FR. CONCEICAO RODRIGUES COLLEGE OF ENGINEERIG

Department of Computer Engineering

Experiment 7 – Data structure in Python

1. Course Details:

Academic Year	**2023 - 24**	**Estimated Time**	**Experiment No. 7 – 02 Hours**
Course & Semester	**S.E. (COMP) – Sem. IV**	**Subject Name**	**Python Programming Lab**
Module No.	**03**	**Chapter Title**	**Python Basics**
Experiment Type	**Software Performance**	**Subject Code**	**CSL405**

Name of Student	Mark lopes	**Roll No.**	9913
Date of Performance.:	8/04/2024	**Date of Submission:**	9/04/2024
CO Mapping	**CSL405.3:** Implement data structures using in built libraries.		

Timeline (2)	Preparedness (2)	Effort (2)	Result (2)	Documentation (2)	Total (10)

2. Aim & Objective of Experiment

Write python program to:

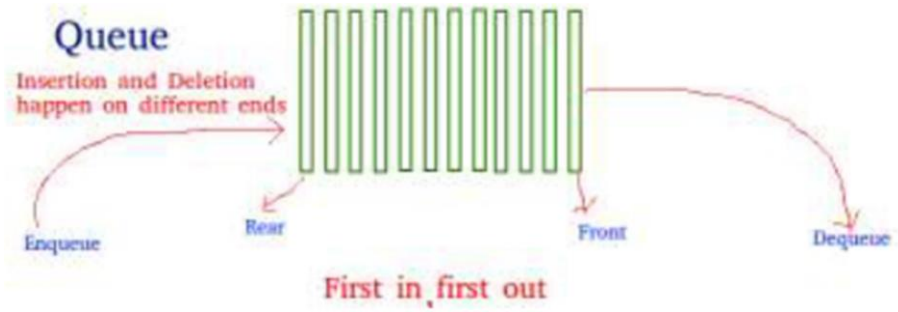
1) Implement Food Delivery Management System. Your program should support the following functionalities:
   a) Order Placement: When a customer places a food order, it should be added to the queue representing the orders awaiting preparation.
   b) Order Preparation: Once an order is placed, it should be picked up from the queue and prepared by the kitchen staff.
   c) Order Delivery: After preparation, the order should be removed from the queue and delivered to the customer.
   d) Display Order Status: Display the current status of orders, including those being prepared, those waiting for delivery, and those already delivered.

2) Implement shopping cart using linked-list.
The user should be able to add item, remove item, display all the items and calculate total amount of the cart. Each Item details contain (Item name, quantity, and price).

Objectives:
Implement different data structures in Python.

Pre-Requisite: Any programming language like C, C++
Tools: Python IDE

Theory:
Queue is a linear data structure that stores items in First in First out (FIFO) manner. With a queue the least recently added item is removed first. A good example of queue is any queue of consumers for a resource where the consumer that came first is served first.



Operations associated with queue are:

- Enqueue: Adds an item to the queue. If the queue is full, then it is said to be an Overflow condition – Time Complexity : O(1)
- Dequeue: Removes an item from the queue. The items are popped in the same order in which they are pushed. If the queue is empty, then it is said to be an Underflow condition – Time Complexity : O(1)
- Front: Get the front item from queue – Time Complexity : O(1)
- Rear: Get the last item from queue – Time Complexity : O(1)

There are various ways to implement a queue in Python. This article covers the implementation of queue using data structures and modules from Python library

Queue in Python can be implemented by the following ways:

- list
- collections.deque
- queue.Queue

Implementation using list:

List is a Python's built-in data structure that can be used as a queue. Instead of enqueue() and dequeue(), append() and pop() function is used. However, lists are quite slow for this purpose because inserting or deleting an element at the beginning requires shifting all of the other elements by one, requiring O(n) time.

Implementation using collections.deque:

Queue in Python can be implemented using deque class from the collections module. Deque is preferred over list in the cases where we need quicker append and pop operations from both the ends of container, as deque provides an O(1) time complexity for append and pop operations as compared to list which provides O(n) time complexity. Instead of enqueue and deque, append() and popleft() functions are used.

Implementation using queue.Queue:

Queue is built-in module of Python which is used to implement a queue. queue.Queue(maxsize) initializes a variable to a maximum size of maxsize. A maxsize of zero '0' means a infinite queue. This Queue follows FIFO rule.
There are various functions available in this module:

- maxsize – Number of items allowed in the queue.
- empty() – Return True if the queue is empty, False otherwise.
- full() – Return True if there are maxsize items in the queue. If the queue was initialized with

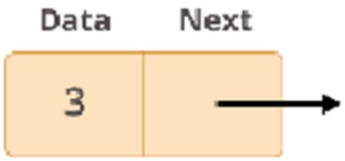
maxsize=0 (the default), then full() never returns True.

- get() – Remove and return an item from the queue. If queue is empty, wait until an item is available.
- get_nowait() – Return an item if one is immediately available, else raise QueueEmpty.
- put(item) – Put an item into the queue. If the queue is full, wait until a free slot is available before adding the item.
- put_nowait(item) – Put an item into the queue without blocking. If no free slot is immediately available, raise QueueFull.
- qsize() – Return the number of items in the queue.

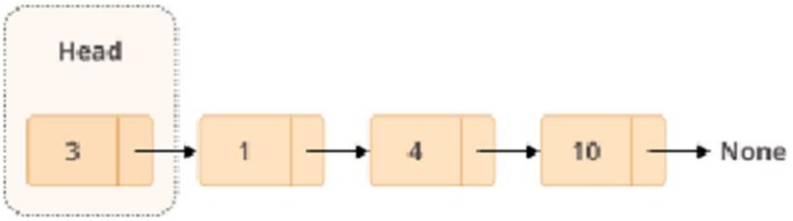
Linked List:

Linked lists are an ordered collection of objects. Each element of a linked list is called a node, and every node has two different fields:

1. Data contains the value to be stored in the node.
2. Next contains a reference to the next node on the list



A linked list is a collection of nodes. The first node is called the head, and it's used as the starting point for any iteration through the list. The last node must have its next reference pointing to None to determine the end of the list. Here's how it looks:



PostLab questions:

1) What is Doubly link list. Write a python program to create a doubly linked list and add a new node after a given node , traverse a doubly linked list.
2) Write a python program to implement linked list using collection.dequeue().

References:

Link List : https://realpython.com/linked-lists-python/
Queue: https://www.guru99.com/python-queue-example.html
Queue: https://www.geeksforgeeks.org/queue-in-python

1

```
import queue
import time

class Order:
    def __init__(self, order_id, item):
        self.order_id = order_id
        self.item = item
        self.status = "Placed"  # Possible statuses: Placed, Preparing, Delivered


def place_order(order_queue, order_id, item):
    order = Order(order_id, item)
    order_queue.put(order)
    print(f"Order {order_id} for {item} has been placed.")

def prepare_orders(order_queue, preparing_queue):
    print("Preparing orders...")
    while not order_queue.empty():
        order = order_queue.get()
        order.status = 'Preparing'
        preparing_queue.put(order)
        print(f"Order {order.order_id} for {order.item} is being prepared.")
        time.sleep(2)  # Simulate preparation time

def deliver_orders(preparing_queue, delivered_orders):
    print("Delivering orders...")
    while not preparing_queue.empty():
        order = preparing_queue.get()
        order.status = 'Delivered'
        delivered_orders.append(order)
        print(f"Order {order.order_id} for {order.item} has been delivered.")

def display_order_status(order_queue, preparing_queue, delivered_orders):
    print("---- Current Order Status ----")
    print("Orders Being Prepared:")
    for order in list(preparing_queue.queue):
        print(f"Order ID: {order.order_id}, Item: {order.item}, Status:
{order.status}")
    print("Orders Waiting for Preparation:")
    for order in list(order_queue.queue):
        print(f"Order ID: {order.order_id}, Item: {order.item}, Status:
{order.status}")
    print("Delivered Orders:")
    for order in delivered_orders:
        print(f"Order ID: {order.order_id}, Item: {order.item}, Status:
{order.status}")
    print("------------------------------")


if __name__ == "__main__":
    order_queue = queue.Queue()
    preparing_queue = queue.Queue()
```

```
    delivered_orders = []

    place_order(order_queue, 1, "Samosa")
    place_order(order_queue, 2, "Vada Pav")
    place_order(order_queue, 3, "Biryani")

    prepare_orders(order_queue, preparing_queue)

    deliver_orders(preparing_queue, delivered_orders)

    display_order_status(order_queue, preparing_queue, delivered_orders)
```

```
PS C:\Users\Mark Lopes\Desktop\college\Sem_4\Python\Lab7> pyth
Order 1 for Samosa has been placed.
Order 2 for Vada Pav has been placed.
Order 3 for Biryani has been placed.
Preparing orders...
Order 1 for Samosa is being prepared.
Order 2 for Vada Pav is being prepared.
Order 3 for Biryani is being prepared.
Delivering orders...
Order 1 for Samosa has been delivered.
Order 2 for Vada Pav has been delivered.
Order 3 for Biryani has been delivered.
---- Current Order Status ----
Orders Being Prepared:
Orders Waiting for Preparation:
Delivered Orders:
Order ID: 1, Item: Samosa, Status: Delivered
Order ID: 2, Item: Vada Pav, Status: Delivered
Order ID: 3, Item: Biryani, Status: Delivered
------------------------------
PS C:\Users\Mark Lopes\Desktop\college\Sem_4\Python\Lab7>
```

2

```
class Node:
    def __init__(self, item_name, quantity, price):
        self.item_name = item_name
        self.quantity = quantity
        self.price = price
        self.next = None

class ShoppingCart:
    def __init__(self):
        self.head = None  #Initialize the head of the linked list

    def add_item(self, item_name, quantity, price):
        new_item = Node(item_name, quantity, price)
        if not self.head:
            self.head = new_item
        else:
            current = self.head
            while current.next:
                current = current.next
            current.next = new_item
```

```
        print(f"{quantity} {item_name}(s) added to the cart.")

    def remove_item(self, item_name):
        if not self.head:
            print("Cart is empty.")
            return

        current = self.head
        prev = None
        found = False

        while current and not found:
            if current.item_name == item_name:
                found = True
            else:
                prev = current
                current = current.next

        if not found:
            print(f"{item_name} not found in the cart.")
            return

        if prev is None:
            self.head = current.next
        else:
            prev.next = current.next

        print(f"{item_name} removed from the cart.")

    def display_items(self):
        if not self.head:
            print("Cart is empty.")
            return

        current = self.head
        print("Items in the cart:")
        while current:
            print(f"{current.quantity} {current.item_name}(s) - ${current.price}
each")
            current = current.next

    def calculate_total_amount(self):
        total_amount = 0
        current = self.head
        while current:
            total_amount += current.quantity * current.price
            current = current.next
        return total_amount


if __name__ == "__main__":
    cart = ShoppingCart()
```

```python
cart.add_item("Apple", 3, 1.25)
cart.add_item("Banana", 5, 0.75)
cart.add_item("Mango", 2, 2.50)

cart.display_items()

cart.remove_item("Banana")

cart.display_items()

total_amount = cart.calculate_total_amount()
print(f"Total amount in the cart: ${total_amount}")
```

```
3 Apple(s) added to the cart.
5 Banana(s) added to the cart.
2 Mango(s) added to the cart.
Items in the cart:
3 Apple(s) - $1.25 each
5 Banana(s) - $0.75 each
2 Mango(s) - $2.5 each
Banana removed from the cart.
Items in the cart:
3 Apple(s) - $1.25 each
2 Mango(s) - $2.5 each
Total amount in the cart: $8.75
PS C:\Users\Mark Lopes\Desktop\college\Sem_4\Python\Lab7>
```

Postlab:-

1)

A doubly linked list is a complex type of linked list where a node contains a pointer to the previous as well as the next node in the sequence. Therefore, in a doubly-linked list, a node consists of three components: node data, pointer to the next node in node(next pointer), pointer to the former node (previous pointer).

```
class Node:
    def __init__(self, data):
        self.data = data
        self.prev = None
        self.next = None

class DoublyLinkedList:
    def __init__(self):
        self.head = None
        self.tail = None

    def append(self, data):
        new_node = Node(data)
        if not self.head: # if there is no node in the linked list
            self.head = new_node # the new node is the first node
        else:
            self.tail.next = new_node
            new_node.prev = self.tail
        self.tail = new_node

    def insert_after(self, existing_node_data, data):
        new_node = Node(data)
        current = self.head

        while current:
            if current.data == existing_node_data:
                new_node.next = current.next
                new_node.prev = current
                if current.next:
                    current.next.prev = new_node
                current.next = new_node
                if current == self.tail:
                    self.tail = new_node
                break
            current = current.next

    def traverse(self):
        current = self.head
        while current:
            print(current.data, end=" ")
            current = current.next
        print()
```

```
if __name__ == "__main__":
    dll = DoublyLinkedList()

    dll.append(1)
    dll.append(2)
    dll.append(3)
    dll.append(4)
    dll.append(5)

    print("Doubly linked list: ")
    dll.traverse()

    dll.insert_after(3, 6)

    print("Doubly linked list after insertion: ")
    dll.traverse()
```

```
Doubly linked list:
1 2 3 4 5
Doubly linked list after insertion:
1 2 3 6 4 5
PS C:\Users\Mark Lopes\Desktop\college\Sem_4\Python\Lab7>
```

2)

```
from collections import deque

class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class LinkedList:
    def __init__(self):
        self.head = None
        self.deque = deque()

    def append(self, data):
        new_node = Node(data)
        if not self.head:
            self.head = new_node
        else:
            current = self.head
            while current.next:
                current = current.next
            current.next = new_node
        self.deque.append(new_node)

    def insert_after(self, prev_node_data, data):
```

```
        new_node = Node(data)
        current = self.head
        while current:
            if current.data == prev_node_data:
                new_node.next = current.next
                current.next = new_node
                self.deque.insert(self.deque.index(current) + 1, new_node)
                break
            current = current.next
        else:
            raise ValueError(f"Node with data '{prev_node_data}' not found")

    def remove(self, data):
        current = self.head
        prev = None
        while current:
            if current.data == data:
                if prev:
                    prev.next = current.next
                else:
                    self.head = current.next
                self.deque.remove(current)
                return
            prev = current
            current = current.next
        raise ValueError(f"Node with data '{data}' not found")

    def traverse(self):
        current = self.head
        while current:
            print(current.data, end=" ")
            current = current.next


if __name__ == "__main__":
    linked_list = LinkedList()

    linked_list.append(1)
    linked_list.append(2)
    linked_list.append(3)

    print("Linked List:")
    linked_list.traverse()

    linked_list.insert_after(3, 4)

    print("\nLinked List after insertion:")
    linked_list.traverse()

    linked_list.remove(2)

    print("\nLinked List after removal:")
    linked_list.traverse()
```

```
PS C:\Users\Mark Lopes\Desktop\college\Sem_4\Python\Lab7> python -
Linked List:
1 2 3
Linked List after insertion:
1 2 3 4
Linked List after removal:
1 3 4
PS C:\Users\Mark Lopes\Desktop\college\Sem_4\Python\Lab7>
```