

***NAMAL UNIVERSITY MIANWALI***

***DEPARTMENT OF ELECTRICAL ENGINEERING***

***DATA STRUCTURE AND ALGORITHM***

***LAB # 08***

***REPORT***

***Title : Deques in Python***

|  |  |
| --- | --- |
| ***Name*** | ***Fahim-Ur-Rehman Shah*** |
| ***Roll No*** | ***NIM-BSEE-2021-24*** |
| ***Intructor*** | ***Ms. Naureen Shaukat*** |
| ***Lab Engineer*** | ***Mr .Ali Hasnain*** |
| ***Date*** | ***21-May-2023*** |
| ***Marks*** |  |

**Instructions:**

1. This is an individual lab. You will perform the tasks individually and submit a report.
2. Some of these tasks (marked as ‘Example’) are for practice purposes only while others (marked as ‘Task’) have to be answered in the report.
3. When asked to display an output in the task, either save it as jpeg or take a screenshot, in order to insert it in the report.
4. The report should be submitted on the given template, including:
   1. Code (copy and pasted, NOT a screenshot)
   2. Output figure (as instructed in 3)
   3. Explanation where required
5. The report should be properly formatted, with easy to read code and easy to see figures.
6. Plagiarism or any hint thereof will be dealt with strictly. Any incident where plagiarism is caught, both (or all) students involved will be given zero marks, regardless of who copied whom. Multiple such incidents will result in disciplinary action being taken.
7. Late submission of report is allowed within 03 days after lab with 20% deduction of marks every day.
8. You have to submit report in pdf format (Reg.X\_DSA\_LabReportX.pdf).

|  |
| --- |
| **Complete Code With Task 1 and Task 2** |

|  |
| --- |
| **Python Code :**  class Double\_Ended\_Queue:      def \_\_init\_\_(self):          """Initialize a Deque object with a fixed capacity"""          self.arr = []   # Circular array to hold the elements      def is\_empty(self):          """Check if the deque is empty"""          return len(self.arr) == 0      def get\_front(self):          """Get the value of the front element"""          if self.is\_empty():              return None          else:              return self.arr[0]        def insert\_front(self, value):          """Insert an element at the front of the deque"""          self.arr.insert(0,value)        def delete\_front(self):          """Delete the element at the front of the deque"""          if self.is\_empty():              return False  # Return False if deque is already empty          else:              self.arr.pop(0)        def display(self):          return self.arr        def insert\_back(self,value):          self.arr.append(value)        def delete\_back(self):          self.arr.pop(-1)    # Create a deque with a capacity of 4  d = Double\_Ended\_Queue()  print("Insert at front :")  d.insert\_front(1)  d.insert\_front(2)  d.insert\_front(3)  print(d.display())  print("\nInsert at back :")  d.insert\_back(4)  d.insert\_back(5)  d.insert\_back(6)  print(d.display())  print("\nDelete from front: ")  d.delete\_front()  print(d.display())  print("\nDelete from back: ")  d.delete\_back()  print(d.display())  print("\nGet from front: ")  print(d.get\_front())  print("Is list empty:", d.is\_empty(),"\n")  **Output Screenshot:**    **Explanation :**  This Python code defines a class called ***Double\_Ended\_Queue*** that represents a double-ended queue or deque. It allows you to add and remove elements from both the front and the back. The class has methods like ***insert\_front*** to add an element at the front, ***insert\_back*** to add an element at the back, ***delete\_front*** to remove an element from the front, and ***delete\_back*** to remove an element from the back. It also has methods like ***get\_front*** to get the value of the front element and display to show the current contents of the deque. The code creates an instance of the deque, performs various operations on it, and displays the results. |

|  |
| --- |
| **Task 1: Using above example code, make a method (insert\_back) to insert**  **elements at the back side of deque. Also make a method (delete\_back) to delete**  **elements at the back side of deque.** |

|  |
| --- |
| **Python Code :**     def insert\_back(self,value):          self.arr.append(value)        def delete\_back(self):          self.arr.pop(-1)  #/////////////////////////////////////////////////////  #Main Code  # Create a deque with a capacity of 4  d = Double\_Ended\_Queue()  print("Insert at front :")  d.insert\_front(1)  d.insert\_front(2)  d.insert\_front(3)  print(d.display())  print("\nInsert at back :")  d.insert\_back(4)  d.insert\_back(5)  d.insert\_back(6)  print(d.display())  print("\nDelete from front: ")  d.delete\_front()  print(d.display())  print("\nDelete from back: ")  d.delete\_back()  print(d.display())  print("\nGet from front: ")  print(d.get\_front())  print("Is list empty:", d.is\_empty(),"\n")  **Explanation :**  Two methods are defined for the ***Double\_Ended\_Queue*** class.   * The ***insert\_back(self, value)*** method allows you to add an element to the back of the deque. It takes a value parameter and uses the append function to add the value to the end of the arr list, which represents the deque. * The ***delete\_back(self)*** method enables you to remove an element from the back of the deque. It doesn't take any parameters. The method uses the pop function with an index of -1 to remove and return the last element from the arr list, effectively deleting it from the deque.   ***Main Code Explanation :***  In this code, a ***double-ended*** queue (deque). It is represented by the variable d using the ***Double\_Ended\_Queue*** class. Elements are then inserted at the front and back of the deque using the ***insert\_front*** and ***insert\_back*** methods, respectively. The inserted elements are displayed by calling the display method.  Next, an element is deleted from both the front and back of the deque using the ***delete\_front*** and ***delete\_back*** methods, respectively. The updated contents of the deque are displayed after each deletion.  The value of the front element is obtained using the ***get\_front*** method and printed to the console. Lastly, the ***is\_empty*** method is used to check if the deque is empty, and the result is printed to the console. |

|  |
| --- |
| **Task 2 : Make a method to print deque of the above Task 1. Also make a main**  **method to call the above methods for output.** |

|  |
| --- |
| **Python Code :**     def display(self):          print(self.arr)  **Explanation :**  In this code, a double-ended queue (deque) is created with a capacity of 4. It is represented by the variable d using the Double\_Ended\_Queue class. Elements are then inserted at the front and back of the deque using the insert\_front and insert\_back methods, respectively. The inserted elements are displayed by calling the display method.  Next, an element is deleted from both the front and back of the deque using the delete\_front and delete\_back methods, respectively. The updated contents of the deque are displayed after each deletion.  The value of the front element is obtained using the get\_front method and printed to the console. Lastly, the is\_empty method is used to check if the deque is empty, and the result is printed to the console. |

|  |
| --- |
| **Task 3: Task 3: Write a python program to implement a ticket booking system where**  **customers can book tickets either from the front or the back of the double ended**  **queue. The system should have the following methods:**   * **book\_front(ticket): Adds a ticket to the front of the queue.** * **book\_back(ticket): Adds a ticket to the back of the queue.** * **cancel\_front(): Cancels the ticket at the front of the queue.** * **cancel\_back(): Cancels the ticket at the back of the queue.** * **get\_total\_tickets(): Returns the total number of tickets booked.**   **Make a class TicketBookingSystem to implement the deque. Also make a main**  **method to call the above methods for output (Number of seats booked and**  **canceled from front and back, and total number of seats booked.).**  **You can import deque using this piece of code:**  **from collections import deque.** |

|  |
| --- |
| **Python Code :**  class TicketBookingSystem:      def \_\_init\_\_(self):          """Initialize a TicketBookingSystem object"""          self.seats = []   # List to hold the seats      def is\_empty(self):          """Check if the booking system is empty"""          return len(self.seats) == 0        def book\_front(self, name):          """Book a seat at the front of the booking system"""          self.seats.insert(0, name)        def cancel\_front(self):          """Cancel the seat at the front of the booking system"""          if self.is\_empty():              return False  # Return False if the booking system is already empty          else:              self.seats.pop(0)        def display(self):          """Display the seats in the booking system"""          return self.seats        def book\_back(self, name):          """Book a seat at the back of the booking system"""          self.seats.append(name)        def cancel\_back(self):          """Cancel the seat at the back of the booking system"""          self.seats.pop(-1)        def get\_total\_tickets(self):          """Get the total number of booked seats"""          return len(self.seats)  # Create a booking system  booking\_system = TicketBookingSystem()  print("Book Seat at front:")  booking\_system.book\_front("Muhammad")  booking\_system.book\_front("Ahmed")  booking\_system.book\_front("Fatima")  print(booking\_system.display())  print("\nBook Seat at back:")  booking\_system.book\_back("Ali")  booking\_system.book\_back("Aisha")  booking\_system.book\_back("Yusuf")  print(booking\_system.display())  print("\nCancel Seat from front:")  booking\_system.cancel\_front()  print(booking\_system.display())  print("\nCancel Seat from back:")  booking\_system.cancel\_back()  print(booking\_system.display())  print("Total Tickets:", booking\_system.get\_total\_tickets(), "\n")  **Output:**    **Explanation :**  The code represents a Ticket Booking System implemented using a class in Python. The ***TicketBookingSystem*** class provides methods to book seats, cancel seats, display the booked seats, and get the total number of booked seats. The booking system maintains a list called seats to store the booked seats.  To book a seat, the ***book\_front*** method is used to add a passenger's name to the front of the list, simulating booking at the front of the system. Similarly, the ***book\_back*** method adds a passenger's name to the end of the list, representing booking at the back of the system.  Cancellation of seats is supported by the ***cancel\_front*** and ***cancel\_back*** methods. They remove the first and last passenger's names from the list, respectively, if the booking system is not already empty.  The display method simply returns the list of booked seats. Lastly, the ***get\_total\_tickets*** method calculates and returns the total number of booked seats by returning the length of the seats list.  The code demonstrates the functionality of the Ticket Booking System by booking and canceling seats at both the front and back, displaying the current booked seats, and displaying the total number of booked tickets. |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Lab Evaluation Rubrics** | | | | | | | |
| **Domain** | **CLOs/**  **Rubric** | **Performance Indicator** | **Unsatisfactory**  **0-5** | **Marginal**  **5-10** | **Satisfactory**  **11-15** | **Exemplary**  **16-20** | **Allocate d Marks** |
| **Psychomotor** | **CLO:1**  **R2** | Implementation with Results  **(P)** | Does not try to solve problems. Many mistakes in code and difficult to comprehend for the instructor. There is not result of the problem. | Does not suggests or refine solutions but is willing to try out solutions suggested by others. Few mistakes in code, but done along with comments, and easy to comprehend for the instructor. Few mistake in result. | Refines solutions suggested by others. Complete and error-free code is done. No comments in the code, but easy to comprehend for the instructor. Results are correctly produced. | Actively looks for and suggests solution to problems. Complete and error free code is done, easy to comprehend for the instructor. Results are correctly produced. Student incorporated comments in the code. |  |
| **Affective** | **CLO:3**  **R3** | Lab Report **(A)** | Code of the problem is not given. Outputs are not provided. Explanation of the solution is not stated. | Code of the problem is not given. Output is not complete. Explanation of the solution is not satisfactory. | Code of the problem is not given. Output is completely given. Explanation of the solution is not satisfactory. | Code of the problem is not given. Output is completely given. Explanation of the solution is satisfactory. |  |
| **CLO:1**  **R5** | Discipline and Behavior **(A)** | Got and wandered around. Chased others, ran, or played around. More than two incidents of talking non-lab related stuff in lab and/or any talk with other groups, voice level exceeding the appropriate level, use of cell phones and involvement in any non lab activity. | Got out of seat and wander around for some time. No more than two incidents of talking non-lab related stuff in lab. Voice level exceeding the appropriate level, use of cell phones and involvement in any non-lab related activity. | Stayed in seat and got up for a specific lab related reason, but took more time than required to do the job. No more than one incidents of talking non-lab related stuff in lab. Voice level exceeding the appropriate level, use of cell phones and involvement in any non-lab related activity. | Stayed in seat and got up for a specific lab related reason. Took care of lab related business and sat down right away. Voice level kept appropriate. Not used cell phones or involved in any non- lab related activity. |  |