X

Programming Assignment 3

Due on 2021-11-24, 23:59 IST

A queue is a sequence that allows you to add elements at the end (the rear) and remove elements from the front, like a normal queue in real life. A double-ended queue, or **deque**, allows insert and remove at both ends, so it supports four operations: insert-front, insert-rear, delete-front and delete-rear.

A partial definition of the Python class `Deque` is given in the code window with an implementation of `insertfront()`. The deque is stored in a sequence of `Node` objects. The `Deque` points to the first node (`head`) and last node (`tail`) in the sequence.

Complete the definition of `Deque` by adding functions `insertrear()`, `deletefront()` and `deleterear()`. Make sure the indentation for your code is correct with respect to the enclosing class definition.

Your function will be tested using the function `testdeque()` which is shown above the class definition. This function creates an empty `Deque` and then performs the list of operations provided to it. Each element of the list of operations is one of the following:

- `("if",v)`, insert `v` using `mydeque.insertfront(v)`
- `("ir",v)`, insert `v` using `mydeque.insertrear(v)`
- `("df",)`, use `mydeque.deletefront()` to remove and return the value at the front of the queue
- `("dr",)`, use `mydeque.deleterear()` to remove and return the value at the front of the queue

At the end of the sequence, `testdeque()` prints out the current deque, the list of values extracted by the `deletefront()` and `deleterear()` operations and the values of the nodes pointed to by the `head` and `tail` of the deque.

- Do not write commands to read any input or print any output.
- You may define additional auxiliary functions as needed.
- In all cases you may assume that the value passed to the function is of

(/programming_2021/progassignment?name=20)

PDSP Assignment 4, due Fri 17 Dec 2021

Quiz 3, Thu 16 Dec 2021

PDSP Quiz 4, Thu 23 Dec 2021

PDSP Assignment 5, due Fri 31 Dec 2021

In all cases you may assume that the value passed to the function is of the expected type, so your function does not have to check for malformed inputs.

- There are normally some public test cases and some (hidden) private test cases.
- "Compile and run" will evaluate your submission against the public test cases.
- "Submit" will evaluate your submission against the hidden private test cases. There are 10 private test cases, with equal weightage. You will get feedback about which private test cases pass or fail, though you cannot see the actual test cases.
- Ignore warnings about "Presentation errors".
- You can submit as many times as you like. Your final submission will be used for scoring.

Here are some examples to show how the code should work.

```
  >>> testdeque([("if",3),("ir",4),("if",1),("ir",2),("df",),("d
r",)])
  [3, 4] [1, 2] 3 4

  >>> testdeque([("ir",1),("if",2),("df",),("df",),("df",),("ir",2),
("if",1)])
  [1, 2] [2, 1] 1 2
```

Private Test cases used for evaluation	Input	Expected Output	Actual Output	Status
Test Case 1	`testdeque([("df",),("dr",),("if",62),("if",2),("if",75),("df",),("ir",16),("dr",),("dr",),("df",),("dr",),("ir",81),("ir",28),("ir",40),("if",74),("dr",),("dr",),("ir",12),("ir",15),("df",)])`	`[81, 12, 15] [75, 16, 62, 2, 40, 28, 74] 81 15\n`	`[81, 12, 15] [75, 16, 62, 2, 40, 28, 74] 81 15\n`	Passed
Test Case 2	`testdeque([("ir",46),("df",),("df",),("if",27),("dr",),("ir",96),("ir",15),("dr",),("df",),("if",98),("if",82),("ir",94),("if",50),("if",84),("dr",),("df",),("dr",),("if",65),("dr",),("if",11)])`	`[11, 65, 50] [46, 27, 15, 96, 94, 84, 98, 82] 11 50\n`	`[11, 65, 50] [46, 27, 15, 96, 94, 84, 98, 82] 11 50\n`	Passed

Test Case 3	`testdeque([("if",96),("if",10), ("if",40),("if",1),("ir",84), ("dr",),("dr",),("df",),("df",), ("if",44),("if",47),("if",75), ("dr",),("ir",32),("ir",58), ("if",84),("if",35),("dr",), ("ir",74),("df",)])`	`[84, 75, 47, 44, 32, 74] [84, 96, 1, 40, 10, 58, 35] 84 74\n`	`[84, 75, 47, 44, 32, 74] [84, 96, 1, 40, 10, 58, 35] 84 74\n`	Passed
Test Case 4	`testdeque([("if",17),("ir",43), ("if",37),("dr",),("ir",67), ("df",),("ir",55),("if",44), ("ir",54),("df",),("if",6),("dr",), ("if",48),("ir",4),("if",71), ("if",27),("ir",58),("df",), ("if",87),("ir",51)])`	`[87, 71, 48, 6, 17, 67, 55, 4, 58, 51] [43, 37, 44, 54, 27] 87 51\n`	`[87, 71, 48, 6, 17, 67, 55, 4, 58, 51] [43, 37, 44, 54, 27] 87 51\n`	Passed
Test Case 5	`testdeque([("if",84),("dr",), ("ir",62),("ir",60),("dr",), ("ir",38),("if",99),("df",), ("df",),("if",40),("ir",59), ("dr",),("if",65),("if",51), ("ir",89),("ir",60),("ir",86), ("if",38),("df",),("dr",)])`	`[51, 65, 40, 38, 89, 60] [84, 60, 99, 62, 59, 38, 86] 51 60\n`	`[51, 65, 40, 38, 89, 60] [84, 60, 99, 62, 59, 38, 86] 51 60\n`	Passed
Test Case 6	`testdeque([("ir",46),("dr",), ("df",),("if",56),("if",68), ("dr",),("dr",),("df",),("dr",), ("ir",36),("dr",),("ir",34), ("df",),("df",),("if",50), ("ir",48),("if",0),("df",), ("ir",96),("if",47),("df",), ("if",41),("dr",),("dr",),("df",), ("df",),("ir",29),("df",), ("if",92),("df",),("df",),("df",), ("ir",23),("if",53),("df",), ("if",2),("if",93),("df",),("dr",), ("if",28)])`	`[28, 2] [46, 56, 68, 36, 34, 0, 47, 96, 48, 41, 50, 29, 92, 53, 93, 23] 28 2\n`	`[28, 2] [46, 56, 68, 36, 34, 0, 47, 96, 48, 41, 50, 29, 92, 53, 93, 23] 28 2\n`	Passed

Test Case 7	testdeque([("dr",),("if",41), ("ir",87),("ir",55),("ir",25), ("if",1),("df",),("df",),("if",84), ("dr",),("if",59),("ir",30), ("dr",),("if",5),("ir",41), ("if",49),("ir",68),("df",), ("dr",),("ir",84),("dr",),("df",), ("dr",),("df",),("df",),("dr",), ("dr",),("ir",97),("ir",54), ("if",97),("ir",68),("ir",13), ("ir",71),("ir",66),("if",11), ("if",14),("ir",45),("df",), ("if",71),("df",)])	[11, 97, 97, 54, 68, 13, 71, 66, 45] [1, 41, 25, 30, 49, 68, 84, 5, 41, 59, 84, 55, 87, 14, 71] 11 45\n	[11, 97, 97, 54, 68, 13, 71, 66, 45] [1, 41, 25, 30, 49, 68, 84, 5, 41, 59, 84, 55, 87, 14, 71] 11 45\n	Passed
Test Case 8	testdeque([("ir",2),("ir",79), ("dr",),("ir",52),("ir",80), ("if",21),("ir",52),("ir",63), ("if",45),("ir",52),("dr",), ("dr",),("dr",),("df",),("if",22), ("ir",36),("df",),("ir",59), ("dr",),("if",59),("if",31), ("dr",),("ir",97),("df",), ("ir",23),("if",78),("df",), ("if",5),("df",),("ir",86),("dr",), ("if",98),("if",42),("if",79), ("df",),("df",),("if",16), ("ir",95),("dr",),("dr",)]) [16, 98, 59, 21, 2, 52, 80, 97] [79, 52, 63, 52, 45, 22, 59, 36, 31, 78, 5, 86, 79, 42, 95, 23] 16 97	[16, 98, 59, 21, 2, 52, 80, 97] [79, 52, 63, 52, 45, 22, 59, 36, 31, 78, 5, 86, 79, 42, 95, 23] 16 97\n	[16, 98, 59, 21, 2, 52, 80, 97] [79, 52, 63, 52, 45, 22, 59, 36, 31, 78, 5, 86, 79, 42, 95, 23] 16 97\n	Passed
Test Case 9	testdeque([("df",),("dr",), ("if",34),("if",55),("dr",), ("dr",),("ir",65),("if",87), ("df",),("ir",43),("if",96), ("df",),("ir",0),("dr",),("df",), ("if",71),("df",),("if",64), ("dr",),("if",34),("if",18), ("df",),("dr",),("dr",),("ir",89), ("df",),("if",84),("if",34), ("ir",51),("df",),("ir",77), ("dr",),("if",33),("ir",48), ("if",40),("if",21),("ir",81), ("if",7),("df",),("if",30)])	[30, 21, 40, 33, 84, 51, 48, 81] [34, 55, 87, 96, 0, 65, 71, 43, 18, 64, 34, 89, 34, 77, 7] 30 81\n	[30, 21, 40, 33, 84, 51, 48, 81] [34, 55, 87, 96, 0, 65, 71, 43, 18, 64, 34, 89, 34, 77, 7] 30 81\n	Passed

Test Case 10	`testdeque([("if",46),("if",35), ("ir",94),("if",95),("if",7), ("if",61),("df",),("df",),("dr",), ("if",53),("df",),("if",48), ("ir",72),("df",),("df",), ("ir",84),("if",86),("ir",0), ("dr",),("ir",4),("df",),("if",16), ("if",0),("ir",51),("df",), ("if",45),("dr",),("ir",31), ("df",),("if",89),("dr",), ("ir",32),("df",),("dr",), ("if",80),("df",),("dr",), ("ir",81),("df",),("ir",27)])`	`[35, 46, 72, 84, 81, 27] [61, 7, 94, 53, 48, 95, 0, 86, 0, 51, 45, 31, 89, 32, 80, 4, 16] 35 27\n`	`[35, 46, 72, 84, 81, 27] [61, 7, 94, 53, 48, 95, 0, 86, 0, 51, 45, 31, 89, 32, 80, 4, 16] 35 27\n`	Passed

The due date for submitting this assignment has passed.
10 out of 10 tests passed.
You scored 100.0/100.

Assignment submitted on 2021-11-23, 17:56 IST

Your last recorded submission was :

```
def testdeque(l):
  mydeque = Deque()
  extractedlist = []
  for op in l:
    if op[0] == "if":
      mydeque.insertfront(op[1])
    elif op[0] == "ir":
      mydeque.insertrear(op[1])
    elif op[0] == "df":
      v = mydeque.deletefront()
      if v != None:
        extractedlist.append(v)
    elif op[0] == "dr":
      v = mydeque.deleterear()
      if v != None:
        extractedlist.append(v)
  print(mydeque,extractedlist,mydeque.head.value,mydeque.tail.value)
  return

##########


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

class Deque:
  def __init__(self):
    newnode = Node()
    self.head = newnode
    self.tail = newnode

  def isempty(self):
    return(self.head.value == None)

  def insertfront(self,v):
    if self.isempty():
      self.head.value = v
    else:
      newnode = Node()
      newnode.value = self.head.value
      newnode.next = self.head.next
```

```
      self.head.value = v
      self.head.next = newnode
      self.tail = newnode
      while self.tail.next != None:
        self.tail = self.tail.next

  def __str__(self):
    if self.head.value == None:
      return(str([]))
    else:
      ptr = self.head
      myl = [ptr.value]
      while ptr.next != None:
        ptr = ptr.next
        myl = myl + [ptr.value]
      return(str(myl))

  ################################################
  # Complete the definition of Deque below this #
  ################################################
  def insertrear(self,v):
    if self.isempty():
        self.head.value = v
    else:
        newnode = Node()
        newnode.value = v
        newnode.next = None
        node = Node()
        node = self.head
        while node.next != None:
          node = node.next
        node.next = newnode
        while self.tail.next!= None:
          self.tail = self.tail.next

  def deletefront(self):
    if self.isempty():
        return
    if self.head.next == None:
        v = self.head.value
        self.head = Node()
        self.tail = Node()
        return v
    else:
        v = self.head.value
        self.head = self.head.next
        return v

  def deleterear(self):
    if self.isempty():
        return
    if self.head.next == None:
        v = self.head.value
        self.head = Node()
        self.tail = Node()
        return v
    else:
        node = Node()
        prev = Node()
        node = self.head
        while(node.next!=None):
          prev = node
          node = node.next
        v = node.value
        self.tail = prev
        self.tail.next = None
        return v
import ast

def parse(inp):
  inp = ast.literal_eval(inp)
  return (inp)

fncall = input().strip()
lparen = fncall.find("(")
rparen = fncall.rfind(")")
fname = fncall[:lparen]
```

```
farg = fncall[lparen+1:rparen]

if fname == "testdeque":
  arg = parse(farg)
  testdeque(arg)
else:
  print("Function", fname, "unknown")

```

Sample solutions (Provided by instructor)

```
def testdeque(l):
  mydeque = Deque()
  extractedlist = []
  for op in l:
    if op[0] == "if":
      mydeque.insertfront(op[1])
    elif op[0] == "ir":
      mydeque.insertrear(op[1])
    elif op[0] == "df":
      v = mydeque.deletefront()
      if v != None:
        extractedlist.append(v)
    elif op[0] == "dr":
      v = mydeque.deleterear()
      if v != None:
        extractedlist.append(v)
  print(mydeque,extractedlist,mydeque.head.value,mydeque.tail.value)
  return

##########


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

class Deque:
  def __init__(self):
    newnode = Node()
    self.head = newnode
    self.tail = newnode

  def isempty(self):
    return(self.head.value == None)

  def insertfront(self,v):
    if self.isempty():
      self.head.value = v
    else:
      newnode = Node()
      newnode.value = self.head.value
      newnode.next = self.head.next
      self.head.value = v
      self.head.next = newnode
      self.tail = newnode
      while self.tail.next != None:
        self.tail = self.tail.next

  def __str__(self):
    if self.head.value == None:
      return(str([]))
    else:
      ptr = self.head
      myl = [ptr.value]
      while ptr.next != None:
        ptr = ptr.next
        myl = myl + [ptr.value]
      return(str(myl))

  ################################################
  # Complete the definition of Deque below this #
  ################################################
  def insertrear(self,v):
    if self.isempty():
      self.head.value = v
```

```
    else:
      newnode = Node()
      newnode.value = v
      newnode.next = None
      self.tail.next = newnode
      self.tail = newnode

  def deletefront(self):
    if self.isempty():
      return

    retval = self.head.value

    if self.head.next == None:
      self.head.value = None
    else:
      self.head = self.head.next

    return(retval)

  def deleterear(self):
    if self.isempty():
      return

    retval = self.tail.value

    if self.head.next == None:
      self.head.value = None
      return(retval)

    ptr = self.head
    while ptr.next != self.tail:
      ptr = ptr.next

    ptr.next = None
    self.tail = ptr

    return(retval)

##########

import ast

def parse(inp):
  inp = ast.literal_eval(inp)
  return (inp)

fncall = input().strip()
lparen = fncall.find("(")
rparen = fncall.rfind(")")
fname = fncall[:lparen]
farg = fncall[lparen+1:rparen]

if fname == "testdeque":
  arg = parse(farg)
  testdeque(arg)
else:
  print("Function", fname, "unknown")

```