day01-Code

快速实现顺序栈模型

```python
"""
python实现栈
思路:
1. 栈的特点: 一端进行插入和删除操作
2. 实现:     可使用列表，列表尾部作为栈顶（进行插入和删除操作），列表头部作为栈底，不做任何操作
"""

class Stack:
    def __init__(self):
        """创建一个空栈"""
        self.stack = []

    def push(self,value):
        """入栈操作: 相当于在列表尾部进行元素添加"""
        self.stack.append(value)

    def pop(self):
        """出栈操作: 相当于在列表尾部弹出1个元素，考虑到空栈的情况"""
        if self.stack == []:
            raise Exception('pop from empty stack')
        else:
            return self.stack.pop()

    def is_empty(self):
        """判断栈是否为空"""
        if self.stack == []:
            return True
        return False

    def top(self):
        """查看栈顶元素，并非出栈"""
        if self.stack:
            return self.stack[-1]
        print('stack is empty')

    def size(self):
        """返回栈的大小"""
        return len(self.stack)

if __name__ == '__main__':
    s = Stack()
    # 此时为空栈，返回 True
    print(s.is_empty())
    # 此时栈中元素为:  100 200 300 , 100在栈底, 300在栈顶
    s.push(100)
    s.push(200)
    s.push(300)
    # 从栈顶弹出1个元素, 即 300
    print(s.pop())
    # 此时栈不为空, 返回 False
    print(s.is_empty())
    # 返回栈顶元素: 200
    print(s.top())
    # 返回栈的大小: 2
    print(s.size())
```

快速实现顺序队列模型

```python
"""
python实现队列
思路
1. 队列特点: 先进先出, 队尾入队, 队头出队
2. 思路:      可使用列表实现, 列表尾部作为队尾进行入队操作, 列表头部作为队头进行出队操作
"""

class Queue:
    def __init__(self):
        """创建一个空队列"""
        self.queue = []

    def enqueue(self,value):
        """入队列: 从列表尾部添加元素"""
        self.queue.append(value)

    def dequeue(self):
        """出队列: 从列表头部弹出元素, 考虑队列为空时的特殊情况"""
        if self.queue == []:
            raise Exception('dequeue from empty queue')
        return self.queue.pop(0)

    def is_empty(self):
        """判断队列是否为空"""
        if self.queue == []:
            return True
        return False

    def top(self):
        """查看队头元素, 考虑为空队列的情况, """
        if self.queue:
            return self.queue[0]
        else:
            raise Exception('queue is empty')

    def travel(self):
        """遍历整个队列, 从对头到队尾输出"""
        for i in self.queue:
                print(i,end=" ")

        print()

if __name__ == '__main__':
    q = Queue()
    # 此时为空队列，返回 True
    print(q.is_empty())
    # 此时队列中元素为 ： 100  200  300
    q.enqueue(100)
    q.enqueue(200)
    q.enqueue(300)
    # 队头出队列，结果为：100
    print(q.dequeue())
    # 此时队列不为空，返回：False
    print(q.is_empty())
    # 获取队头元素，结果为：200
    print(q.top())
    # 队头到队尾元素：200 300
    q.travel()
```

快速实现单链表

```python
"""
python实现单链表
思路：
1、节点类：数据区、指针区两个属性
2、链表类：实现链表的 增加、删除、遍历、判断是否为空等功能
"""

class Node:
    """节点类"""
    def __init__(self,elem,next=None):
        self.elem = elem
        self.next = next

class SingleList:
    """链表类"""
    def __init__(self,node=None):
        """创建链表存储空间，创建链表时给元素了，则为非空链表，反之为空链表"""
        self.head = node

    def is_empty(self):
        """判断链表是否为空"""
        if self.head is None:
            return True
        return False

    def add(self,value):
        """在链表头部添加元素
           1. value节点的指针指向头节点
           2. 把value节点设置为头节点
        """
```

```python
            node = Node(value)
            node.next = self.head
            self.head = node

    def append(self,value):
        """在链表尾部添加元素
           1. 找到尾节点，把尾节点的next指向value节点
           2. 把value的节点的next指向None
        """
        node = Node(value)
        if self.is_empty():
            self.head = node
        else:
            current = self.head
            # 循环完成后，current指向尾节点
            while current.next:
                current = current.next
            current.next = node
            node.next = None

    def travel(self):
        """遍历链表
           1.找到头节点，依次往后遍历，打印输出即可（考虑空链表的情况）
        """
        if self.is_empty():
            return
        else:
            current = self.head
            while current:
                print(current.elem,end=" ")
                current = current.next

            print()

    def length(self):
        """获取链表长度：从头到尾遍历即可"""
        if self.is_empty():
            return 0
        count = 0
        current = self.head
        while current:
            current = current.next
            count += 1

        return count

    def get_value(self,position):
        """获取指定下标的元素值"""
        number = self.length()
        if position < 0 or position > (number-1):
            raise Exception('index out of range')
        count = 0
        current = self.head
        while current:
            current = current.next
            count += 1
            if count == position:
                return current.elem


if __name__ == '__main__':
    s = SingleList()
    # 此时为空链表，返回 True
    print(s.is_empty())
    # 链表头部添加2个元素，则结果：100 200
    s.add(200)
    s.add(100)
    # 链表尾部添加2个元素，则结果：100 200 300 400
    s.append(300)
    s.append(400)
    # 遍历链表，则结果： 100 200 300 400
    s.travel()
    # 获取链表长度，结果： 4
    print(s.length())
    # 获取下表索引为2的，即第三个元素：300
    print(s.get_value(2))
```

快速实现单向循环链表

```python
"""
python实现单向循环链表
思路：
1、节点类：数据区、指针区两个属性
2、链表类：实现链表的 增加、删除、遍历、判断是否为空等功能
3、单向循环链表特点：尾节点指向头节点
"""

class Node:
    """节点类"""
    def __init__(self,elem,next=None):
        self.elem = elem
        self.next = next

class SingleList:
    """链表类"""
    def __init__(self,node=None):
        """创建链表存储空间，创建链表时给元素了，则为非空链表，反之为空链表"""
        self.head = node
        if node:
            node.next = node

    def is_empty(self):
        """判断链表是否为空"""
        if self.head is None:
            return True
        return False

    def add(self,value):
        """在链表头部添加元素
           1. value节点的指针指向头节点
```

```python
            2. 把value重新设置成头节点
            3. 把尾节点指向value节点
        """
        node = Node(value)
        if self.is_empty():
            self.head = node
            node.next = node
        else:
            current = self.head
            while current.next != self.head:
                current = current.next


            node.next = self.head
            self.head = node
            current.next = node

    def append(self,value):
        """在链表尾部添加元素
           1. 找到尾节点，把尾节点的next指向value节点
           2. 把value的节点的next指向头节点
        """
        node = Node(value)
        if self.is_empty():
            self.head = node
        else:
            current = self.head
            # 循环完成后，current指向尾节点
            while current.next != self.head:
                current = current.next

            current.next = node
            node.next = self.head

    def travel(self):
        """遍历链表
           1.找到头节点，依次往后遍历，打印输出即可（考虑空链表的情况）
        """
        if self.is_empty():
            return
        else:
            current = self.head
            while current.next != self.head:
                print(current.elem,end=" ")
                current = current.next

            # 退出循环，current指向尾节点但是并未打印
            print(current.elem)

    def length(self):
        """获取链表长度：从头到尾遍历即可"""
        if self.is_empty():
            return 0
        count = 1
        current = self.head
        while current.next != self.head:
            current = current.next
```

```python
                count += 1

            return count

        def get_value(self,position):
            """获取指定下标的元素值"""
            number = self.length()
            if position < 0 or position > (number-1):
                raise Exception('index out of range')
            count = 0
            current = self.head
            while current.next != self.head:
                current = current.next
                count += 1
                if count == position:
                    return current.elem


if __name__ == '__main__':
    s = SingleList()
    # 此时为空链表，返回 True
    print(s.is_empty())
    # 链表头部添加2个元素，则结果：100 200
    s.add(200)
    s.add(100)
    # 链表尾部添加2个元素，则结果：100 200 300 400
    s.append(300)
    s.append(400)
    # 遍历链表，则结果： 100 200 300 400
    s.travel()
    # 获取链表长度，结果： 4
    print(s.length())
    # 获取下表索引为2的，即第三个元素：300
    print(s.get_value(2))
```

快速实现链式栈模型

```python
"""
使用链式存储实现栈
思路：
1、栈特点：后进先出，所有操作只能在栈顶
2、封装方法：入栈 出栈 栈空 栈顶元素
3、链表的开头作为栈顶
"""

# 创建节点类
class Node:
    def __init__(self,val):
        self.val = val
        self.next = None

# 链式栈
class LinkStack:
    def __init__(self):
        # 标记顶位置，创建一个空栈，链表头部作为栈顶
        self.top = None

    def is_empty(self):
        """判断是否为空栈，空栈返回True，反之返回False"""
        if self.top is None:
            return True
        return False

    def push(self,val):
        """入栈：相当于在链表头部添加节点"""
        node = Node(val)
        node.next = self.top
        self.top = node

    def pop(self):
        """出栈：相当于删除头节点"""
        if self.top is None:
            raise Exception("pop from empty stack")

        value = self.top.val
        self.top = self.top.next

        return value

    def stack_top(self):
        """查看栈顶元素：查看头节点"""
        if self.top is None:
            raise Exception("Stack is empty")

        return self.top.val

    def size(self):
        if self.top is None:
            return 0
        count = 0
        current = self.top
        while current != None:
            current = current.next
            count += 1

        return count

if __name__ == '__main__':
    ls = LinkStack()
    # 入栈后，从栈顶到栈底以此为：300 200 100
    ls.push(100)
    ls.push(200)
    ls.push(300)
    # 出栈：从栈顶出栈  300
    print(ls.pop())
    # 查看栈顶元素：  200
    print(ls.stack_top())
    # 获取栈大小：  2
    print(ls.size())
```

快速实现链式队列模型

```python
"""
如何用链表实现队列
思路：
1、队列特点：先进先出，队尾进，队头出
2、实现    ： 使用单链表实现 尾部添加节点（入队），删除头节点（出队）等操作
"""

class Node:
    """节点类，包含数据区和指针区两个属性"""
    def __init__(self,elem):
        self.elem = elem
        self.next = None

class Queue:
    def __init__(self,node=None):
        """创建一个队列（链表）"""
        self.head = node

    def is_empty(self):
        """判断队列是否为空：头节点为空则一定为空队列"""
        if self.head is None:
            return True
        return False

    def enqueue(self,value):
        """入队列：从链表尾部添加一个节点"""
        node = Node(value)
        if self.is_empty():
            self.head = node
        else:
            current = self.head
            while current.next:
                current = current.next

            current.next = node
            node.next = None

    def dequeue(self):
        """出队列：获取链表头节点，并指向新的头"""

        if self.is_empty():
            raise Exception('queue is empty')

        result = self.head.elem
        self.head = self.head.next

        return result

    def top(self):
        """查看队头元素：查看self.head的元素值"""
        if self.is_empty():
            raise Exception('queue is empty')

        return self.head.elem

    def travel(self):
        """遍历整个队列，从队头到队尾输出"""
        current = self.head
        while current:
            print(current.elem,end=" ")
            current = current.next

        print()

if __name__ == '__main__':
    q = Queue()
    # 空队列，返回 True:
    print(q.is_empty())
    # 入队列: 100 200 300
    q.enqueue(100)
    q.enqueue(200)
    q.enqueue(300)
    # 出队列: 100
    print(q.dequeue())
    # 此时队列不为空，返回: False
    print(q.is_empty())
    # 查看队头元素:  200
    print(q.top())
    # 遍历整个队列: 200 300
    q.travel()
```