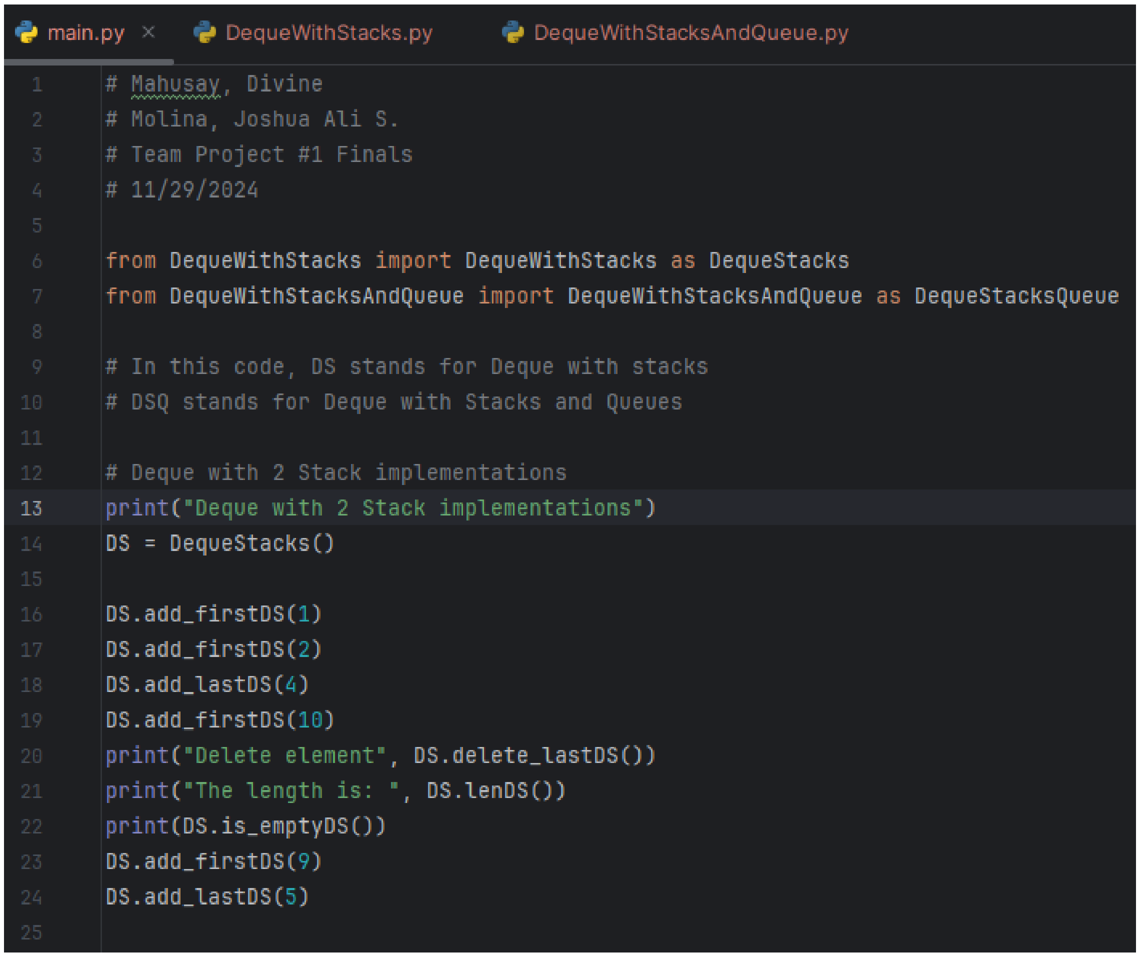
Mahusay, Divine Mars

Molina, Joshua Ali S.

IDB2 DSALGO1

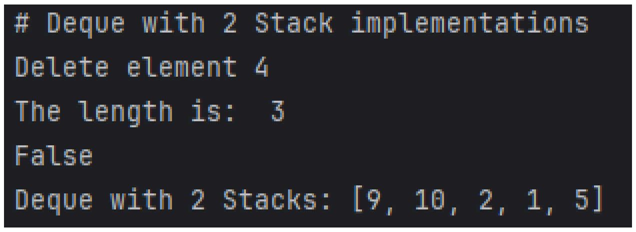
11/29/2024

Deque with 2 Stack implementations:



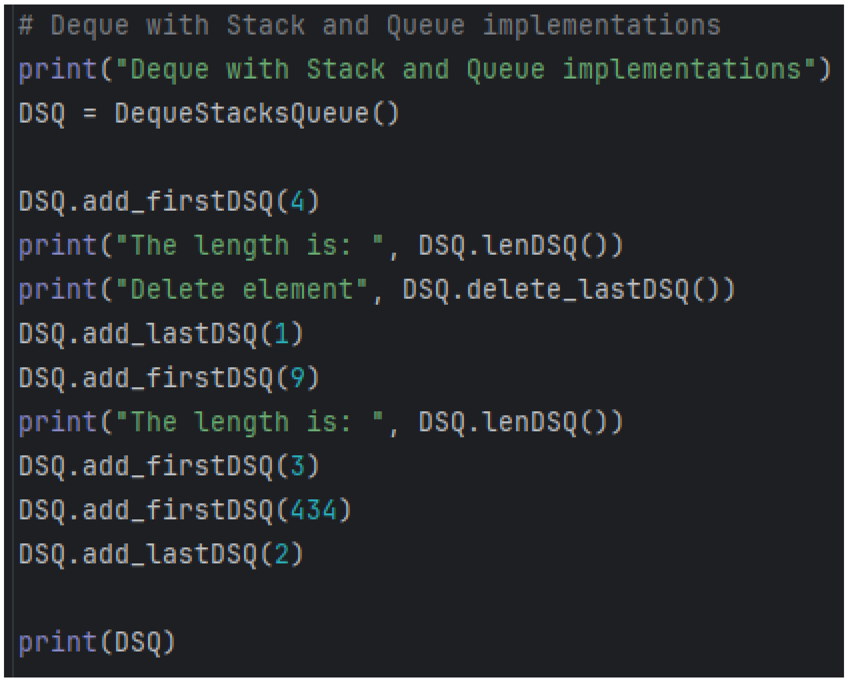
main.py × DequeWithStacks.py DequeWithStacksAndQueue.py

```python
# Mahusay, Divine
# Molina, Joshua Ali S.
# Team Project #1 Finals
# 11/29/2024

from DequeWithStacks import DequeWithStacks as DequeStacks
from DequeWithStacksAndQueue import DequeWithStacksAndQueue as DequeStacksQueue

# In this code, DS stands for Deque with stacks
# DSQ stands for Deque with Stacks and Queues

# Deque with 2 Stack implementations
print("Deque with 2 Stack implementations")
DS = DequeStacks()

DS.add_firstDS(1)
DS.add_firstDS(2)
DS.add_lastDS(4)
DS.add_firstDS(10)
print("Delete element", DS.delete_lastDS())
print("The length is: ", DS.lenDS())
print(DS.is_emptyDS())
DS.add_firstDS(9)
DS.add_lastDS(5)

```

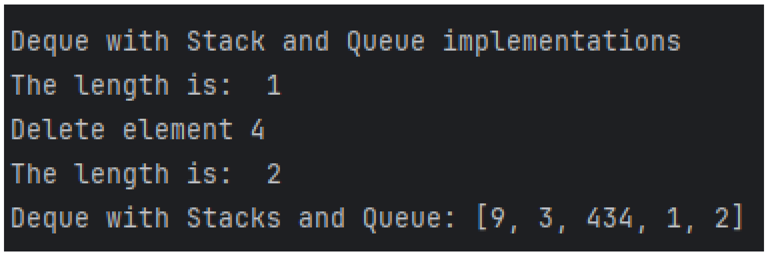


```
# Deque with 2 Stack implementations
Delete element 4
The length is:  3
False
Deque with 2 Stacks: [9, 10, 2, 1, 5]
```

Deque with Stack and Queue Implementation:



```python
# Deque with Stack and Queue implementations
print("Deque with Stack and Queue implementations")
DSQ = DequeStacksQueue()

DSQ.add_firstDSQ(4)
print("The length is: ", DSQ.lenDSQ())
print("Delete element", DSQ.delete_lastDSQ())
DSQ.add_lastDSQ(1)
DSQ.add_firstDSQ(9)
print("The length is: ", DSQ.lenDSQ())
DSQ.add_firstDSQ(3)
DSQ.add_firstDSQ(434)
DSQ.add_lastDSQ(2)

print(DSQ)
```



```
Deque with Stack and Queue implementations
The length is:  1
Delete element 4
The length is:  2
Deque with Stacks and Queue: [9, 3, 434, 1, 2]
```

DequeWithStacks class:

```python
class DequeWithStacks:
  def __init__(self):
    self.frontStack = []
    self.backStack = []


  def __str__(self):


    combined = self.frontStack[::-1] + self.backStack
    return f"Deque with 2 Stacks: {combined}"


  def __repr__(self):
    return self.__str__()
  def _move_to_stack(self, source, destination):
    while source:
    destination.append(source.pop())

  def add_firstDS(self, value):
    # adds an element to the front of the Deque
    self.frontStack.append(value)

  def add_lastDS(self, value):
    # Adds an element to the back of the deque
    self.backStack.append(value)

  def delete_firstDS(self):
    # deletes the first element
    if not self.frontStack:
        if not self.backStack:
            raise IndexError("pop_front from empty deque")
        self._move_to_stack(self.backStack, self.frontStack)
    return self.frontStack.pop()

  def delete_lastDS(self):
    # deletes the last element
    if not self.backStack:
        if not self.frontStack:  # If both stacks are empty
            raise IndexError("pop_back from empty deque")
        self._move_to_stack(self.frontStack, self.backStack)
    return self.backStack.pop()

  def firstDS(self):
    # returns the first element
    if not self.frontStack:
        if not self.backStack:
            raise IndexError("peek_front from empty deque")
        self._move_to_stack(self.backStack, self.frontStack)
    return self.frontStack[-1]

  def lastDS(self):
    # returns the last element

    if not self.backStack:

        if not self.frontStack:

            raise IndexError("peek_back from empty deque")

        self._move_to_stack(self.frontStack, self.backStack)

    return self.backStack[-1]


  def is_emptyDS(self):

    # checks if the deque is empty

    return not self.frontStack and not self.backStack


  def lenDS(self):

    # returns the length of the deque

    return len(self.frontStack) + len(self.backStack)
```

DequeWithStacksAndQueue class:

```
from LinkedStack import LinkedStack as LinkedStack
from LinkedQueue import LinkedQueue as LinkedQueue
class DequeWithStacksAndQueue:
    def __init__(self):
        self.frontStack = LinkedStack()
        self.backStack = LinkedStack()
        self.queue = LinkedQueue()

    def _move_to_stack(self, source, destination):
        while not source.is_empty():
            destination.push(source.pop())

    def __str__(self):
        elements = []

        temp_stack = LinkedStack()
        while not self.frontStack.is_empty():
            temp_stack.push(self.frontStack.pop())
        while not temp_stack.is_empty():
            value = temp_stack.pop()
            elements.append(value)
            self.frontStack.push(value)

        temp_stack = LinkedStack()
        while not self.backStack.is_empty():
            temp_stack.push(self.backStack.pop())
        while not temp_stack.is_empty():
            value = temp_stack.pop()
            elements.append(value)
            self.backStack.push(value)

        return f"Deque with Stacks and Queue: {elements}"

    def __repr__(self):
        return self.__str__()

    def add_firstDSQ(self, value):
        # Adds an element to the front of the Deque
        self.frontStack.push(value)

    def add_lastDSQ(self, value):
        # Adds an element to the back of the Deque
        self.backStack.push(value)

    def delete_firstDSQ(self):
        # Deletes the first element
        if self.frontStack.is_empty():
            if self.backStack.is_empty():
                raise IndexError("delete_first from empty deque")
            self._move_to_stack(self.backStack, self.frontStack)
        return self.frontStack.pop()

    def delete_lastDSQ(self):
        # Deletes the last element
        if self.backStack.is_empty():
            if self.frontStack.is_empty():
                raise IndexError("delete_last from empty deque")
            self._move_to_stack(self.frontStack, self.backStack)
        return self.backStack.pop()

    def firstDSQ(self):
        # Returns the first element
        if self.frontStack.is_empty():
            if self.backStack.is_empty():
                raise IndexError("first from empty deque")
            self._move_to_stack(self.backStack, self.frontStack)
        return self.frontStack.peek()

    def lastDSQ(self):
        # Returns the last element
        if self.backStack.is_empty():
            if self.frontStack.is_empty():
                raise IndexError("last from empty deque")
            self._move_to_stack(self.frontStack, self.backStack)
        return self.backStack.peek()

    def is_emptyDSQ(self):
        # Checks if the Deque is empty
    return self.frontStack.is_empty() and self.backStack.is_empty()

def lenDSQ(self):
    # Retuns the length of the deque
    return len(self.frontStack) + len(self.backStack)
```