Day02 - 数据结构

数据结构分类

- **线性结构 - N个数据元素的有限序列**

```
【1】顺序表
【2】链表
【3】栈 - 后进先出（LIFO）：栈顶进行 '入栈' 和 '出栈' 操作
【4】队列 - 先进先出（FIFO）：'队尾'进行入队，'队头'进行出队
```

- **非线性结构 - 一个结点元素可能有多个直接前驱和多个直接后继**

```
【1】集合
 1.1> 特点：  集合中任何两个数据元素之间都没有逻辑关系,组织形式松散

【2】树形结构
 2.1> 特点：树形结构具有分支、层次特性,其形态有点象自然界中的树
 2.2> 几个定义
      2.2.1> 树根          '没有父节点的节点'
      2.2.2> 节点的度       '一个节点的子树的个数'
      2.2.3> 节点的层次     '从根开始定义起，根为第1层'
      2.2.4> 树的深度       '树中节点的最大层次'
 2.3> 二叉树特点
      2.3.1> n个节点的有限集合
      2.3.2> 由根节点即左子树和右子树组成
      2.3.3> 严格区分左孩子和右孩子
 2.4> 二叉树的遍历
      2.4.1> 广度遍历 - 一层一层遍历，如何实现？--可利用队列
      2.4.2> 深度遍历
             1）前序遍历 ：根、左、右
             2）中序遍历 ：左、根、右
             3）后序遍历 ：左、右、根
 2.5> '用Python实现二叉树'


【3】图状结构
 3.1> 特点：图状结构中的结点按逻辑关系互相缠绕,任何两个结点都可以邻接
```

练习题1 - 链表

- **题目描述+试题解析**

```
【1】题目描述
    输入一个链表，按链表值从尾到头的顺序返回一个 ArrayList

【2】试题解析
    将链表的每个值取出来然后存放到一个列表 ArrayList 中
    解题思路1：将链表中从头节点开始依次取出节点元素，append到array_list中，并进行最终反转
    解题思路2：将链表中从头节点开始依次取出节点元素，insert到array_list中的第1个位置
```

- **代码实现 - 方法1**

```python
"""
输入一个链表，按链表值从尾到头的顺序返回一个 ArrayList
"""

class Node:
    """链表节点类"""
    def __init__(self,x):
        self.val = x
        self.next = None

class Solution:
    # 返回从尾部到头部的序列，node为头节点
    def get_list_from_tail_to_head(self,node):
        array_list = []
        while node:
            array_list.insert(0,node.val)
            node = node.next

        return array_list

if __name__ == '__main__':
      s = Solution()
      # 100 200 300
      n1 = Node(100)
      n1.next = Node(200)
      n1.next.next = Node(300)
      # 反转返回数组： [ 300, 200, 100 ]
      array_list = s.get_list_from_tail_to_head(n1)
      print(array_list)
```

- **代码实现 - 方法2**

```python
"""
输入一个链表，按链表值从尾到头的顺序返回一个 ArrayList
"""

class Node:
    """链表节点类"""
    def __init__(self,x):
        self.val = x
        self.next = None

class Solution:
    # 返回从尾部到头部的序列，node为头节点
    def get_list_from_tail_to_head(self,node):
```

```
        array_list = []
        while node:
            array_list.append(node.val)
            node = node.next
        # 将最终列表进行反转,无返回值,直接改变列表
        array_list.reverse()

        return array_list

if __name__ == '__main__':
      s = Solution()
      # 100 200 300
      n1 = Node(100)
      n1.next = Node(200)
      n1.next.next = Node(300)
      # 反转返回数组:  [ 300, 200, 100 ]
      array_list = s.get_list_from_tail_to_head(n1)
      print(array_list)
```

练习题2 - 链表

- **题目描述+试题解析**

```
【1】题目描述
    输入一个链表，输出该链表中倒数第 k 个节点

【2】试题解析
    可将链表中的每一个元素保存到列表中，在列表中寻找倒数第 k 个元素
```

- **代码实现**

```
"""
输入一个链表，  输出该链表中倒数第 k 个结点
"""
class Node:
    def __init__(self,value,next=None):
        self.value = value
        self.next = next

class Solution:
    def find_k_to_tail(self,head,k):
        t_list = []
        while head:
            t_list.append(head.value)
            head = head.next

        if k > len(t_list) or k < 1:
            return None

        return t_list[-k]

if __name__ == '__main__':
```

```
    s = Solution()
    # 100 200 300
    n1 = Node(100)
    n1.next = Node(200)
    n1.next.next = Node(300)
    # 倒数第2个：200
    result = s.find_k_to_tail(n1,2)
    print(result)
```

练习3 - 链表

- **题目描述+试题解析**

```
【1】题目描述
    输入一个链表，反转链表后，输出新链表的表头

【2】试题解析
    可以将链表的每一个节点取出来，插入到新的链表表头，同时要保存原链表的下一个节点
```

- **代码实现**

```
"""
输入一个链表，反转链表后，输出新链表的表头
"""

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

class Solution:
    def reverse_list(self,head):
        """反转链表，从头节点依次遍历，存入另外一个链表"""
        # 空链表 或者 只有1个节点的链表
        if head is None or head.next is None:
            return head

        # 整体思路：当前节点的指针指向上一个节点
        # 记录当前节点
        current = head
        # 记录当前节点的前1个节点
        pre = None

        while current is not None:
            next_node = current.next
            # 1、当前节点的指针指向前1个节点
            current.next = pre
            # 2、前一个节点更新
            pre = current
            # 3、当前节点后移
            current = next_node
```

```
        return pre

if __name__ == '__main__':
    s = Solution()
    # 100->200->300->400
    n1 = Node(100)
    n1.next = Node(200)
    n1.next.next = Node(300)
    n1.next.next.next = Node(400)
    # 进行反转后获取头节点: 400
    print(s.reverse_list(n1).value)
```

练习4 - 字符串

- **题目描述+试题解析**

```
【1】题目描述
    请实现一个函数，将一个字符串中的每个空格替换成”%20”。例如，当字符串 为 We Are Family  则经过替换之后的字符串为 We%20Are%20Family

【2】试题解析
    a> 利用字符串的replace()方法
    b> 用法: string.replace(old,new[,max])
       old : 将被替换的子字符串
       new : 新字符串，用于替换 old 子字符串
       max : 可选,字符串替换不超过 max 次
```

- **代码实现**

```
"""
请实现一个函数，将一个字符串中的每个空格替换成”%20”。例如，当字符串 为 We Are Family  则经过替换之后的字符串为 We%20Are%20Family
"""
class Solution:
    # s 源字符串
    def replace_space(self,s):
        return s.replace(' ', '%20')

if __name__ == '__main__':
    s = Solution()
    string = 'We Are Family'
    result = s.replace_space(string)
    print(result)
```

练习5 - 字符串

- **题目描述+试题解析**

```
【1】题目描述
    牛客近来了一个新员工 Fish，每天早晨总是会拿着一本英文杂志，写些句 子在本子上。同事 Cat 对 Fish 写的内容颇感兴趣，有一天他向 Fish 借来翻看，但却读不懂它 的意思。如，”student a am I”。后来才意识到，这家伙原来把句子单词的顺序反转了，正确句子应该是”I am a student”。Cat对一一的反转这些单词顺序可不在行，你能帮助他吗?

【2】试题解析
   a> 先 split()
   b> 再 join()
```

- **代码实现**

```
"""
牛客近来了一个新员工 Fish，每天早晨总是会拿着一本英文杂志，写些句 子在本子上。同事 Cat 对 Fish 写的内容颇感兴趣，有一天他向 Fish 借来翻看，但却读不懂它 的意思。例如，”student a am I”。后来才意识到，这家伙原来把句子单词的顺序翻转了，正 确句子应该是”I am a student”。Cat 对一一的翻转这些单词顺序可不在行，你能帮助他吗?
"""

class Solution:
    def reverse_sentence(self,fish):
        fish_list = fish.split(' ')
        fish_list.reverse()
        cat = ' '.join(fish_list)

        return cat

if __name__ == '__main__':
    s = Solution()
    fish = 'student a am I'
    cat = s.reverse_sentence(fish)
    print(cat)
```

练习6 - 字符串

- **题目描述+试题解析**

```
【1】题目描述
    汇编语言汇总有一种移位指令叫做循环左移(ROL)，现在有个简单的任务，就 是用字符串模拟这个指令的运算结果。对于一个给定的字符序列 S，请你把循环左移 K 位后 的序列输出。例如，字符串序列 S=”abcXYZdef”，要求输出循环左移 3 位后的结果， 即”XYZdefabc”

【2】试题解析
   字符串切片
```

- **代码实现**

```
"""
汇编语言汇总有一种移位指令叫做循环左移(ROL)，现在有个简单的任务，就 是用字符串模拟这个指令的运算结果。对于一个给定的字符序列 S，请你把循环左移 K 位后 的序列输出。例如，字符串序列 S=”abcXYZdef”，要求输出循环左移 3 位后的结果， 即”XYZdefabc”。是不是很简单? OK，搞定它
```

```
"""

class Solution:
    def left_k_string(self,string,k):
        left_li = string[:k]
        right_li = string[k:]

        return right_li + left_li

if __name__ == '__main__':
    s = Solution()
    result_li = s.left_k_string('ABCDEFG',3)
    print(result_li)
```

题目7 - 链表

- **题目描述+试题解析**

```
【1】题目描述
    输入两个链表，找出它们的第一个公共节点

【2】试题解析
    如果有公共节点，则两个链表公共节点后面的节点一定完全相同，因为节点有数据区和指针区，而next只能指向1个节点

    思路:
        stack1 = []     存储第1个链表中的节点
        stack2 = []     存储第2个链表中的节点

        两边同时pop，后面节点一定相同，一直找到最后1个相同的节点即可
```

- **代码实现**

```
"""
输入两个链表，找出它们的第一个公共节点
"""

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

class Solution:
    def get_first_same_node(self,head1,head2):
        # 用来存放两个链表中的所有节点 - 入栈（append）
        stack_1 = []
        stack_2 = []

        while head1:
            stack_1.append(head1)
            head1 = head1.next

```

```
        while head2:
            stack_2.append(head2)
            head2 = head2.next

        node = None
        # 两个栈不为空，且最后1个节点相同，则弹出继续往前找
        while stack_1 and stack_2 and stack_1[-1] is stack_2[-1]:
            node = stack_1.pop()
            stack_2.pop()

        return node

if __name__ == '__main__':
    s = Solution()
    # 定义几个节点
    p1 = Node(100)
    p2 = Node(200)
    p3 = Node(300)
    p4 = Node(400)
    p5 = Node(800)
    p6 = Node(900)
    # 链表1: 100 200 300 400
    p1.next = p2
    p2.next = p3
    p3.next = p4
    # 链表2: 800 900 300 400
    p5.next = p6
    p6.next = p3
    p3.next = p4
    # 第一个公共的节点:  300
    node = s.get_first_same_node(p1,p5)
    print(node.value)
```

练习8 - 链表

- **题目描述+试题解析**

```
【1】题目描述
    输入两个单调递增的链表，输出两个链表合成后的链表，当然我们需要合成后的链表满足单调不减规则

【2】试题解析
    a> 比较两个链表的头节点，确认合成后链表的头节点
    b> 继续依次比较两个链表元素的大小，将元素小的结点插入到新的链表中，直到一个链 表为空
```

- **代码实现 - 非递归**

```
"""
从头开始比较两个链表元素的大小，将元素小的结点插入到新的链表中，直到一个链 表为空
"""

class Node:
    def __init__(self,value):
```

```
        self.value = value
        self.next = None

class Solution:
    def merge_link_list(self,head1,head2):
        # 锁定新链表的链表头
        if head1 and head2:
            if head1.value < head2.value:
                merge_head = head1
                head1 = head1.next
            else:
                merge_head = head2
                head2 = head2.next
            p = merge_head
        elif head1:
            return head1
        else:
            return head2

        # 开始遍历两个链表比较
        while head1 and head2:
            if head1.value >= head2.value:
                merge_head.next = head2
                head2 = head2.next
            else:
                merge_head.next = head1
                head1 = head1.next
            merge_head = merge_head.next


        # 循环执行完成后一定有1个head为None了
        if head1:
            merge_head.next = head1
        elif head2:
            merge_head.next = head2

        # 返回合并后新链表的头节点
        return p

if __name__ == '__main__':
    s = Solution()
    # 递增链表1: 100 200 300 400
    head1 = Node(100)
    head1.next = Node(200)
    head1.next.next = Node(300)
    head1.next.next.next = Node(400)
    # 递增链表2: 1 2 3
    head2 = Node(200)
    head2.next = Node(250)
    head2.next.next = Node(360)
    # 合并后返回头节点: 1
    new_head = s.merge_link_list(head1,head2)
    # 合并后结果: 1 2 3 100 200 300 400
    while new_head:
        print(new_head.value,end=" ")
        new_head = new_head.next

```

```
64      print()
```