

day01-Code

快速实现顺序栈模型

```
1  """
2  python实现栈
3  思路:
4  1. 栈的特点: 一端进行插入和删除操作
5  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
6  """
7
8  class Stack:
9      def __init__(self):
10         """创建一个空栈"""
11         self.stack = []
12
13     def push(self,value):
14         """入栈操作: 相当于在列表尾部进行元素添加"""
15         self.stack.append(value)
16
17     def pop(self):
18         """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"""
19         if self.stack == []:
20             raise Exception('pop from empty stack')
21         else:
22             return self.stack.pop()
23
24     def is_empty(self):
25         """判断栈是否为空"""
26         if self.stack == []:
27             return True
28         return False
29
30     def top(self):
31         """查看栈顶元素, 并非出栈"""
32         if self.stack:
33             return self.stack[-1]
34         print('stack is empty')
35
36     def size(self):
37         """返回栈的大小"""
38         return len(self.stack)
39
40 if __name__ == '__main__':
41     s = Stack()
42     # 此时为空栈, 返回 True
43     print(s.is_empty())
```

```

44     # 此时栈中元素为: 100 200 300 , 100在栈底, 300在栈顶
45     s.push(100)
46     s.push(200)
47     s.push(300)
48     # 从栈顶弹出1个元素, 即 300
49     print(s.pop())
50     # 此时栈不为空, 返回 False
51     print(s.is_empty())
52     # 返回栈顶元素: 200
53     print(s.top())
54     # 返回栈的大小: 2
55     print(s.size())

```

快速实现顺序队列模型

```

1  """
2  python实现队列
3  思路
4  1. 队列特点: 先进先出, 队尾入队, 队头出队
5  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
6  """
7
8  class Queue:
9      def __init__(self):
10         """创建一个空队列"""
11         self.queue = []
12
13     def enqueue(self, value):
14         """入队列: 从列表尾部添加元素"""
15         self.queue.append(value)
16
17     def dequeue(self):
18         """出队列: 从列表头部弹出元素, 考虑队列为空时的特殊情况"""
19         if self.queue == []:
20             raise Exception('dequeue from empty queue')
21         return self.queue.pop(0)
22
23     def is_empty(self):
24         """判断队列是否为空"""
25         if self.queue == []:
26             return True
27         return False
28
29     def top(self):
30         """查看队头元素, 考虑为空队列的情况, """
31         if self.queue:
32             return self.queue[0]
33         else:
34             raise Exception('queue is empty')
35
36     def travel(self):
37         """遍历整个队列, 从对头到队尾输出"""
38         for i in self.queue:

```

```

39         print(i,end=" ")
40
41     print()
42
43 if __name__ == '__main__':
44     q = Queue()
45     # 此时为空队列, 返回 True
46     print(q.is_empty())
47     # 此时队列中元素为 : 100 200 300
48     q.enqueue(100)
49     q.enqueue(200)
50     q.enqueue(300)
51     # 队头出队列, 结果为: 100
52     print(q.dequeue())
53     # 此时队列不为空, 返回: False
54     print(q.is_empty())
55     # 获取队头元素, 结果为: 200
56     print(q.top())
57     # 队头到队尾元素: 200 300
58     q.travel()

```

快速实现单链表

```

1  """
2  python实现单链表
3  思路:
4  1、节点类: 数据区、指针区两个属性
5  2、链表类: 实现链表的 增加、删除、遍历、判断是否为空等功能
6  """
7
8  class Node:
9      """节点类"""
10     def __init__(self,elem,next=None):
11         self.elem = elem
12         self.next = next
13
14     class SingleList:
15         """链表类"""
16         def __init__(self,node=None):
17             """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"""
18             self.head = node
19
20         def is_empty(self):
21             """判断链表是否为空"""
22             if self.head is None:
23                 return True
24             return False
25
26         def add(self,value):
27             """在链表头部添加元素
28             1. value节点的指针指向头节点
29             2. 把value节点设置为头节点
30             """

```

```

31     node = Node(value)
32     node.next = self.head
33     self.head = node
34
35 def append(self,value):
36     """在链表尾部添加元素
37     1. 找到尾节点, 把尾节点的next指向value节点
38     2. 把value的节点的next指向None
39     """
40     node = Node(value)
41     if self.is_empty():
42         self.head = node
43     else:
44         current = self.head
45         # 循环完成后, current指向尾节点
46         while current.next:
47             current = current.next
48         current.next = node
49         node.next = None
50
51 def travel(self):
52     """遍历链表
53     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)
54     """
55     if self.is_empty():
56         return
57     else:
58         current = self.head
59         while current:
60             print(current.elem,end=" ")
61             current = current.next
62
63         print()
64
65 def length(self):
66     """获取链表长度: 从头到尾遍历即可"""
67     if self.is_empty():
68         return 0
69     count = 0
70     current = self.head
71     while current:
72         current = current.next
73         count += 1
74
75     return count
76
77 def get_value(self,position):
78     """获取指定下标的元素值"""
79     number = self.length()
80     if position < 0 or position > (number-1):
81         raise Exception('index out of range')
82     count = 0
83     current = self.head
84     while current:
85         current = current.next
86         count += 1
87         if count == position:

```

```

88         return current.elem
89
90
91 if __name__ == '__main__':
92     s = SingleList()
93     # 此时为空链表, 返回 True
94     print(s.is_empty())
95     # 链表头部添加2个元素, 则结果: 100 200
96     s.add(200)
97     s.add(100)
98     # 链表尾部添加2个元素, 则结果: 100 200 300 400
99     s.append(300)
100    s.append(400)
101    # 遍历链表, 则结果: 100 200 300 400
102    s.travel()
103    # 获取链表长度, 结果: 4
104    print(s.length())
105    # 获取下标索引为2的, 即第三个元素: 300
106    print(s.get_value(2))

```

快速实现单向循环链表

```

1  """
2  python实现单向循环链表
3  思路:
4  1、节点类: 数据区、指针区两个属性
5  2、链表类: 实现链表的 增加、删除、遍历、判断是否为空等功能
6  3、单向循环链表特点: 尾节点指向头节点
7  """
8
9  class Node:
10     """节点类"""
11     def __init__(self, elem, next=None):
12         self.elem = elem
13         self.next = next
14
15  class SingleList:
16     """链表类"""
17     def __init__(self, node=None):
18         """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"""
19         self.head = node
20         if node:
21             node.next = node
22
23     def is_empty(self):
24         """判断链表是否为空"""
25         if self.head is None:
26             return True
27         return False
28
29     def add(self, value):
30         """在链表头部添加元素
31         1. value节点的指针指向头节点

```

```

32         2. 把value重新设置成头节点
33         3. 把尾节点指向value节点
34     """
35     node = Node(value)
36     if self.is_empty():
37         self.head = node
38         node.next = node
39     else:
40         current = self.head
41         while current.next != self.head:
42             current = current.next
43
44         node.next = self.head
45         self.head = node
46         current.next = node
47
48 def append(self,value):
49     """在链表尾部添加元素
50         1. 找到尾节点, 把尾节点的next指向value节点
51         2. 把value的节点的next指向头节点
52     """
53     node = Node(value)
54     if self.is_empty():
55         self.head = node
56     else:
57         current = self.head
58         # 循环完成后, current指向尾节点
59         while current.next != self.head:
60             current = current.next
61
62         current.next = node
63         node.next = self.head
64
65 def travel(self):
66     """遍历链表
67         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)
68     """
69     if self.is_empty():
70         return
71     else:
72         current = self.head
73         while current.next != self.head:
74             print(current.elem,end=" ")
75             current = current.next
76
77         # 退出循环, current指向尾节点但是并未打印
78         print(current.elem)
79
80 def length(self):
81     """获取链表长度: 从头到尾遍历即可"""
82     if self.is_empty():
83         return 0
84     count = 1
85     current = self.head
86     while current.next != self.head:
87         current = current.next

```

```

89         count += 1
90
91     return count
92
93     def get_value(self, position):
94         """获取指定下标的元素值"""
95         number = self.length()
96         if position < 0 or position > (number-1):
97             raise Exception('index out of range')
98         count = 0
99         current = self.head
100        while current.next != self.head:
101            current = current.next
102            count += 1
103            if count == position:
104                return current.elem
105
106
107 if __name__ == '__main__':
108     s = SingleList()
109     # 此时为空链表, 返回 True
110     print(s.is_empty())
111     # 链表头部添加2个元素, 则结果: 100 200
112     s.add(200)
113     s.add(100)
114     # 链表尾部添加2个元素, 则结果: 100 200 300 400
115     s.append(300)
116     s.append(400)
117     # 遍历链表, 则结果: 100 200 300 400
118     s.travel()
119     # 获取链表长度, 结果: 4
120     print(s.length())
121     # 获取下标索引为2的, 即第三个元素: 300
122     print(s.get_value(2))

```

快速实现链式栈模型

```

1  """
2  使用链式存储实现栈
3  思路:
4  1、栈特点: 后进先出, 所有操作只能在栈顶
5  2、封装方法: 入栈 出栈 栈空 栈顶元素
6  3、链表的开头作为栈顶
7  """
8
9  # 创建节点类
10 class Node:
11     def __init__(self, val):
12         self.val = val
13         self.next = None
14
15 # 链式栈
16 class LinkStack:

```

```

17 def __init__(self):
18     # 标记顶位置, 创建一个空栈, 链表头部作为栈顶
19     self.top = None
20
21 def is_empty(self):
22     """判断是否为空栈, 空栈返回True, 反之返回False"""
23     if self.top is None:
24         return True
25     return False
26
27 def push(self, val):
28     """入栈: 相当于在链表头部添加节点"""
29     node = Node(val)
30     node.next = self.top
31     self.top = node
32
33 def pop(self):
34     """出栈: 相当于删除头节点"""
35     if self.top is None:
36         raise Exception("pop from empty stack")
37
38     value = self.top.val
39     self.top = self.top.next
40
41     return value
42
43 def stack_top(self):
44     """查看栈顶元素: 查看头节点"""
45     if self.top is None:
46         raise Exception("Stack is empty")
47
48     return self.top.val
49
50 def size(self):
51     if self.top is None:
52         return 0
53     count = 0
54     current = self.top
55     while current != None:
56         current = current.next
57         count += 1
58
59     return count
60
61 if __name__ == '__main__':
62     ls = LinkStack()
63     # 入栈后, 从栈顶到栈底以此为: 300 200 100
64     ls.push(100)
65     ls.push(200)
66     ls.push(300)
67     # 出栈: 从栈顶出栈 300
68     print(ls.pop())
69     # 查看栈顶元素: 200
70     print(ls.stack_top())
71     # 获取栈大小: 2
72     print(ls.size())

```


快速实现链式队列模型

```
1  """
2  如何用链表实现队列
3  思路：
4  1、队列特点：先进先出，队尾进，队头出
5  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
6  """
7
8  class Node:
9      """节点类，包含数据区和指针区两个属性"""
10     def __init__(self, elem):
11         self.elem = elem
12         self.next = None
13
14     class Queue:
15         def __init__(self, node=None):
16             """创建一个队列（链表）"""
17             self.head = node
18
19         def is_empty(self):
20             """判断队列是否为空：头节点为空则一定为空队列"""
21             if self.head is None:
22                 return True
23             return False
24
25         def enqueue(self, value):
26             """入队列：从链表尾部添加一个节点"""
27             node = Node(value)
28             if self.is_empty():
29                 self.head = node
30             else:
31                 current = self.head
32                 while current.next:
33                     current = current.next
34
35                 current.next = node
36                 node.next = None
37
38         def dequeue(self):
39             """出队列：获取链表头节点，并指向新的头"""
40
41             if self.is_empty():
42                 raise Exception('queue is empty')
43
44             result = self.head.elem
45             self.head = self.head.next
46
47             return result
48
49         def top(self):
```

```
50         """查看队头元素: 查看self.head的元素值"""
51         if self.is_empty():
52             raise Exception('queue is empty')
53
54         return self.head.elem
55
56     def travel(self):
57         """遍历整个队列, 从队头到队尾输出"""
58         current = self.head
59         while current:
60             print(current.elem,end=" ")
61             current = current.next
62
63         print()
64
65 if __name__ == '__main__':
66     q = Queue()
67     # 空队列, 返回 True:
68     print(q.is_empty())
69     # 入队列: 100 200 300
70     q.enqueue(100)
71     q.enqueue(200)
72     q.enqueue(300)
73     # 出队列: 100
74     print(q.dequeue())
75     # 此时队列不为空, 返回: False
76     print(q.is_empty())
77     # 查看队头元素: 200
78     print(q.top())
79     # 遍历整个队列: 200 300
80     q.travel()
```