

第 1 章



入门 100 例

► 例 1 反转一个 3 位整数

1. 问题描述

反转一个只有 3 位数的整数。

2. 问题示例

输入 number = 123, 输出 321; 输入 number = 900, 输出 9。

3. 代码实现

```
class Solution:  
    # 参数 number: 一个 3 位整数  
    # 返回值: 反转后的数字  
    def reverseInteger(self, number):  
        h = int(number/100)  
        t = int(number % 100/10)  
        z = int(number % 10)  
        return(100 * z + 10 * t + h)  
  
    # 主函数  
    if __name__ == '__main__':  
        solution = Solution()  
        num = 123  
        ans = solution.reverseInteger(num)  
        print("输入:", num)  
        print("输出:", ans)
```

4. 运行结果

输入: 123

输出: 321

► 例 2 合并排序数组

1. 问题描述

合并两个升序的整数数组 A 和 B, 形成一个新的数组, 新数组也要有序。

2. 问题示例

输入 A = [1], B = [1], 输出 [1,1], 返回合并后的数组。输入 A = [1,2,3,4], B = [2,4,5,6], 输出 [1,2,2,3,4,4,5,6], 返回合并所有元素后的数组。

3. 代码实现

```
class Solution:
    # 参数 A: 有序整数数组 A
    # 参数 B: 有序整数数组 B
    # 返回: 一个新的有序整数数组
    def mergeSortedArray(self, A, B):
        i, j = 0, 0
        C = []
        while i < len(A) and j < len(B):
            if A[i] < B[j]:
                C.append(A[i])
                i += 1
            else:
                C.append(B[j])
                j += 1
        while i < len(A):
            C.append(A[i])
            i += 1
        while j < len(B):
            C.append(B[j])
            j += 1
        return C

    # 主函数
if __name__ == '__main__':
    A = [1, 4]
    B = [1, 2, 3]
    D = [1, 2, 3, 4]
    E = [2, 4, 5, 6]
    solution = Solution()
    print("输入:", A, " ", B)
    print("输出:", solution.mergeSortedArray(A, B))
    print("输入:", D, " ", E)
    print("输出:", solution.mergeSortedArray(D, E))
```

4. 运行结果

输入: [1, 4] [1, 2, 3]

```
输出: [1, 1, 2, 3, 4]
输入: [1, 2, 3, 4] [2, 4, 5, 6]
输出: [1, 2, 2, 3, 4, 4, 5, 6]
```

► 例 3 旋转字符串

1. 问题描述

给定一个字符串(以字符数组的形式)和一个偏移量,根据偏移量原地从左向右旋转字符串。

2. 问题示例

输入 str="abcdefg",offset = 3,输出"efgabcd"。输入 str="abcdefg",offset = 0,输出"abcdefg"。输入 str="abcdefg",offset = 1,输出"abcdefg",返回旋转后的字符串。输入 str="abcdefg",offset = 2,输出"fgabcde",返回旋转后的字符串。

3. 代码实现

```
class Solution:
    #参数 s:字符列表
    #参数 offset:整数
    #返回值:无
    def rotateString(self, s, offset):
        if len(s) > 0:
            offset = offset % len(s)
            temp = (s + s)[len(s) - offset : 2 * len(s) - offset]
            for i in range(len(temp)):
                s[i] = temp[i]
    #主函数
if __name__ == '__main__':
    s = ["a", "b", "c", "d", "e", "f", "g"]
    offset = 3
    solution = Solution()
    solution.rotateString(s, offset)
    print("输入:s =", ["a", "b", "c", "d", "e", "f", "g"], " ", "offset =", offset)
    print("输出:s =", s)
```

4. 运行结果

```
输入: s = ['a', 'b', 'c', 'd', 'e', 'f', 'g'] offset = 3
输出: s = ['e', 'f', 'g', 'a', 'b', 'c', 'd']
```

► 例 4 相对排名

1. 问题描述

根据 N 名运动员得分,找到相对等级和获得最高分前 3 名的人,分别获得金牌、银牌和

铜牌。 N 是正整数，并且不超过 10 000。所有运动员的成绩都保证是独一无二的。

2. 问题示例

输入 [5, 4, 3, 2, 1]，输出 ["Gold Medal", "Silver Medal", "Bronze Medal", "4", "5"]，前 3 名运动员得分较高，根据得分依次获得金牌、银牌和铜牌。对于后两名运动员，根据分数输出相对等级。

3. 代码实现

```
class Solution:
    # 参数 nums: 整数列表
    # 返回列表
    def findRelativeRanks(self, nums):
        score = {}
        for i in range(len(nums)):
            score[nums[i]] = i
        sortedScore = sorted(nums, reverse=True)
        answer = [0] * len(nums)
        for i in range(len(sortedScore)):
            res = str(i + 1)
            if i == 0:
                res = 'Gold Medal'
            if i == 1:
                res = 'Silver Medal'
            if i == 2:
                res = 'Bronze Medal'
            answer[score[sortedScore[i]]] = res
        return answer
# 主函数
if __name__ == '__main__':
    num = [5, 4, 3, 2, 1]
    s = Solution()
    print("输入:", num)
    print("输出:", s.findRelativeRanks(num))
```

4. 运行结果

```
输入: [5, 4, 3, 2, 1]
输出: ['Gold Medal', 'Silver Medal', 'Bronze Medal', '4', '5']
```

► 例 5 二分查找

1. 问题描述

给定一个排序的整数数组(升序)和一个要查找的目标整数 target，查找到 target 第 1 次出现的下标(从 0 开始)，如果 target 不存在于数组中，返回 -1。

2. 问题示例

输入数组[1,4,4,5,7,7,8,9,9,10]和目标整数1,输出其所在的位置为0,即第1次出现在第0个位置。输入数组[1, 2, 3, 3, 4, 5, 10]和目标整数3,输出2,即第1次出现在第2个位置。输入数组[1, 2, 3, 3, 4, 5, 10]和目标整数6,输出-1,即没有出现过6,返回-1。

3. 代码实现

```
class Solution:
    # 参数 nums: 整数数组
    # 参数 target: 要查找的目标数字
    # 返回值: 目标数字的第 1 个位置,从 0 开始
    def binarySearch(self, nums, target):
        return self.search(nums, 0, len(nums) - 1, target)
    def search(self, nums, start, end, target):
        if start > end:
            return -1
        mid = (start + end)//2
        if nums[mid] > target:
            return self.search(nums, start, mid, target)
        if nums[mid] == target:
            return mid
        if nums[mid] < target:
            return self.search(nums, mid, end, target)
    # 主函数
if __name__ == '__main__':
    my_solution = Solution()
    nums = [1,2,3,4,5,6]
    target = 3
    targetIndex = my_solution.binarySearch(nums, target)
    print("输入:nums =", nums, ", target =", target)
    print("输出:", targetIndex)
```

4. 运行结果

```
输入: nums = [1, 2, 3, 4, 5, 6]  target = 3
输出: 2
```

▶例 6 下一个更大的数

1. 问题描述

两个不重复的数组 nums1 和 nums2,其中 nums1 是 nums2 的子集。在 nums2 的相应位置找到 nums1 所有元素的下一个更大数字。

nums1 中的数字 x 的下一个更大数字是 nums2 中 x 右边第 1 个更大的数字。如果它

不存在，则为此数字输出 -1。nums1 和 nums2 中的所有数字都是唯一的，nums1 和 nums2 的长度不超过 1000。

2. 问题示例

输入 nums1 = [4,1,2], nums2 = [1,3,4,2]，输出 [-1,3,-1]。对于第 1 个数组中的数字 4，在第 2 个数组中找不到下一个更大的数字，因此输出 -1；对于第 1 个数组中的数字 1，第 2 个数组中的下一个更大数字是 3；对于第 1 个数组中的数字 2，第 2 个数组中没有下一个更大的数字，因此输出 -1。

3. 代码实现

```
class Solution:
    # 参数 nums1: 整数数组
    # 参数 nums2: 整数数组
    # 返回整数数组
    def nextGreaterElement(self, nums1, nums2):
        answer = {}
        stack = []
        for x in nums2:
            while stack and stack[-1] < x:
                answer[stack[-1]] = x
                del stack[-1]
            stack.append(x)
        for x in stack:
            answer[x] = -1
        return [answer[x] for x in nums1]
# 主函数
if __name__ == '__main__':
    s = Solution()
    nums1 = [4,1,2]
    nums2 = [1,3,4,2]
    print("输入 1:", nums1)
    print("输入 2:", nums2)
    print("输出 :", s.nextGreaterElement(nums1, nums2))
```

4. 运行结果

```
输入 1: [4, 1, 2]
输入 2: [1, 3, 4, 2]
输出: [-1, 3, -1]
```

► 例 7 字符串中的单词数

1. 问题描述

计算字符串中的单词数，其中一个单词定义为不含空格的连续字符串。

2. 问题示例

输入"Hello, my name is John",输出 5。

3. 代码实现

```
class Solution:
    # 参数 s: 字符串
    # 返回整数
    def countSegments(self, s):
        res = 0
        for i in range(len(s)):
            if s[i] != ' ' and (i == 0 or s[i - 1] == ' '):
                res += 1
        return res

# 主函数
if __name__ == '__main__':
    s = Solution()
    n = "Hello, my name is John"
    print("输入:", n)
    print("输出:", s.countSegments(n))
```

4. 运行结果

```
输入: Hello, my name is John
输出: 5
```

▶ 例 8 勒索信

1. 问题描述

给定一个表示勒索信内容的字符串和另一个表示杂志内容字符串,写一个方法判断能否通过剪下杂志中的内容构造出这封勒索信,若可以,返回 True,否则返回 False。注: 杂志字符串中的每一个字符仅能在勒索信中使用一次。

2. 问题示例

输入 ransomNote = "aa", magazine = "aab",输出 True,勒索信的内容可以从杂志内容剪辑而来。

3. 代码实现

```
class Solution:
    """
    参数 ransomNote: 字符串
    参数 magazine: 字符串
    返回布尔类型
    """
    def canConstruct(self, ransomNote, magazine):
```

```

arr = [0] * 26
for c in magazine:
    arr[ord(c) - ord('a')] += 1
for c in ransomNote:
    arr[ord(c) - ord('a')] -= 1
    if arr[ord(c) - ord('a')] < 0:
        return False
return True
# 主函数
if __name__ == '__main__':
    s = Solution()
    ransomNote = "aa"
    magazine = "aab"
    print("输入勒索信:", ransomNote)
    print("输入杂志内容:", magazine)
    print("输出:", s.canConstruct(ransomNote, magazine))

```

4. 运行结果

```

输入勒索信: aa
输入杂志内容: aab
输出: True

```

► 例 9 不重复的两个数

1. 问题描述

给定一个数组 $a[]$, 其中除了 2 个数, 其他均出现 2 次, 请找到不重复的 2 个数并返回。

2. 问题示例

给出 $a = [1, 2, 5, 5, 6, 6]$, 返回 $[1, 2]$, 除 1 和 2 外其他数都出现了 2 次, 因此返回 $[1, 2]$ 。给出 $a = [3, 2, 7, 5, 5, 7]$, 返回 $[2, 3]$, 除了 2 和 3 其他数都出现了 2 次, 因此返回 $[2, 3]$ 。

3. 代码实现

```

# 参数 arr: 输入的待查数组
# 返回值: 内容没有重复的两个值的列表
class Solution:
    def theTwoNumbers(self, a):
        ans = [0, 0]
        for i in a:
            ans[0] = ans[0] ^ i
        c = 1
        while c & ans[0] != c:
            c = c << 1
        for i in a:

```

```

        if i & c == c:
            ans[1] = ans[1] ^ i
        ans[0] = ans[0] ^ ans[1]
    return ans

if __name__ == '__main__':
    arr = [1, 2, 5, 1]
    solution = Solution()
    print("数组:", arr)
    print("两个没有重复的数字:", solution.theTwoNumbers(arr))

```

4. 运行结果

数组: [1, 2, 5, 1]
两个没有重复的数字是: [2, 5]

▶ 例 10 双胞胎字符串

1. 问题描述

给定两个字符串 s 和 t, 每次可以任意交换 s 的奇数位或偶数位上的字符, 即奇数位上的字符能与其他奇数位的字符互换, 偶数位上的字符也能与其他偶数位的字符互换, 问能否经过若干次交换, 使 s 变成 t。

2. 问题示例

输入为 s = "abcd", t = "cdab", 输出是"Yes", 第 1 次 a 与 c 交换, 第 2 次 b 与 d 交换。
输入 s = "abcd", t = "bcda", 输出是"No", 无论如何交换, 都无法得到 bcda。

3. 代码实现

```

# 参数 s 和 t: 一对字符串
# 返回值: 字符串, 表示能否根据规则转换
class Solution:
    def isTwin(self, s, t):
        if len(s) != len(t):
            return "No"
        oddS = []
        evenS = []
        oddT = []
        evenT = []
        for i in range(len(s)):
            if i & 1:
                oddS.append(s[i])
                oddT.append(t[i])
            else:
                evenS.append(s[i])
                evenT.append(t[i])

```

```

        oddS.sort()
        oddT.sort()
        evenS.sort()
        evenT.sort()
        for i in range(len(oddS)) :
            if oddS[i] != oddT[i]:
                return "No"
        for i in range (len(evenS)) :
            if evenS[i] != evenT[i]:
                return "No"
        return "Yes"
if __name__ == '__main__':
    s = "abcd"
    t = "cdab"
    solution = Solution()
    print("s 与 t 分别为:", s, t)
    print("是否为双胞胎:", solution.isTwin(s, t))

```

4. 运行结果

s 与 t 分别为: abcd cdab
是否为双胞胎: Yes

► 例 11 最接近 target 的值

1. 问题描述

给出一个数组，在数组中找到 2 个数，使得它们的和最接近但不超过目标值，返回它们的和。

2. 问题示例

输入 $\text{target} = 15, \text{array} = [1, 3, 5, 11, 7]$ ，输出 $14, 11 + 3 = 14$ 。输入 $\text{target} = 16$ 和 $\text{array} = [1, 3, 5, 11, 7]$ ，输出 $16, 11 + 5 = 16$ 。

3. 代码实现

```

# 参数 array: 输入列表
# 参数 target: 目标值
# 返回值是整数
class Solution:
    def closestTargetValue(self, target, array):
        n = len(array)
        if n < 2:
            return -1
        array.sort()
        diff = 0xffffffff

```

```

left = 0
right = n - 1
while left < right:
    if array[left] + array[right] > target:
        right -= 1
    else:
        diff = min(diff, target - array[left] - array[right])
        left += 1
if diff == 0xffffffff:
    return -1
else:
    return target - diff
if __name__ == '__main__':
    array = [1,3,5,11,7]
    target = 15
    solution = Solution()
    print(" 输入数组:", array, "目标值:", target)
    print(" 最近可以得到值:", solution.closestTargetValue(target, array))

```

4. 运行结果

输入数组: [1, 3, 5, 11, 7] 目标值: 15
 最近可以得到的值: 14

▶ 例 12 点积

1. 问题描述

给出 2 个数组,求它们的点积。

2. 问题示例

输入为 A = [1,1,1] 和 B = [2,2,2],输出为 $6, 1 * 2 + 1 * 2 + 1 * 2 = 6$ 。输入为 A = [3,2] 和 B = [2,3,3],输出为 -1,没有点积。

3. 代码实现

```

# 参数 A 和 B: 输入列表
# 返回值: 整数,是点积
class Solution:
    def dotProduct(self, A, B):
        if len(A) == 0 or len(B) == 0 or len(A) != len(B):
            return -1
        ans = 0
        for i in range(len(A)):
            ans += A[i] * B[i]
        return ans

```

```

if __name__ == '__main__':
    A = [1,1,1]
    B = [2,2,2]
    solution = Solution()
    print("A 与 B 分别为:", A, B)
    print("点积为:", solution.dotProduct(A, B))

```

4. 运行结果

A 与 B 分别为: [1, 1, 1] [2, 2, 2]
点积为: 6

► 例 13 函数运行时间

1. 问题描述

给定一系列描述函数进入和退出的时间，问每个函数的运行时间是多少。

2. 问题示例

输入 $s = ["F1 Enter 10", "F2 Enter 18", "F2 Exit 19", "F1 Exit 20"]$ ，则输出 $["F1|10", "F2|1"]$ ，即 F1 从 10 时刻进入，20 时刻退出，运行时长为 10，F2 从 18 时刻进入，19 时刻退出，运行时长为 1。

输入 $s = ["F1 Enter 10", "F1 Exit 18", "F1 Enter 19", "F1 Exit 20"]$ ，则输出 $["F1|9"]$ ，即 F1 从 10 时刻进入，18 时刻退出；又从 19 时刻进入，20 时刻退出，总运行时长为 9。

3. 代码实现

```

# 参数 s: 输入原始字符串
# 返回值: 字符串, 意为对应名字的函数运行时长
class Solution:
    def getRuntime(self, a):
        map = {}
        for i in a:
            count = 0
            while not i[count] == ' ':
                count = count + 1
            fun = i[0:count]
            if i[count+2] == 'n':
                count = count + 7
            v = int(i[count:len(i)])
            if fun in map.keys():
                map[fun] = v - map[fun]
            else:
                map[fun] = v
        else:
            count = count + 6

```

```

        v = int(i[count:len(i)])
        map[fun] = v - map[fun]
    res = []
    for i in map:
        res.append(i)
    res.sort()
    for i in range(0,len(res)):
        res[i] = res[i] + ' | ' + str(map[res[i]])
    return res
if __name__ == '__main__':
    s = ["F1 Enter 10", "F2 Enter 18", "F2 Exit 19", "F1 Exit 20"]
    solution = Solution()
    print("输入运行时间:", s)
    print("每个输出时间:", solution.getRuntime(s))

```

4. 运行结果

输入运行时间 : ['F1 Enter 10', 'F2 Enter 18', 'F2 Exit 19', 'F1 Exit 20']
 每个输出时间: ['F1|10', 'F2|1']

► 例 14 查询区间

1. 问题描述

给定一个包含若干个区间的 List 数组, 长度是 1000, 如 [500,1500]、[2100,3100]。给定一个 number, 判断 number 是否在这些区间内, 返回 True 或 False。

2. 问题示例

输入是 List = [[100,1100],[1000,2000],[5500,6500]] 和 number = 6000, 输出是 True, 因为 6000 在区间 [5500,6500]。输入是 List = [[100,1100],[2000,3000]] 和 number = 3500, 输出是 False, 因为 3500 不在 List 的任何一个区间中。

3. 代码实现

```

# 参数 List: 区间列表
# 参数 number: 待查数字
# 返回值: 字符串, True 或者 False
class Solution:
    def isInterval(self, intervalList, number):
        high = len(intervalList) - 1
        low = 0
        while high >= low:
            if 0 < (number - intervalList[(high + low)//2][0]) <= 1000:
                return 'True'
            elif 1000 < number - intervalList[(high + low)//2][0]:
                low = (high + low) // 2 + 1

```

```

        elif 0 > number - intervalList[(high + low)//2][0]:
            high = (high + low) // 2 - 1
        return 'False'
if __name__ == '__main__':
    number = 6000
    intervalList = [[100,1100],[1000,2000],[5500,6500]]
    solution = Solution()
    print(" 区间 List:", intervalList)
    print(" 数字:", number)
    print(" 是否在区间中:", solution.isInterval(intervalList, number))

```

4. 运行结果

区间 List: [[100, 1100], [1000, 2000], [5500, 6500]]
 数字: 6000
 是否在区间中: True

► 例 15 飞行棋

1. 问题描述

一维棋盘，起点在棋盘的最左侧，终点在棋盘的最右侧，棋盘上有几个位置和其他位置相连，如果 A 与 B 相连，但连接是单向的，即当棋子落在位置 A 时，可以选择不投骰子，直接移动棋子从 A 到 B，但不能从 B 移动到 A。给定这个棋盘的长度(length)和位置的相连情况(connections)，用六面的骰子(点数 1~6)，问最少需要投几次才能到达终点。

2. 问题示例

输入 length = 10 和 connections = [[2, 10]]，输出为 1，可以 0->2(投骰子), 2->10(直接相连)。输入 length = 15 和 connections = [[2, 8], [6, 9]]，输出为 2，因为可以 0->6(投骰子), 6->9(直接相连), 9->15(投骰子)。

3. 代码实现

```

# 参数 length: 棋盘长度(不包含起始点)
# 参数 connections: 跳点集合
# 返回值: 整数, 代表最小步数
class Solution:
    def modernLudo(self, length, connections):
        ans = [i for i in range(length+1)]
        for i in range(length+1):
            for j in range(1,7):
                if i - j >= 0:
                    ans[i] = min(ans[i], ans[i-j] + 1)
        for j in connections:
            if i == j[1]:

```

```

        ans[i] = min(ans[i], ans[j[0]])
    return ans[length]

# SPFA 解法
class Solution:
    def modernLudo(self, length, connections):
        dist = [1000000000 for i in range(100050)]
        vis = [0 for i in range(100050)]
        Q = [0 for i in range(100050)]
        st = 0
        ed = 0
        dist[1] = 0
        vis[1] = 1
        Q[ed] = 1;
        ed += 1
        while(st < ed) :
            u = Q[st]
            st += 1
            vis[u] = 0
            for roads in connections :
                if(roads[0] != u):
                    continue
                v = roads[1]
                if(dist[v] > dist[u]):
                    dist[v] = dist[u]
                    if(vis[v] == 0) :
                        vis[v] = 1
                        Q[ed] = v
                        ed += 1
            for i in range(1, 7):
                if (i + u > length):
                    break
                v = i + u
                if(dist[v] > dist[u] + 1) :
                    dist[v] = dist[u] + 1
                    if(vis[v] == 0):
                        vis[v] = 1
                        Q[ed] = v
                        ed += 1
        return dist[length]
if __name__ == '__main__':
    length = 15
    connections = [[2, 8],[6, 9]]
    solution = Solution()
    print(" 棋盘长度:", length)
    print(" 连接:", connections)
    print(" 最小需要:", solution.modernLudo(length, connections))

```

4. 运行结果

```
棋盘长度: 15
连接: [[2, 8], [6, 9]]
最小需要: 2
```

► 例 16 移动石子

1. 问题描述

在 x 轴上分布着 n 个石子, 用 arr 数组表示它们的位置。把这些石子移动到 $1, 3, 5, 7, 2n-1$ 或者 $2, 4, 6, 8, 2n$ 。也就是说, 这些石子移动到从 1 开始连续的奇数位, 或从 2 开始连续的偶数位上。返回最少的移动次数。每次只可以移动 1 个石子, 只能把石子往左移动 1 个单位或往右移动 1 个单位。同一个位置不能同时有 2 个石子。

2. 问题示例

$[5, 4, 1]$, 只需要把 4 移动 1 步到 3, 所以输出是 1。 $arr = [1, 6, 7, 8, 9]$, 最优的移动方案为把 1 移动到 2, 把 6 移动到 4, 把 7 移动到 6, 把 9 移动到 10, 所以输出是 5。

3. 代码实现

```
# 参数 arr: 一个列表
# 返回值: 整数, 为最小移动次数
class Solution:
    def movingStones(self, arr):
        arr = sorted(arr)
        even = 0
        odd = 0
        for i in range(0, len(arr)):
            odd += abs(arr[i] - (2 * i + 1))
            even += abs(arr[i] - (2 * i + 2))
        if odd < even:
            return odd
        return even
if __name__ == '__main__':
    arr = [1, 6, 7, 8, 9]
    solution = Solution()
    print(" 数组:", arr)
    print(" 最小移动数:", solution.movingStones(arr))
```

4. 运行结果

```
数组: [1, 6, 7, 8, 9]
最小移动数: 5
```

► 例 17 数组剔除元素后的乘积

1. 问题描述

给定一个整数数组 A。定义 $B[i] = A[0] * \dots * A[i-1] * A[i+1] * \dots * A[n-1]$ ，即 $B[i]$ 为剔除 $A[i]$ 元素之后所有数组元素之积，计算数组 B 的时候请不要使用除法，输出数组 B。

2. 问题示例

输入 $A = [1, 2, 3]$ ，输出 $[6, 3, 2]$ ，即 $B[0] = A[1] * A[2] = 6$ ； $B[1] = A[0] * A[2] = 3$ ； $B[2] = A[0] * A[1] = 2$ 。输入 $A = [2, 4, 6]$ ，输出 $[24, 12, 8]$ 。

3. 代码实现

```
class Solution:  
    # 参数 A: 整数数组 A  
    # 返回值: 整数数组 B  
    def productExcludeItself(self, A):  
        length, B = len(A), []  
        f = [0 for i in range(length + 1)]  
        f[length] = 1  
        for i in range(length - 1, 0, -1):  
            f[i] = f[i + 1] * A[i]  
        tmp = 1  
        for i in range(length):  
            B.append(tmp * f[i + 1])  
            tmp *= A[i]  
        return B  
  
    # 主函数  
if __name__ == '__main__':  
    solution = Solution()  
    A = [1, 2, 3, 4]  
    B = solution.productExcludeItself(A)  
    print("输入:", A)  
    print("输出:", B)
```

4. 运行结果

```
输入: [1, 2, 3, 4]  
输出: [24, 12, 8, 6]
```

► 例 18 键盘的一行

1. 问题描述

给定一个单词列表，返回可以在键盘（如图 1-1 所示）的一行上使用字母键输入的单词。可以多次使用键盘中的一个字符，输入字符串仅包含字母表的字母。

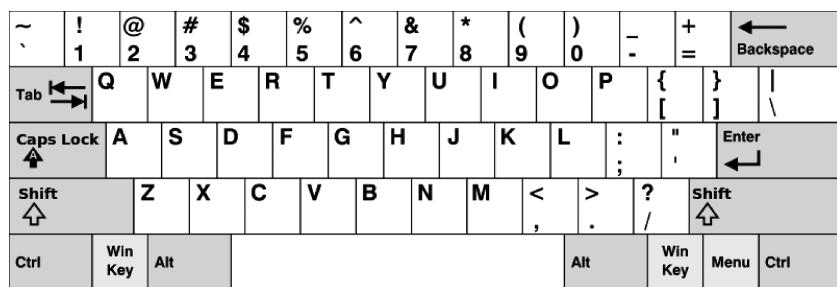


图 1-1 键盘示意图

2. 问题示例

输入["Hello", "Alaska", "Dad", "Peace"],输出["Alaska", "Dad"],即这两个单词可以在键盘的第 3 行输出。

3. 代码实现

```
class Solution:
    # 参数 words: 字符串列表
    # 返回字符串列表
    def findWords(self, words):
        res = []
        s = ["qwertyuiop", "asdfghjkl", "zxcvbnm"]
        for w in words:
            for j in range(3):
                flag = 1
                for i in w:
                    if i.lower() not in s[j]:
                        flag = 0
                        break
                if flag == 1:
                    res.append(w)
                    break
        return res

    # 主函数
if __name__ == '__main__':
    word = ["Hello", "Alaska", "Dad", "Peace"]
    s = Solution()
    print("输入:", word)
    print("输出:", s.findWords(word))
```

4. 运行结果

```
输入: ['Hello', 'Alaska', 'Dad', 'Peace']
输出: ['Alaska', 'Dad']
```

► 例 19 第 n 个数位

1. 问题描述

找出无限正整数数列 $1, 2, \dots$ 中的第 n 个数位。

2. 问题示例

输入 11，输出 0，表示数字序列 $1, 2, \dots$ 中的第 11 位是 0。

3. 代码实现

```
class Solution:
    """
    参数 n: 整数
    返回整数
    """

    def findNthDigit(self, n):
        # 初始化一位数的个数为 9, 从 1 开始
        length = 1
        count = 9
        start = 1
        while n > length * count:
            # 以此类推, 二位数的个数为 90, 从 10 开始
            n -= length * count
            length += 1
            count *= 10
            start *= 10
        # 找到第 n 位数所在的整数 start
        start += (n - 1) // length
        return int(str(start)[(n - 1) % length])

    # 主函数
    if __name__ == '__main__':
        s = Solution()
        n = 11
        print("输入:", n)
        print("输出:", s.findNthDigit(n))
```

4. 运行结果

输入：11

输出：0

► 例 20 找不同

1. 问题描述

给定两个只包含小写字母的字符串 s 和 t 。字符串 t 由随机打乱字符顺序的字符串 s

在随机位置添加一个字符生成。找出在 t 中添加的字符。

2. 问题示例

例如,输入 $s = "abcd"$, $t = "abcde"$,输出 e , e 是加入的字符。

3. 代码实现

```
class Solution:
    """
    参数 s: 字符串
    参数 t: 字符串
    返回字符
    """
    def findTheDifference(self, s, t):
        flag = 0
        for i in range(len(s)):
            # 计算不同字符的 ASCII 码之差
            flag += (ord(t[i]) - ord(s[i]))
        flag += ord(t[-1])
        return chr(flag)

# 主函数
if __name__ == '__main__':
    s = Solution()
    n = "abcd"
    t = "abcde"
    print("输入字符串 1:", n)
    print("输入字符串 2:", t)
    print("输出插入字符:", s.findTheDifference(n, t))
```

4. 运行结果

```
输入字符串 1: abcd
输入字符串 2: abcde
输出插入字符: e
```

► 例 21 第 k 个组合

1. 问题描述

有 n 个人,编号分别为 $1, 2, \dots, n, n$ 为偶数。选择其中的一半人,有 $C(n, n/2)$ 种组合方式,每一种组合方式按照编号从小到大排序,再将已排序的组合方式按照字典序排序,求第 k 种组合方式。

字典序的定义:首先比较两个字符串的长度,长度小的字典序更小,如果长度相同,则从字符串左边开始逐位比较,找到第一位不同的字符,对应字符小的字符串,字典序更小。

2. 问题示例

给出 $n = 2, k = 1$,返回[1],所有组合方式按照字典序排序:[1],[2]。给出 $n = 4$,