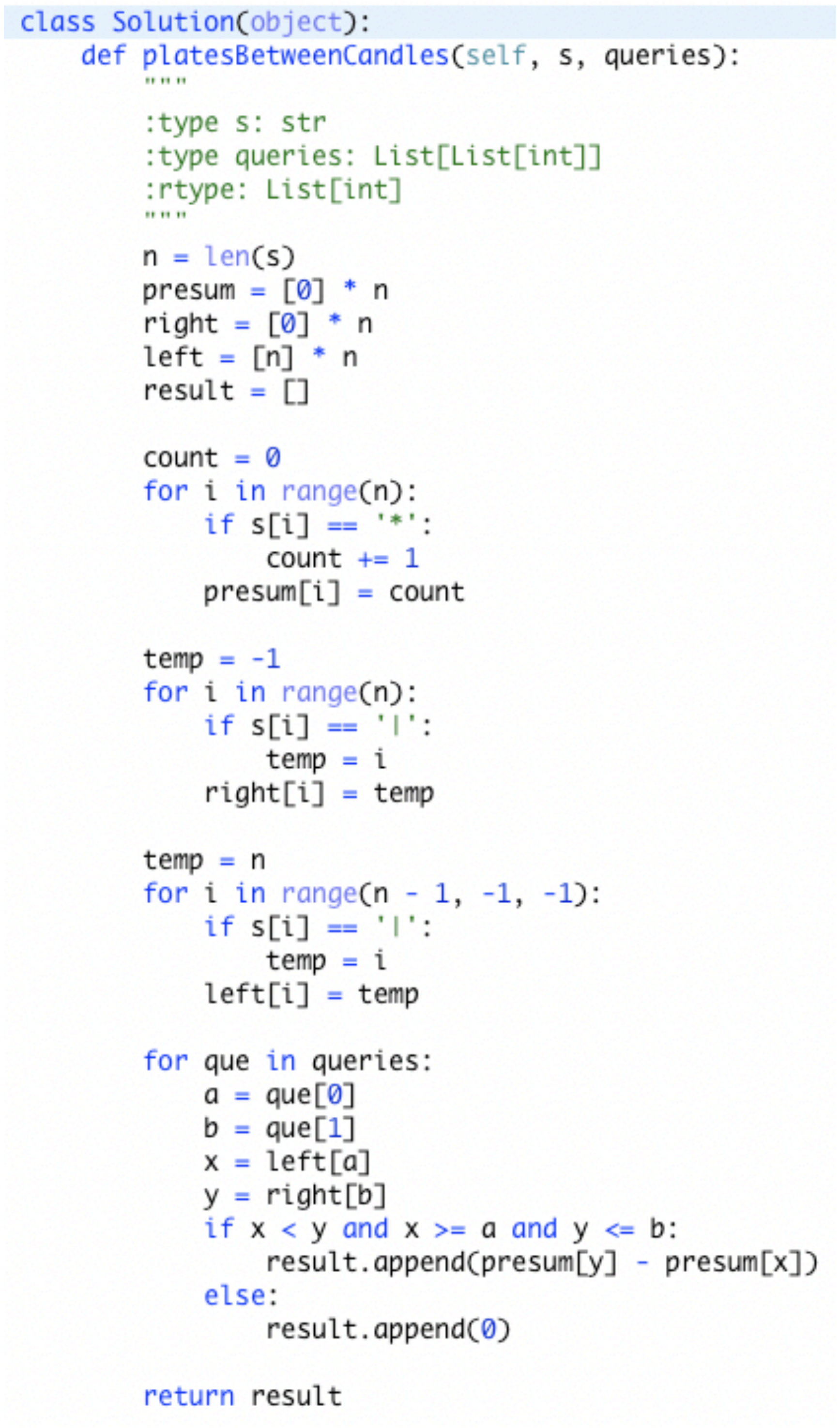
TikTok OA

Star and bars:



```python
class Solution(object):
    def platesBetweenCandles(self, s, queries):
        """
        :type s: str
        :type queries: List[List[int]]
        :rtype: List[int]
        """
        n = len(s)
        presum = [0] * n
        right = [0] * n
        left = [n] * n
        result = []

        count = 0
        for i in range(n):
            if s[i] == '*':
                count += 1
            presum[i] = count

        temp = -1
        for i in range(n):
            if s[i] == '|':
                temp = i
            right[i] = temp

        temp = n
        for i in range(n - 1, -1, -1):
            if s[i] == '|':
                temp = i
            left[i] = temp

        for que in queries:
            a = que[0]
            b = que[1]
            x = left[a]
            y = right[b]
            if x < y and x >= a and y <= b:
                result.append(presum[y] - presum[x])
            else:
                result.append(0)

        return result
```

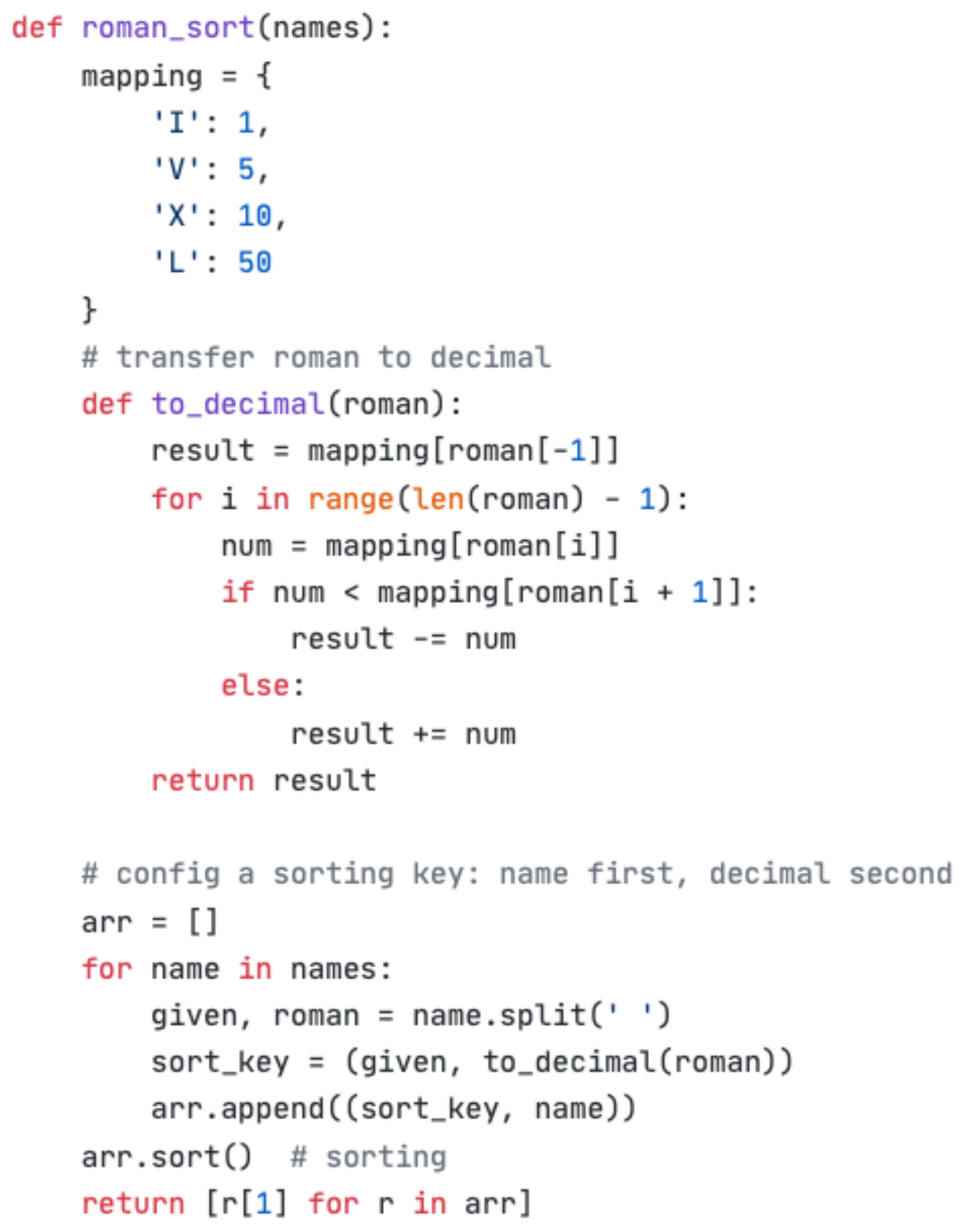
Inversions:

```
def getInvCount(arr):
    result = 0
    n = len(arr)
    for i in range(n - 1):
        small = 0
        for j in range(i + 1, n):
            if arr[i] > arr[j]:
                small += 1

        great = 0
        for j in range(i - 1, -1, -1):
            if arr[i] < arr[j]:
                great += 1

        result += small * great

    return result
```

Domino 3D:

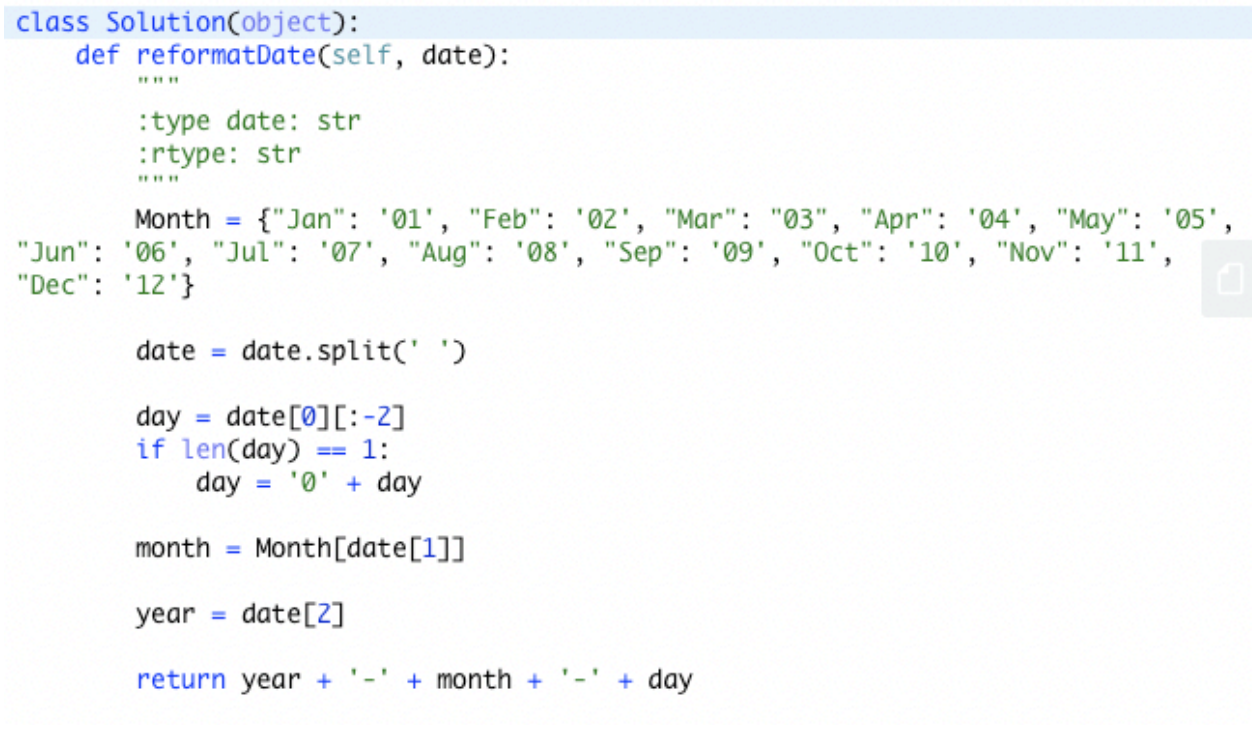
```
def dominos(n):
    def f(n):
        if n == 0:
            return 1
        if n == 1:
            return 2

        return 2*g(n) + 2* g(n-1) + f(n-2)

    def g(n):
        if n == 0:
            return 0
        if n == 1:
            return 1

        return g(n-1) + f(n-1)

    return f(n)
```

Ancestral names:



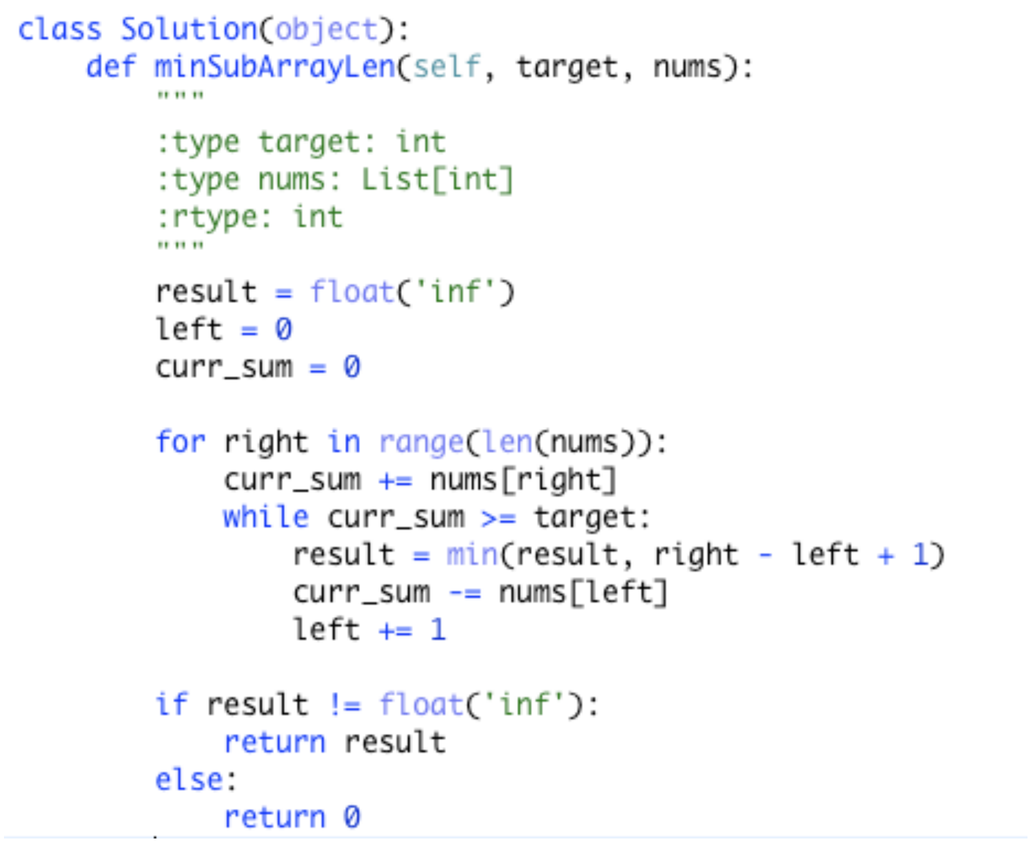
```python
def roman_sort(names):
    mapping = {
        'I': 1,
        'V': 5,
        'X': 10,
        'L': 50
    }
    # transfer roman to decimal
    def to_decimal(roman):
        result = mapping[roman[-1]]
        for i in range(len(roman) - 1):
            num = mapping[roman[i]]
            if num < mapping[roman[i + 1]]:
                result -= num
            else:
                result += num
        return result

    # config a sorting key: name first, decimal second
    arr = []
    for name in names:
        given, roman = name.split(' ')
        sort_key = (given, to_decimal(roman))
        arr.append((sort_key, name))
    arr.sort()  # sorting
    return [r[1] for r in arr]
```

Preprocess date:



```
class Solution(object):
    def reformatDate(self, date):
        """
        :type date: str
        :rtype: str
        """
        Month = {"Jan": '01', "Feb": '02', "Mar": "03", "Apr": '04', "May": '05',
"Jun": '06', "Jul": '07', "Aug": '08', "Sep": '09', "Oct": '10', "Nov": '11',
"Dec": '12'}

        date = date.split(' ')

        day = date[0][:-2]
        if len(day) == 1:
            day = '0' + day

        month = Month[date[1]]

        year = date[2]

        return year + '-' + month + '-' + day
```

Minimum size subarray sum:

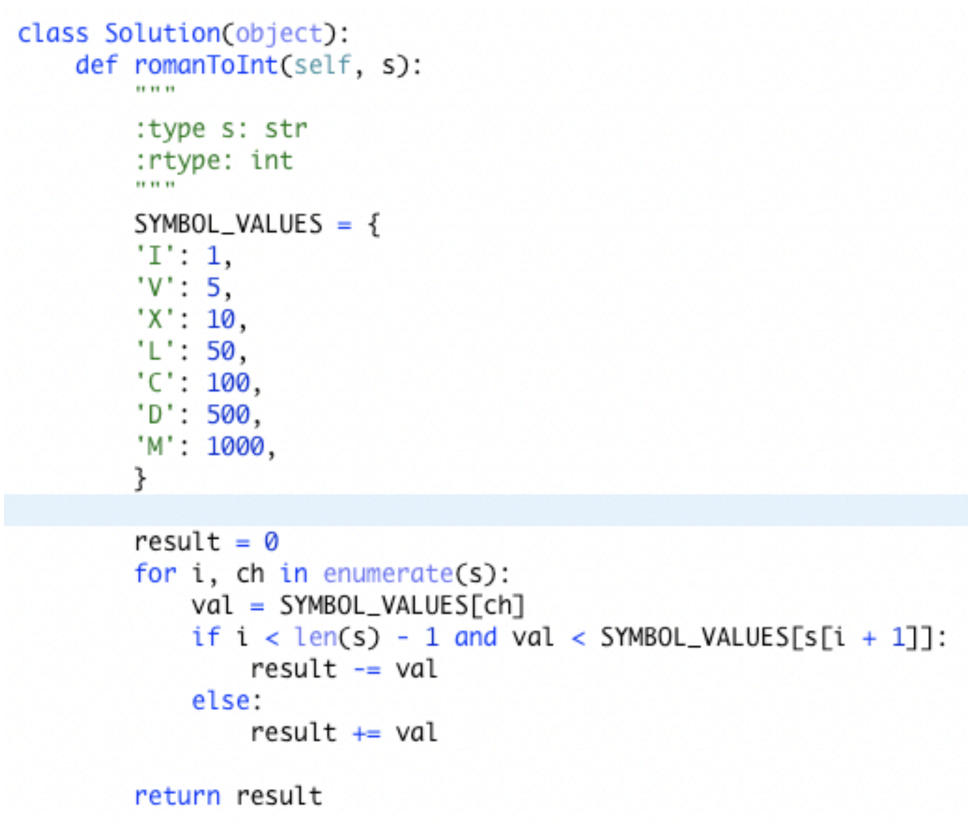


```
class Solution(object):
    def minSubArrayLen(self, target, nums):
        """
        :type target: int
        :type nums: List[int]
        :rtype: int
        """
        result = float('inf')
        left = 0
        curr_sum = 0

        for right in range(len(nums)):
            curr_sum += nums[right]
            while curr_sum >= target:
                result = min(result, right - left + 1)
                curr_sum -= nums[left]
                left += 1

        if result != float('inf'):
            return result
        else:
            return 0
```

Find the Distance Value Between Two Arrays:

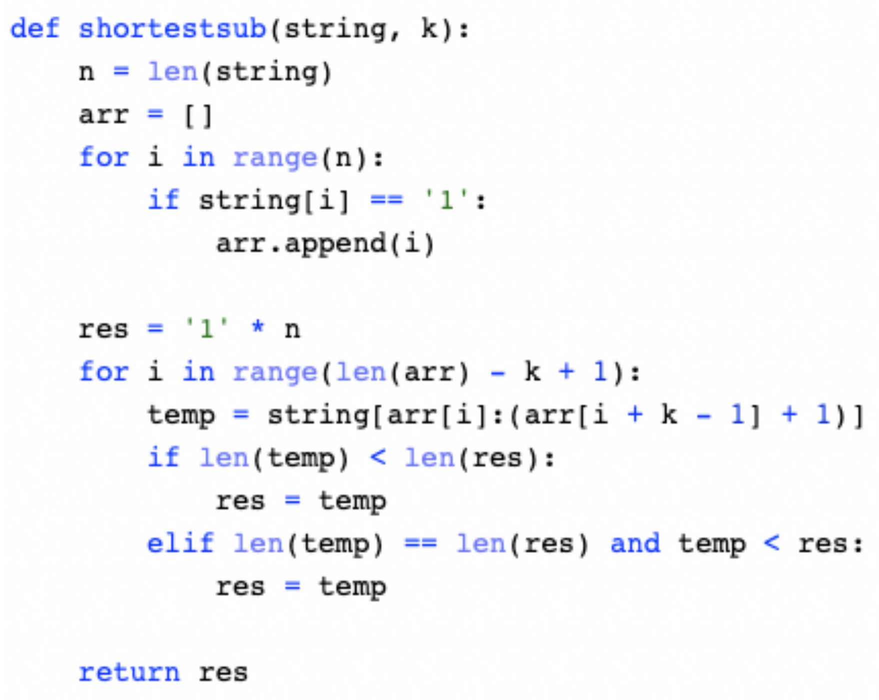
```
class Solution(object):
    def findTheDistanceValue(self, arr1, arr2, d):
        """
        :type arr1: List[int]
        :type arr2: List[int]
        :type d: int
        :rtype: int
        """
        arr2.sort()
        len2 = len(arr2)
        distance = 0
        for n in arr1:
            idx = bisect_left(arr2, n)
            if idx < len2 and arr2[idx] - n > d:
                if idx > 0:
                    if n - arr2[idx - 1] > d:
                        distance += 1
                else:
                    distance += 1
            elif idx == len2 and n - arr2[idx - 1] > d:
                distance += 1

        return distance
```

Roman to integer:



```python
class Solution(object):
    def romanToInt(self, s):
        """
        :type s: str
        :rtype: int
        """
        SYMBOL_VALUES = {
        'I': 1,
        'V': 5,
        'X': 10,
        'L': 50,
        'C': 100,
        'D': 500,
        'M': 1000,
        }

        result = 0
        for i, ch in enumerate(s):
            val = SYMBOL_VALUES[ch]
            if i < len(s) - 1 and val < SYMBOL_VALUES[s[i + 1]]:
                result -= val
            else:
                result += val

        return result
```

Find the shortest and lexicographically correct substring that contain k 1's:



```python
def shortestsub(string, k):
    n = len(string)
    arr = []
    for i in range(n):
        if string[i] == '1':
            arr.append(i)

    res = '1' * n
    for i in range(len(arr) - k + 1):
        temp = string[arr[i]:(arr[i + k - 1] + 1)]
        if len(temp) < len(res):
            res = temp
        elif len(temp) == len(res) and temp < res:
            res = temp

    return res
```