enqueue(1)
queue.enqueue(2)
queue.enqueue(3)
print(queue.dequeue())  # Output: 1
print(queue.front())     # Output: 2
```

8Q: Explain the concept of a circular linked list and provide a scenario where it might be advantageous.

Answer:

A circular linked list is a linked list where the last node points back to the first node, forming a closed loop. It can be advantageous in scenarios where continuous processing or looping through elements is required, such as in a round-robin scheduling algorithm.

9Q: Discuss the role of the sort() method in Python lists and its time complexity.

Answer:

The sort() method in Python lists is used to sort the elements in ascending order. It has a time complexity of O(n log n) for average and worst-case scenarios. The method modifies the original list in-place.

10Q: Compare and contrast the list and array data structures in terms of their applications and limitations.

Answer:

List:

Applications: Dynamic collections, easy manipulation, suitable for various data types.

Limitations: Variable sizes may impact performance, not optimized for mathematical operations.

Array:

Applications: Fixed-size collections, efficient for mathematical operations, contiguous memory access.

Limitations: Fixed size, additional memory management for resizing, less flexible than lists.

.

Multiple-Choice Questions (MCQs):

1Q: What is the key feature that distinguishes a linked list from an array?

A. Contiguous memory allocation

B. Fixed size

C. Dynamic size and memory allocation

D. Bidirectional linking

Answer: C

2Q: What is the time complexity for searching an element in an unsorted array or linked list?

A. O(1)

B. O(log n)

C. O(n)

D. O(n^2)

Answer: C

3Q: In Python, which method is used to add an element to the end of a list?

A. add()

B. insert()

C. append()

D. extend()

Answer: C

4Q: What does the pop() method do in Python lists?

A. Adds an element to the list

B. Removes and returns the last element

C. Sorts the list

D. Reverses the list

Answer: B

5Q: In a doubly linked list, each node contains references to both the ___________.

A. Next node and previous node

B. Next node and parent node

C. Previous node and child node

D. Sibling nodes

Answer: A

6Q: What is the time complexity of the sort() method in Python lists?

A. O(n)

B. O(n log n)

C. O(n^2)

D. O(log n)

Answer: B

7Q: Which type of list allows efficient insertion and deletion at both ends?

A. Array

B. Linked List

C. Circular Linked List

D. Doubly Linked List

Answer: D

8Q: The index() method in Python lists is used for _____________.

A. Sorting the list

B. Counting occurrences of an element

C. Finding the index of a specific element

D. Reversing the list

Answer: C

9Q: What is the primary advantage of using a circular linked list?

A. Efficient random access

B. Efficient sequential access

C. Continuous processing

D. Fixed size

Answer: C

10Q: Which data structure is more suitable for mathematical operations and fixed-size collections?

A. List

B. Array

C. Linked List

D. Queue

Answer: B

Fill in the Blanks :

Q1: The remove() method in Python lists is used to remove the ________ occurrence of a specified element.

Answer: first

Q2: The is_empty() method returns True if the list is ________.

Answer: empty

Q3: The extend() method in Python lists is used to append the elements of an ________ to the end of the list.

Answer: iterable

Q4: The front() method in a queue implemented using a list returns the ________ element.

Answer: first

Q5: A doubly linked list allows for efficient traversal in ________ directions.

Answer: both

Q6: The del statement in Python lists can be used to delete elements by ________.

Answer: index

Q7: In a circular linked list, the last node points back to the ________ node.

Answer: first

Q8: The time complexity for searching an element in a sorted array or linked list is O(________).

Answer: log n

Q9: The count() method in Python lists is used to count the number of occurrences of a specific ________.

Answer: element

10Q: The peek() method in a stack implemented using a list returns the ________ element.

Answer: top