**AI ASSISTED CODING**

**LAB ASSIGNMENT – 11.4**

**NAME : N.KEERTHIREDDY**

**HALL NO. :** 2403A52376

BATCH : AI 14

TASK -1

CODE:

class StackWithList:

    """A stack implementation using a Python list."""

    def \_\_init\_\_(self):

        """Initializes an empty stack."""

        self.\_items = []

    def push(self, item):

        """Pushes an item onto the stack.

        Args:

            item: The item to be added to the stack.

        """

        self.\_items.append(item)

    def pop(self):

        """Removes and returns the top item of the stack.

        Returns:

            The last pushed item from the stack.

        Raises:

            IndexError: If the stack is empty.

        """

        if self.is\_empty():

            raise IndexError("pop from empty stack")

        return self.\_items.pop() # pop() on a list removes and returns the last element

    def peek(self):

        """Returns the top item of the stack without removing it.

        Returns:

            The last pushed item on the stack.

        Raises:

            IndexError: If the stack is empty.

        """

        if self.is\_empty():

            raise IndexError("peek from empty stack")

        return self.\_items[-1] # Access the last element

    def is\_empty(self):

        """Checks whether the stack is empty.

        Returns:

            bool: True if the stack is empty, False otherwise.

        """

        return not self.\_items

# Example usage with the list implementation:

my\_list\_stack = StackWithList()

my\_list\_stack.push(1)

my\_list\_stack.push(2)

print(f"List stack peek: {my\_list\_stack.peek()}")

print(f"List stack pop: {my\_list\_stack.pop()}")

print(f"List stack empty: {my\_list\_stack.is\_empty()}")

OUTPUT :

A black text on a white background

AI-generated content may be incorrect.

EXPLANATION :

This cell contains the implementation of the Stack class using collections.deque and test code to demonstrate its functionality.

* The Stack class uses a deque object (self.\_items) to store the stack elements. deque is preferred over a standard list for stack operations because appending and popping from either end are both O(1) operations, making it efficient for both push and pop.
* \_\_init\_\_(self): Initializes an empty deque.
* push(self, item): Adds an item to the right side of the deque, effectively placing it on top of the stack.
* pop(self): Removes and returns the item from the right side of the deque (the top of the stack). It raises an IndexError if the stack is empty.
* peek(self): Returns the item at the right side of the deque (the top of the stack) without removing it. It also raises an IndexError if the stack is empty.
* is\_empty(self): Checks if the deque is empty.

The test code then demonstrates these methods:

* It creates an empty Stack.
* Checks if it's empty.
* Pushes three items (10, 20, 30) onto the stack.
* Checks if it's empty again.
* Uses peek to see the top item (which is 30).
* Uses pop three times to remove and print each item, showing how the stack behaves (Last-In, First-Out).
* Checks if the stack is empty after popping all items.
* Includes try...except blocks to show how the IndexError is raised when trying to pop or peek from an empty stack.

**Cell 3fc4664a**:

This cell provides an alternative implementation of the Stack class using a standard Python list.

* The StackWithList class uses a Python list (self.\_items) to store the stack elements.
* \_\_init\_\_(self): Initializes an empty list.
* push(self, item): Uses append() to add an item to the end of the list, placing it on top of the stack. This is an O(1) operation.
* pop(self): Uses pop() without an index to remove and return the last element of the list (the top of the stack). This is also an O(1) operation. It raises an IndexError if the stack is empty.
* peek(self): Accesses the last element of the list (self.\_items[-1]) to return the top item without removing it. It raises an IndexError if the stack is empty.
* is\_empty(self): Checks if the list is empty.

# TASK – 2

CODE :

class QueueWithList:

    """A Queue implementation using a Python list."""

    def \_\_init\_\_(self):

        """Initializes an empty queue."""

        self.\_items = []

    def enqueue(self, item):

        """Adds an item to the rear of the queue.

        Args:

            item: The item to be added to the queue.

        """

        self.\_items.append(item)  # Add to the end of the list (rear of the queue)

    def dequeue(self):

        """Removes and returns the item from the front of the queue.

        Returns:

            The first added item from the queue.

        Raises:

            IndexError: If the queue is empty.

        """

        if self.is\_empty():

            raise IndexError("dequeue from empty queue")

        return self.\_items.pop(0)  # Remove from the beginning of the list (front of the queue)

    def is\_empty(self):

        """Checks whether the queue is empty.

        Returns:

            bool: True if the queue is empty, False otherwise.

        """

        return not self.\_items

# Example usage with the list implementation:

my\_list\_queue = QueueWithList()

print(f"Is the list queue empty? {my\_list\_queue.is\_empty()}")

my\_list\_queue.enqueue(10)

my\_list\_queue.enqueue(20)

my\_list\_queue.enqueue(30)

print(f"\nAfter enqueuing 10, 20, 30:")

print(f"Is the list queue empty? {my\_list\_queue.is\_empty()}")

print(f"Dequeued item: {my\_list\_queue.dequeue()}")

print(f"Dequeued item: {my\_list\_queue.dequeue()}")

print(f"\nAfter dequeuing two items:")

print(f"Is the list queue empty? {my\_list\_queue.is\_empty()}")

print(f"Dequeued item: {my\_list\_queue.dequeue()}")

print(f"\nAfter dequeuing all items:")

print(f"Is the list queue empty? {my\_list\_queue.is\_empty()}")

# Attempt to dequeue from an empty queue

try:

    my\_list\_queue.dequeue()

except IndexError as e:

    print(f"\nError when dequeuing from empty queue: {e}")

OUTPUT :

A white text box with black text

AI-generated content may be incorrect.

EXPLANATION :

This cell contains the implementation of a Queue data structure using a standard Python list and example code to show how it works.

* The QueueWithList class uses a Python list (self.\_items) to store the queue elements.
* \_\_init\_\_(self): Initializes an empty list.
* enqueue(self, item): This method adds an item to the **rear** of the queue. In this list-based implementation, we achieve this by using append(item), which adds the item to the end of the list. This is generally an efficient O(1) operation.
* dequeue(self): This method removes and returns the item from the **front** of the queue. With a list, removing an item from the beginning is done using pop(0). This is **not** an efficient operation for large lists as it requires shifting all subsequent elements to the left, resulting in an O(n) time complexity where 'n' is the number of elements in the list. It raises an IndexError if the queue is empty.
* is\_empty(self): Checks if the list is empty. This is an efficient O(1) operation.

The example usage demonstrates these methods:

* It creates an empty QueueWithList.
* Checks if it's empty.
* Enqueues three items (10, 20, 30).
* Checks if it's empty again.
* Uses dequeue twice to remove and print the first two items added (10 and 20), demonstrating the First-In, First-Out (FIFO) nature of a queue.
* Checks the empty status after dequeuing.
* Dequeues the last item (30).
* Checks the empty status again.
* Includes a try...except block to show how the IndexError is raised when trying to dequeue from an empty queue.

# TASK -3

CODE :