Python プログラミングのオンライン問題集

ここでは、オンラインの評価システムを使って問題を解いてみましょう。英語が得意なら、`Codeforces` や `LeetCode` を使うことができます。中国語なら、計蒜客や力扣を利用できます。ここでは `LeetCode` を使います。私は 10 問の問題を解きました。そして、最後の 1 問には複数の方法を試し、プログラムの効率を提出の 10% を上回るものから 99% を上回るものに最適化しました。

Figure 1: cf

1480. 1 次元配列の累積和

> 配列 `nums` が与えられます。配列のランニングサムを `runningSum[i] = sum(nums[0]…nums[i])` と定義します。
>
> `nums` のランニングサムを返してください。

```
class Solution:
    def runningSum(self, nums: [int]) -> [int]:
      running = []
      s = 0
      for num in nums:
```

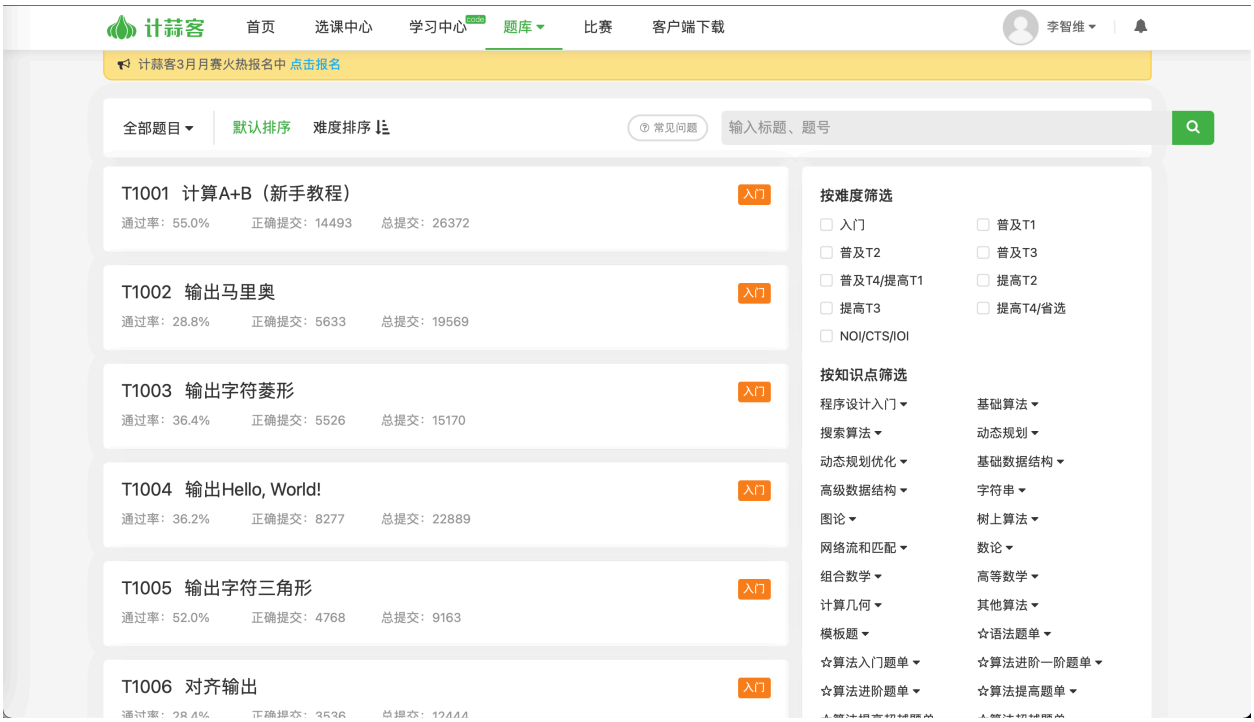

Figure 2: jsk

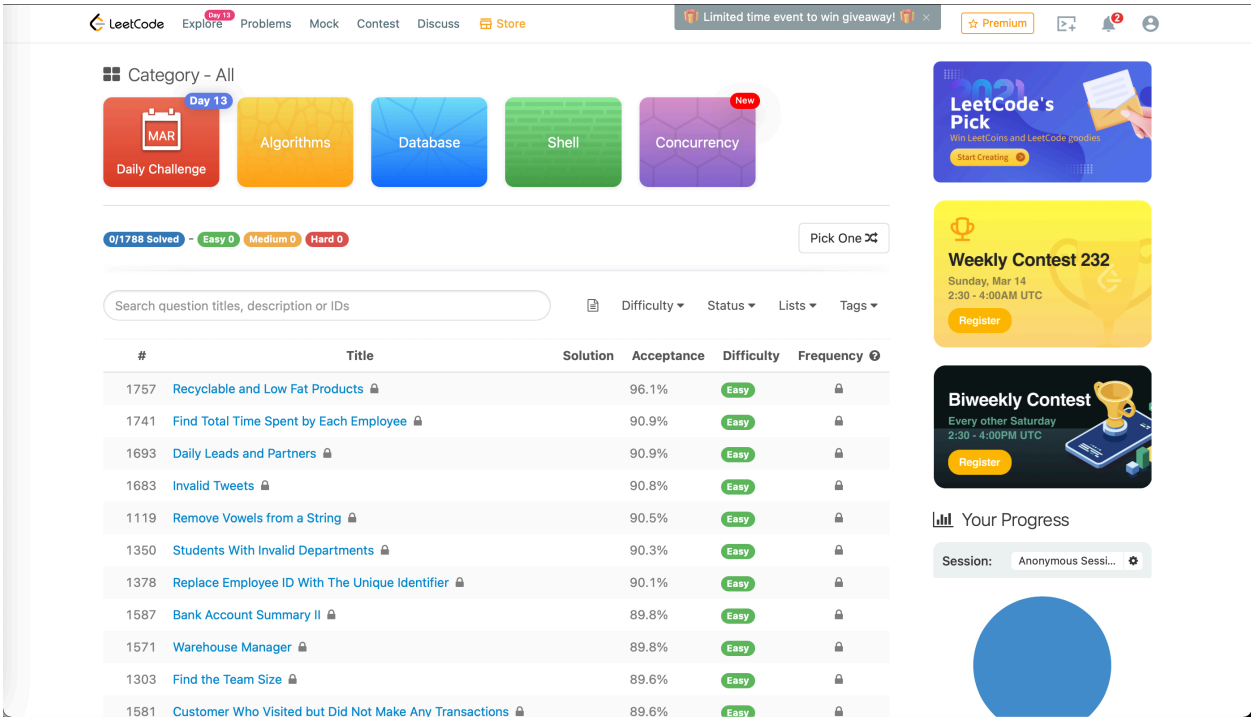

Figure 3: leetcode

```
            s += num
            running.append(s)

        return running
```

このコードは、与えられた整数リスト `nums` に対して、累積和を計算するメソッド `runningSum` を定義しています。具体的には、リスト `nums` の各要素を順番に加算していき、その結果を新しいリスト `running` に追加しています。最終的に、累積和のリスト `running` を返します。

```
print(Solution().runningSum([1,2,3,4]))
```

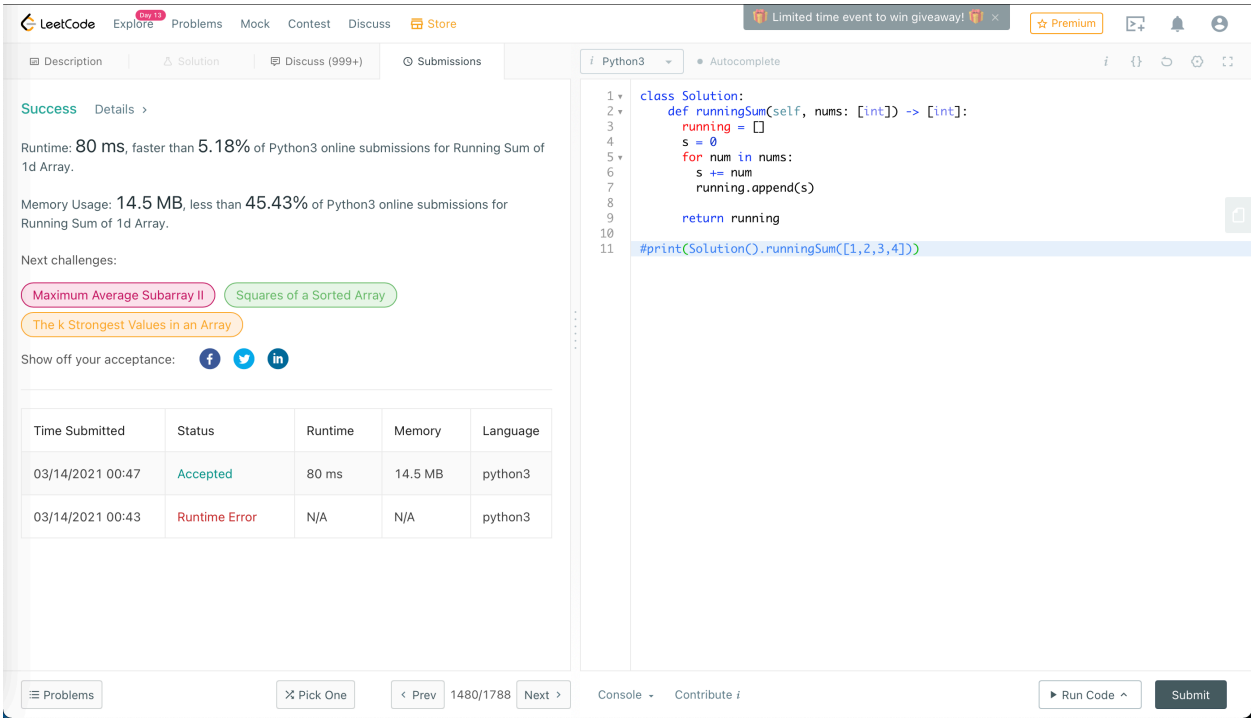

Figure 4: ac

第一問、合格です。

1108. IP アドレスの難読化

> 有効な（IPv4）IP `address` が与えられたとき、その IP アドレスの「無害化」バージョンを返してください。
>
> 無害化された IP アドレス では、すべてのピリオド `"."` が `"[.]"` に置き換えられます。

```python
class Solution:
    def defangIPaddr(self, address: str) -> str:
        return address.replace('.', '[.]')
```

このコードは、与えられたIPアドレスの文字列中のピリオド（.）を`[.]`に置き換えることで、IPアドレスを「デファング」するものです。例えば、`"192.168.1.1"`というIPアドレスは`"192[.]168[.]1[.]1"`に変換されます。これは、IPアドレスを安全に表示するための一般的な方法の一つです。

print(Solution().defangIPaddr('1.1.1.1'))

1431. キャンディーの数が最も多い子供たち

> 配列 `candies` と整数 `extraCandies` が与えられます。ここで、`candies[i]` は ***i番目*** の子供が持ってい
>
> 各子供について、`extraCandies` を子供たちの間で分配することで、その子供が **最大** のキャンディーを持つことが

```python
class Solution:
    def kidsWithCandies(self, candies: [int], extraCandies: int) -> [bool]:
        max = 0
        for candy in candies:
          if candy > max:
            max = candy
        greatests = []
        for candy in candies:
          if candy + extraCandies >= max:
            greatests.append(True)
          else:
            greatests.append(False)
        return greatests
```

このコードは、各子供が持っているキャンディの数に追加のキャンディを加えたときに、その子供が最も多くのキャンディを持っているかどうかを判断するものです。以下にその動作を説明します。

1. max 変数を初期化し、キャンディの最大値を保持します。
2. candies リストをループして、最大のキャンディの数を max に設定します。
3. greatests リストを初期化し、各子供が追加のキャンディを持ったときに最大値以上になるかどうかを判断します。
4. 各子供のキャンディの数に extraCandies を加え、それが max 以上であれば True を、そうでなければ False を greatests リストに追加します。
5. 最終的に greatests リストを返します。

このコードは、各子供が追加のキャンディを持ったときに、最も多くのキャンディを持っているかどうかを示すブール値のリストを返します。

print(Solution().kidsWithCandies([2,3,5,1,3], 3))

1672. 最も裕福な顧客の資産

> `m x n` の整数グリッド `accounts` が与えられます。ここで、`accounts[i][j]` は `i` 番目の顧客が `j` 番目の銀
>
> 顧客の**富**は、その顧客がすべての銀行口座に持っている金額の合計です。最も裕福な顧客とは、**富**が最大の顧客のこ

例

入力: accounts = [[1,2,3],[3,2,1]]
出力: 6
説明:
1番目の顧客の富は 1 + 2 + 3 = 6
2番目の顧客の富は 3 + 2 + 1 = 6
両方の顧客が同じ富を持っているため、6が返されます。

入力: accounts = [[1,5],[7,3],[3,5]]
出力: 10
説明:
1番目の顧客の富は 1 + 5 = 6
2番目の顧客の富は 7 + 3 = 10
3番目の顧客の富は 3 + 5 = 8
最も裕福な顧客は2番目の顧客で、富は10です。

制約

- `m == accounts.length`
- `n == accounts[i].length`
- `1 <= m, n <= 50`
- `1 <= accounts[i][j] <= 100`

解答

```python
def maximumWealth(accounts):
    max_wealth = 0
    for customer in accounts:
        current_wealth = sum(customer)
        if current_wealth > max_wealth:
            max_wealth = current_wealth
    return max_wealth
```

解説

この問題では、各顧客の銀行口座の金額を合計し、最も裕福な顧客の富を見つける必要があります。以下の手順で解決できます。

1. **初期化**: 最大の富を保持する変数 `max_wealth` を 0 で初期化します。
2. **各顧客の富を計算**: 各顧客の銀行口座の金額を合計し、`current_wealth` に代入します。
3. **最大の富を更新**: `current_wealth` が `max_wealth` より大きい場合、`max_wealth` を更新します。
4. **結果を返す**: すべての顧客の富を計算し終えたら、`max_wealth` を返します。

このアプローチでは、各顧客の富を一度だけ計算し、最大値を保持するため、効率的に問題を解決できます。

```python
class Solution:
    def maximumWealth(self, accounts: [[int]]) -> int:
        max = 0
        for account in accounts:
          s = sum(account)
          if max < s:
```

```
            max = s
        return max
```

このコードは、各顧客の口座残高がリストとして与えられた場合に、最も資産が多い顧客の総資産を返すメソッド maximumWealth を定義しています。具体的には、accounts という 2 次元リストが引数として渡され、各顧客の口座残高がリスト内のリストとして格納されています。このメソッドは、各顧客の口座残高の合計を計算し、その中で最大の値を返します。

```
#print(Solution().maximumWealth([[1,2,3],[3,2,1]]))
```

このコードは、Solution クラスの maximumWealth メソッドを呼び出し、2 次元リスト [[1,2,3],[3,2,1]] を引数として渡しています。このリストは、各顧客の資産を表しており、各サブリストが 1 人の顧客の資産を表しています。maximumWealth メソッドは、このリストを受け取り、最も資産が多い顧客の総資産を返すことが期待されています。

このコードはコメントアウトされているため、実際には実行されません。コメントを外すと、maximumWealth メソッドが呼び出され、結果が出力されます。

1470. 配列をシャッフルする

> 配列 nums が 2n 個の要素からなり、形式 [x1,x2,...,xn,y1,y2,...,yn] で与えられるとします。
>
> 配列を [x1,y1,x2,y2,...,xn,yn] の形式で返してください。

```
class Solution:
  def shuffle(self, nums: [int], n: int) -> [int]:
    ns1 = nums[:n]  # 前半部分の要素を取得
    ns2 = nums[n:]  # 後半部分の要素を取得
    ns = []  # シャッフル後のリストを格納するための空リスト
    for i in range(n):
      ns.append(ns1[i])  # 前半部分の要素を追加
      ns.append(ns2[i])  # 後半部分の要素を追加
    return ns  # シャッフルされたリストを返す
```

print(Solution().shuffle([2,5,1,3,4,7], 3))

1512. 良いペアの数

> 整数の配列 `nums` が与えられます。
>
> ペア `(i,j)` は、`nums[i]` == `nums[j]` かつ `i` < `j` のときに *good* と呼ばれます。
>
> *good* なペアの数を返してください。

```python
class Solution:
    def numIdenticalPairs(self, nums: [int]) -> int:
        j = 1
        n = len(nums)
        p = 0
        while j < n:
          for i in range(j):
            if nums[i] == nums[j]:
              p += 1
          j += 1
        return p
```

このコードは、与えられた整数リスト `nums` の中に、同じ値のペアがいくつ存在するかを数えるためのものです。具体的には、`nums` の中の異なるインデックス `i` と `j` において、`nums[i] == nums[j]` となるペアの数を返します。

- `j` はリストのインデックスを表し、1 から始まります。
- `n` はリストの長さです。
- `p` はペアの数をカウントするための変数です。
- `while` ループで `j` を 1 つずつ増やしながら、`for` ループで `j` より前のインデックス `i` をチェックし、`nums[i] == nums[j]` となるペアを見つけたら `p` を増やします。

最終的に、`p` の値が返されます。

print(Solution().numIdenticalPairs([1,2,3,1,1,3]))

771. 宝石と石

> あなたには、宝石の種類を表す文字列 `jewels` と、持っている石を表す文字列 `stones` が与えられます。`stones` の
>
> 文字は大文字と小文字を区別するので、`"a"` と `"A"` は異なる種類の石と見なされます。

```python
class Solution:
    def numJewelsInStones(self, jewels: str, stones: str) -> int:
        n = 0
        for i in range(len(jewels)):
          js = jewels[i:i+1]
          n += stones.count(js)
        return n
```

このコードは、jewels に含まれる各文字が stones に何回出現するかを数えるためのものです。
以下にその動作を説明します。

1. n という変数を 0 で初期化します。これは、jewels に含まれる文字が stones に出現する回数をカウントするための変数です。
2. for ループを使用して、jewels の各文字を 1 つずつ取り出します。jewels[i:i+1] は、jewels の i 番目の文字を取得するためのスライスです。
3. stones.count(js) を使用して、stones の中に js が何回出現するかを数え、その結果を n に加算します。
4. ループが終了したら、n を返します。これが jewels に含まれる文字が stones に出現する総回数です。

このコードは、jewels の各文字が stones に何回出現するかを効率的に数えることができます。

print(Solution().numJewelsInStones(“aA”, “aAAbbbb”))

1603. 駐車システムの設計

> 駐車場のための駐車システムを設計してください。駐車場には、大型、中型、小型の3種類の駐車スペースがあり、それぞれの
>
> `ParkingSystem` クラスを実装してください：
>
> - `ParkingSystem(int big, int medium, int small)`：`ParkingSystem` クラスのオブジェクトを初期化します。各

> - `bool addCar(int carType)`：駐車場に入ろうとしている車に対して、`carType` の駐車スペースがあるかどうかを確

```python
class ParkingSystem:
    slots = [0, 0, 0]

    def __init__(self, big: int, medium: int, small: int):
      self.slots[0] = big
      self.slots[1] = medium
      self.slots[2] = small

    def addCar(self, carType: int) -> bool:
      if self.slots[carType - 1] > 0:
        self.slots[carType - 1] -=1
        return True
      else:
        return False

# parkingSystem = ParkingSystem(1, 1, 0)
# print(parkingSystem.addCar(1))
# print(parkingSystem.addCar(2))
# print(parkingSystem.addCar(3))
# print(parkingSystem.addCar(1))
```

上記のコードは、駐車システムを表す `ParkingSystem` クラスのインスタンスを作成し、異なるタイプの車を駐車場に追加する例です。各 `addCar` メソッドの呼び出しは、指定されたタイプの車を駐車場に追加しようとし、成功したかどうかを示すブール値を返します。

1773. ルールに一致するアイテムを数える

> あなたは配列 `items` を与えられており、各 `items[i] = [typei, colori, namei]` は `i` 番目のアイテムの種類、色、名前を表しています。また、2 つの文字列 `ruleKey` と `ruleValue` で表されるルールも与えられます。
>
> `i` 番目のアイテムがルールに一致するのは、以下の**いずれか**が真である場合です：
>
> - `ruleKey == "type"` かつ `ruleValue == typei`。
> - `ruleKey == "color"` かつ `ruleValue == colori`。

- `ruleKey == "name"` かつ `ruleValue == namei`。

与えられたルールに一致するアイテムの数を返してください。

```
class Solution:
    def countMatches(self, items: [[str]], ruleKey: str, ruleValue: str) -> int:
      i = 0
      if ruleKey == "type":
        i = 0
      elif ruleKey == "color":
        i = 1
      else:
        i = 2
      n = 0
      for item in items:
        if item[i] == ruleValue:
          n +=1
      return n
```

このコードは、与えられた `items` リストの中から、特定の `ruleKey` に基づいて `ruleValue` と一致するアイテムの数を数えるものです。`ruleKey` が`"type"`、`"color"`、または`"name"` のいずれかに応じて、`items` 内の各アイテムの対応するインデックスをチェックし、`ruleValue` と一致するかどうかを確認します。一致するアイテムの数 `n` を返します。

print(Solution().countMatches([[“phone”,“blue”,“pixel”],[“computer”,“sil
“color”, “silver”))

1365. 現在の数値よりも小さい数値はいくつあるか

> 配列 `nums` が与えられたとき、各 `nums[i]` に対して、配列内にあるそれより小さい数の個数を見つけ出してください。
>
> 答えを配列として返してください。

> ```
> 入力: nums = [8,1,2,2,3]
> 出力: [4,0,1,1,3]

> 説明:
> nums[0]=8 に対して、それより小さい数が4つ存在します（1，2，2，3）。
> nums[1]=1 に対して、それより小さい数は存在しません。
> nums[2]=2 に対して、それより小さい数が1つ存在します（1）。
> nums[3]=2 に対して、それより小さい数が1つ存在します（1）。
> nums[4]=3 に対して、それより小さい数が3つ存在します（1，2，2）。
> ```

```python
class Solution:
    def smallerNumbersThanCurrent(self, nums: [int]) -> [int]:
        ns = []
        l = len(nums)
        for i in range(l):
          n = 0
          for j in range(l):
            if i != j:
              if nums[j] < nums[i]:
                n += 1
          ns.append(n)
        return ns
```

このコードは、与えられた整数リスト nums に対して、各要素よりも小さい要素の数を計算し、その結果をリストとして返すものです。具体的には、各要素に対して、リスト内の他のすべての要素と比較し、自分よりも小さい要素の数を数えています。その結果を新しいリスト ns に追加し、最終的にそのリストを返します。

print(Solution().smallerNumbersThanCurrent([8,1,2,2,3]))

用时528ms，击败了11.81%的程序。これを最適化してみましょう。

```python
class Solution:
    def smallerNumbersThanCurrent(self, nums: [int]) -> [int]:
        l = len(nums)
```

```
sort_nums = nums.copy()

        ins = list(range(l))
        for i in range(l):
          for j in range(i+1, l):
            if sort_nums[i] > sort_nums[j]:
              a = sort_nums[i]
              sort_nums[i] = sort_nums[j]
              sort_nums[j] = a

              a = ins[i]
              ins[i] = ins[j]
              ins[j] = a

        smalls = [0]
        for i in range(1, l):
          if sort_nums[i-1] == sort_nums[i]:
            smalls.append(smalls[i-1])
          else:
            smalls.append(i)
```

このコードは、リスト `sort_nums` をソートし、その過程で元のインデックスを保持するためのリスト `ins` を更新します。その後、ソートされたリスト内の各要素が何番目に小さいかを示すリスト `smalls` を作成します。具体的には、`smalls` の各要素は、その位置の要素がソートされたリスト内で何番目に小さいかを示します。同じ値が連続する場合、前の要素と同じ値を保持します。

```
    # print(sort_nums)
    # print(smalls)

r_is = list(range(l))
for i in ins:
    r_is[ins[i]] = i

ns = []
for i in range(l):
    ns.append(smalls[r_is[i]])
return ns
```

print(Solution().smallerNumbersThanCurrent([8,1,2,2,3]))

このテストの実行時間は`284ms`で、先ほどの`528ms`よりも短くなりました。

システムの関数を使って簡略化します。

```python
class Solution:
    def smallerNumbersThanCurrent(self, nums: [int]) -> [int]:
        # numsのコピーを作成し、ソートする
        sort_nums = nums.copy()
        sort_nums.sort()

        # 各要素について、ソートされたリストでのインデックスを取得
        ns = []
        for num in nums:
          ns.append(sort_nums.index(num))
        return ns
```

print(Solution().smallerNumbersThanCurrent([8,1,2,2,3]))

この処理にはわずか`64ms`しかかからず、提出されたコードの`71%`を上回る性能を発揮しました。

```python
class Solution:
    def smallerNumbersThanCurrent(self, nums: [int]) -> [int]:
        l = len(nums)
        ns = [0] * l
        for i in range(l):
          for j in range(i+1, l):
            if nums[i] > nums[j]:
              ns[i] +=1
            elif nums[i] < nums[j]:
              ns[j] +=1
            else:
```

```
            pass
        return ns
```

このコードは、与えられた整数リスト `nums` に対して、各要素よりも小さい要素の数を計算するものです。具体的には、リスト `ns` に各要素よりも小さい要素の数を格納し、最終的にそのリストを返します。

- `l` は `nums` の長さを表します。
- `ns` は、各要素よりも小さい要素の数を格納するためのリストで、初期値はすべて 0 です。
- 二重の `for` ループを使用して、各要素 `nums[i]` とその後の要素 `nums[j]` を比較します。
  - `nums[i]` が `nums[j]` より大きい場合、`ns[i]` を 1 増やします。
  - `nums[i]` が `nums[j]` より小さい場合、`ns[j]` を 1 増やします。
  - 等しい場合は何もしません。

最終的に、`ns` リストが返されます。このリストには、各要素よりも小さい要素の数が格納されています。

print(Solution().smallerNumbersThanCurrent([8,1,2,2,3]))

また別の解法を思いつきました。実行時間は`400ms`です。

```python
class Solution:
    def smallerNumbersThanCurrent(self, nums: [int]) -> [int]:
        ss = sorted((e,i) for i,e in enumerate(nums))

l = len(nums)
smalls = [0]
for i in range(1, l):
    (e0, j0) = ss[i-1]
    (e1, j1) = ss[i]
    if e0 == e1:
        smalls.append(smalls[i-1])
    else:
        smalls.append(i)
```

```
ns = [0]*l
for i in range(l):
    (e, j) = ss[i]
    ns[j] = smalls[i]
return ns
```

このコードは、長さ `l` のリスト `ns` を初期化し、`ss` リストの各要素 `(e, j)` を使用して、`smalls` リストの対応する要素を `ns` の適切な位置に配置しています。最終的に、`ns` リストが返されます。

print(Solution().smallerNumbersThanCurrent([8,1,2,2,3]))

> 実行時間: 52 ms、Python3オンライン提出の91.45%より速いです。
>
> メモリ使用量: 14.6 MB、Python3オンライン提出の15.18%より少ないです。

ついに成功しました！この方法はさらに速く、`91.45%`の提出を上回りました。

さらに簡潔にします。

```python
class Solution:
    def smallerNumbersThanCurrent(self, nums: [int]) -> [int]:
        ss = sorted((e, i) for i, e in enumerate(nums))

l = len(nums)
smalls = [0]
ns = [0]*l
for i in range(1, l):
  (e0, j0) = ss[i-1]
  (e1, j1) = ss[i]
  if e0 == e1:
    smalls.append(smalls[i-1])
  else:
    smalls.append(i)

ns[j1] = smalls[i]
return ns
```

print(Solution().smallerNumbersThanCurrent([8,1,2,2,3]))

続けます。

```python
class Solution:
    def smallerNumbersThanCurrent(self, nums: [int]) -> [int]:
        ss = sorted((e, i) for i, e in enumerate(nums))
```

このコードは、与えられた整数リスト `nums` に対して、各要素よりも小さい要素の数を計算するためのクラス `Solution` とそのメソッド `smallerNumbersThanCurrent` を定義しています。具体的には、`nums` の各要素とそのインデックスをタプルとしてソートし、その結果を `ss` に格納しています。

```
l = len(nums)
last = 0
ns = [0] * l
for i in range(1, l):
    (e0, j0) = ss[i - 1]
    (e1, j1) = ss[i]
    if e0 == e1:
        pass
    else:
        last = i

ns[j1] = last
return ns
```

print(Solution().smallerNumbersThanCurrent([8,1,2,2,3]))

この時点で、私たちのプログラムは`40ms`で動作し、`99.81%`のプログラムを上回りました。

> 実行時間: 40 ms、Python3オンライン提出の99.81%より高速です。
>
> メモリ使用量: 14.4 MB、Python3オンライン提出の15.18%より少ないです。

もう一つの解法を紹介します。

```python
class Solution:
    def smallerNumbersThanCurrent(self, nums: [int]) -> [int]:
        l = len(nums)
        n = [0] * 101
        max_num = 0
        for num in nums:
            n[num] += 1
            if num > max_num:
                max_num = num

sm = [0] * (max_num + 1)
sum = 0
for i in range(max_num + 1):
    sm[i] = sum
    sum += n[i]

ns = [0] * l
for i in range(l):
    ns[i] = sm[nums[i]]

  ns を返す
```

print(Solution().smallerNumbersThanCurrent([8,1,2,2,3]))

以下是一个稍微复杂一些的例子：

```markdown
---
layout: post
title: "深入理解JavaScript中的闭包"
date: 2023-10-05
author: "John Doe"
categories: [JavaScript, 编程]
```

tags: [闭包, 高级特性]

深入理解JavaScript中的闭包

在JavaScript中，闭包（Closure）是一个非常重要的概念。它不仅在函数式编程中扮演着关键角色，还在日常开发中有着广泛的

什么是闭包？

闭包是指一个函数能够访问并操作其词法作用域（Lexical Scope）中的变量，即使这个函数在其词法作用域之外执行。换句话说

示例代码

```javascript
function outerFunction() {
    let outerVariable = 'I am outside!';

    function innerFunction() {
        console.log(outerVariable);
    }

    return innerFunction;
}

const closureExample = outerFunction();
closureExample(); // 输出: I am outside!
```

在这个例子中，`innerFunction` 是一个闭包，因为它能够访问 `outerFunction` 中的 `outerVariable`，即使 `outerFunction` 已经执行完毕。

闭包的工作原理

闭包的工作原理可以归结为以下几点：

1. **词法作用域**：JavaScript 使用词法作用域，这意味着函数的作用域在函数定义时就已经确定，而不是在函数调用时。

2. **环境记录**：当一个函数被创建时，它会创建一个环境记录，用于存储其词法作用域中的变量。
3. **引用保留**：即使外部函数已经执行完毕，只要内部函数仍然存在引用，外部函数的变量就不会被垃圾回收。

闭包的实际应用

闭包在 JavaScript 中有许多实际应用场景，以下是一些常见的例子：

1. 数据封装

闭包可以用于创建私有变量，从而实现数据封装。

```
function createCounter() {
    let count = 0;

    return function() {
        count++;
        return count;
    };
}

const counter = createCounter();
console.log(counter()); // 输出: 1
console.log(counter()); // 输出: 2
```

在这个例子中，`count` 变量被封装在 `createCounter` 函数内部，外部无法直接访问或修改它。

2. 回调函数

闭包常用于回调函数中，特别是在异步编程中。

```
function fetchData(url, callback) {
    setTimeout(() => {
        const data = `Data from ${url}`;
        callback(data);
    }, 1000);
```

```
}

fetchData('https://example.com', function(data) {
    console.log(data); // 输出: Data from https://example.com
});
```

在这个例子中，回调函数形成了一个闭包，能够访问 `fetchData` 函数中的 `url` 变量。

3. 函数柯里化

闭包还可以用于实现函数柯里化（Currying），即将一个多参数函数转换为一系列单参数函数。

```
function add(a) {
    return function(b) {
        return a + b;
    };
}

const addFive = add(5);
console.log(addFive(3)); // 输出: 8
```

在这个例子中，`add` 函数返回了一个闭包，该闭包记住了 `a` 的值，并在后续调用中使用了它。

总结

闭包是 JavaScript 中一个强大且灵活的特性，理解它的工作原理和应用场景对于编写高效、可维护的代码至关重要。通过闭包，我们可以实现数据封装、回调函数、函数柯里化等多种功能。希望本文能帮助你更好地理解和应用闭包。

这个例子展示了如何在Jekyll博客中使用Markdown格式编写一篇关于JavaScript闭包的技术文章。文章包含了代码示例、解释以

```python
class Solution:
    def smallerNumbersThanCurrent(self, nums: [int]) -> [int]:
        l = len(nums)
```

```
        n = [0] * 101
        max_num = 0
        for num in nums:
            n[num] += 1
            if num > max_num:
                max_num = num

short_n = []
short_num = [] * l
zn = [0] * 101
j = 0
for i in range(max_num+1):
    if n[i] > 0:
        zn[i] = j
        short_n.append(n[i])
        short_num.append(num)
        j += 1

sm = [0] * j
sum = 0
for i in range(j):
    sm[i] = sum
    sum += short_n[i]

ns = [0] * l
for i in range(l):
    ns[i] = sm[zn[nums[i]]]
return ns
```

print(Solution().smallerNumbersThanCurrent([8,1,2,2,3]))

```python
class Solution:

    def smallerNumbersThanCurrent(self, nums: [int]) -> [int]:
      max_num = max(nums)
```

このコードスニペットは、与えられた整数リスト `nums` の中で、各要素よりも小さい要素の数を計算するためのメソッド `smallerNumbersThanCurrent` の一部です。`max_num` は `nums` リストの中の最大値を取得するために使用されています。この最大値は、後続の処理で各要素よりも小さい要素の数を効率的に計算するために利用される可能性があります。

この部分だけでは完全な処理は見えませんが、`max_num` を使って各要素よりも小さい要素の数を数えるための準備をしていると推測できます。

```
n = [0] * (max_num + 1)
for num in nums:
    n[num] += 1

sorted_ls = []
for i in range(max_num + 1):
    if n[i] > 0:
        sorted_ls.append(i)

sm = [0] * (max_num + 1)
sum = 0
for i in range(len(sorted_ls)):
    v = sorted_ls[i]
    sm[v] = sum
    sum += n[v]

ns = []
for i in range(len(nums)):
    ns.append(sm[nums[i]])
return ns
# print(Solution().smallerNumbersThanCurrent([72,48,32,16,10,59,83,38,1,4,68,7,67,16,5,35,99,15,55,11,2
```

練習

- 学生は上記のようにいくつかの問題を解くことができます。