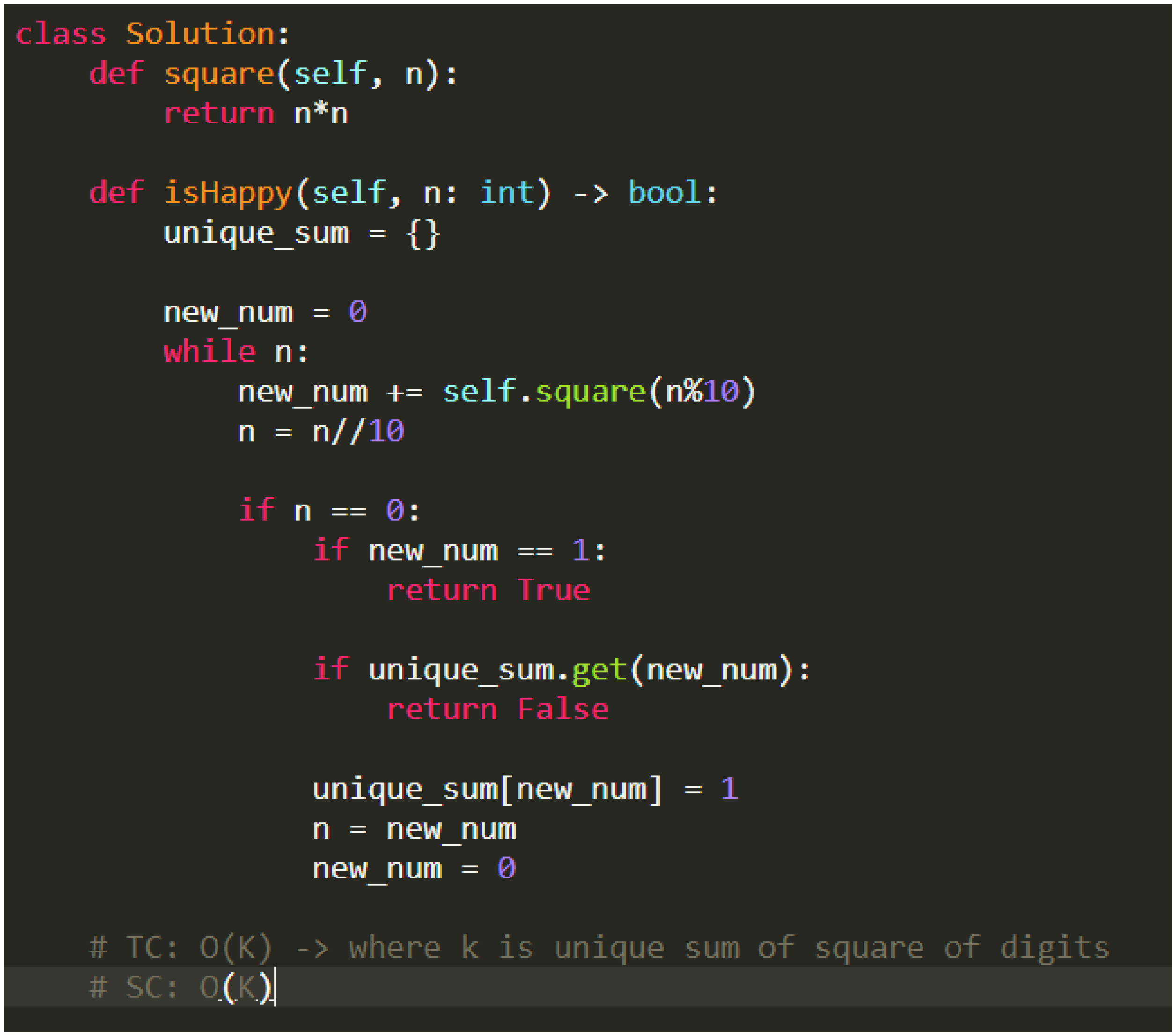
1. https://leetcode.com/problems/happy-number/

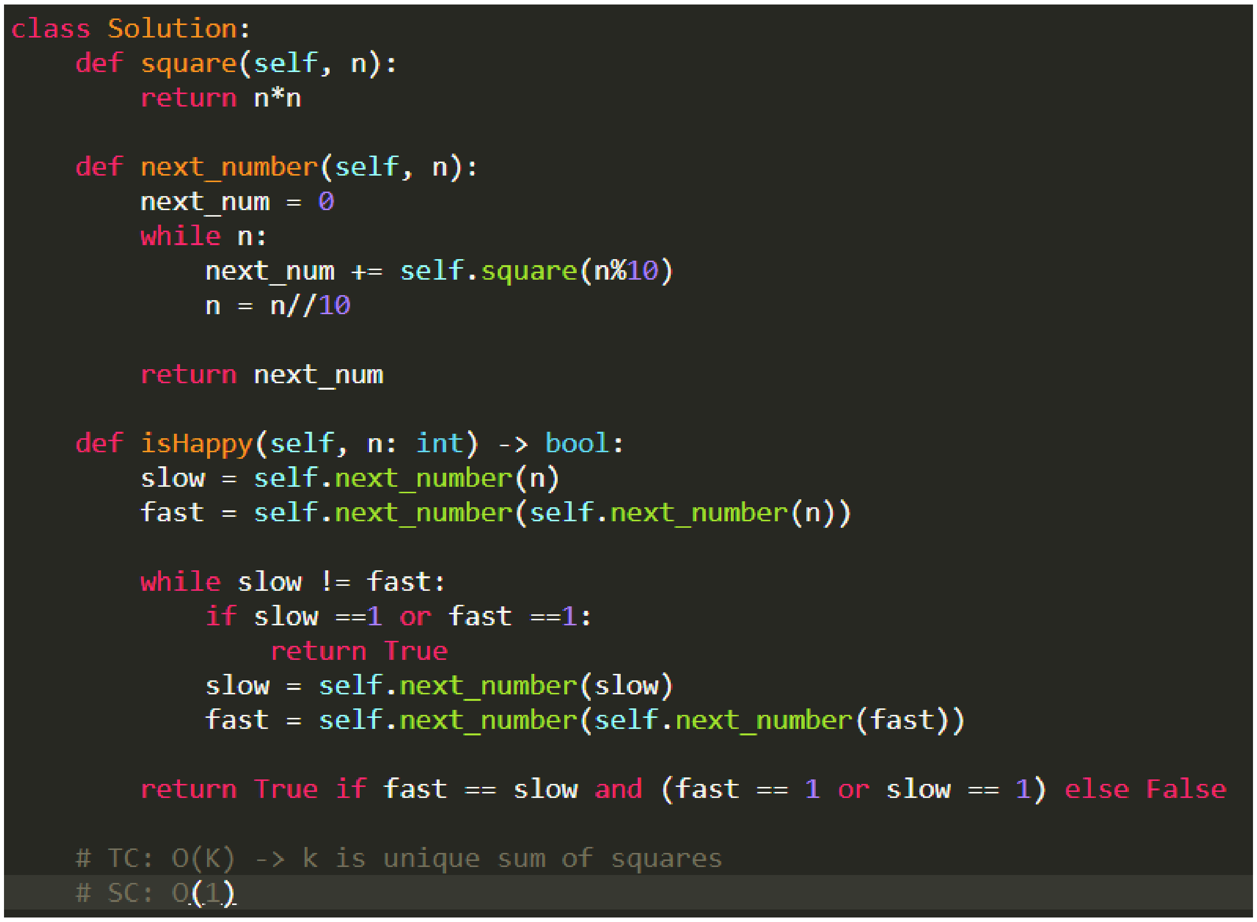
Brute force:
-> run a while loop until the number becomes 1 or there is repetition.



```python
class Solution:
    def square(self, n):
        return n*n

    def isHappy(self, n: int) -> bool:
        unique_sum = {}

        new_num = 0
        while n:
            new_num += self.square(n%10)
            n = n//10

            if n == 0:
                if new_num == 1:
                    return True

                if unique_sum.get(new_num):
                    return False

                unique_sum[new_num] = 1
                n = new_num
                new_num = 0

    # TC: O(K) -> where k is unique sum of square of digits
    # SC: O(K)
```

Better:

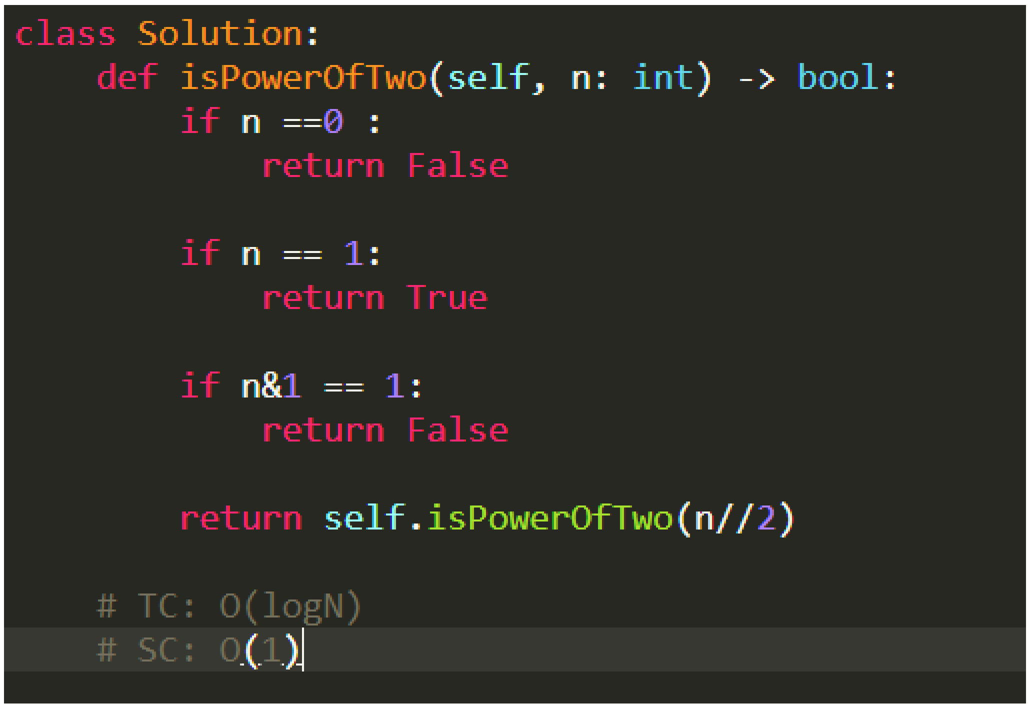
-> use two pointer logic, one slow and other fast. Fast will point to sum of square of next to next number, slow will point to sum of square of next number. At some if they meet and that point is not 1 then false otherwise true.



```python
class Solution:
    def square(self, n):
        return n*n

    def next_number(self, n):
        next_num = 0
        while n:
            next_num += self.square(n%10)
            n = n//10

        return next_num

    def isHappy(self, n: int) -> bool:
        slow = self.next_number(n)
        fast = self.next_number(self.next_number(n))

        while slow != fast:
            if slow ==1 or fast ==1:
                return True
            slow = self.next_number(slow)
            fast = self.next_number(self.next_number(fast))

        return True if fast == slow and (fast == 1 or slow == 1) else False

    # TC: O(K) -> k is unique sum of squares
    # SC: O(1)
```

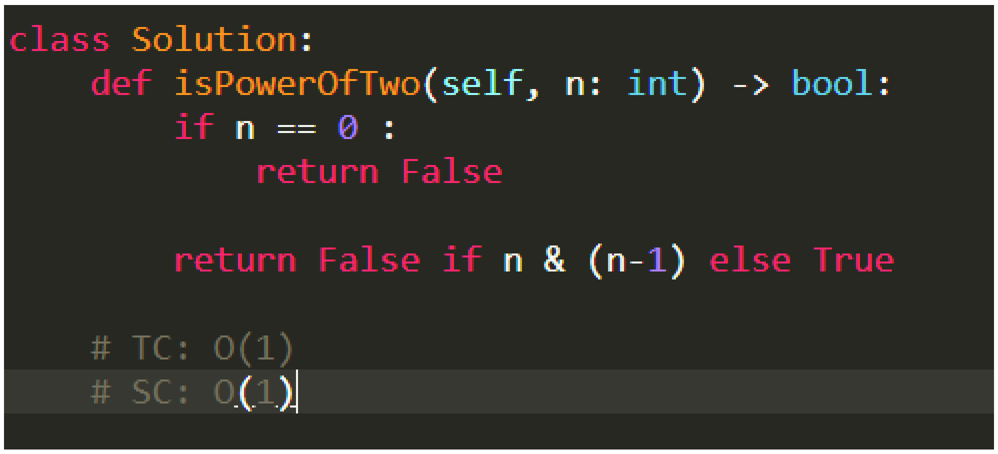
2. https://leetcode.com/problems/power-of-two/

Brute force:
-> check if no. is odd or 0, then false otherwise keep dividing no. by 2 if it gets to 1 the True



```python
class Solution:
    def isPowerOfTwo(self, n: int) -> bool:
        if n ==0 :
            return False

        if n == 1:
            return True

        if n&1 == 1:
            return False

        return self.isPowerOfTwo(n//2)

    # TC: O(logN)
    # SC: O(1)
```

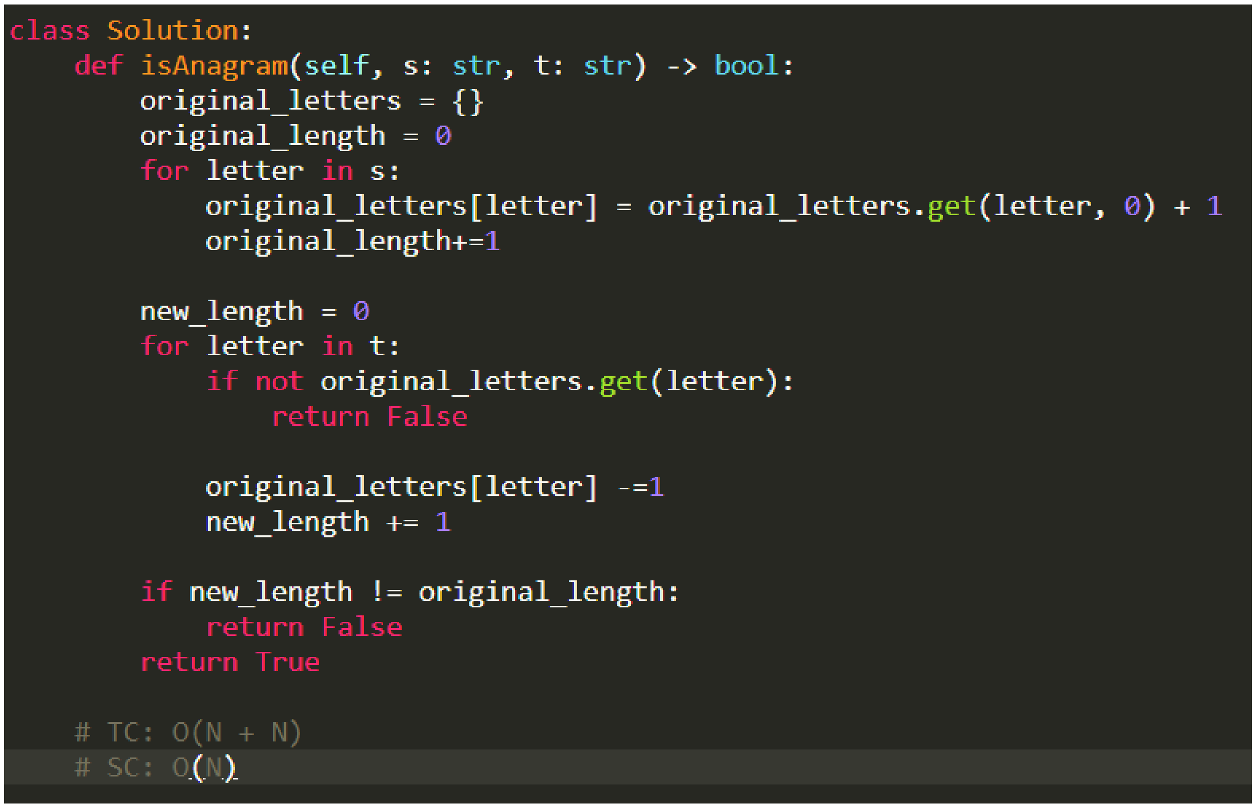
Optimal:
-> check using bit trick that n & (n-1) is 0 or not.



```python
class Solution:
    def isPowerOfTwo(self, n: int) -> bool:
        if n == 0 :
            return False

        return False if n & (n-1) else True

    # TC: O(1)
    # SC: O(1)
```

3. https://leetcode.com/problems/valid-anagram/

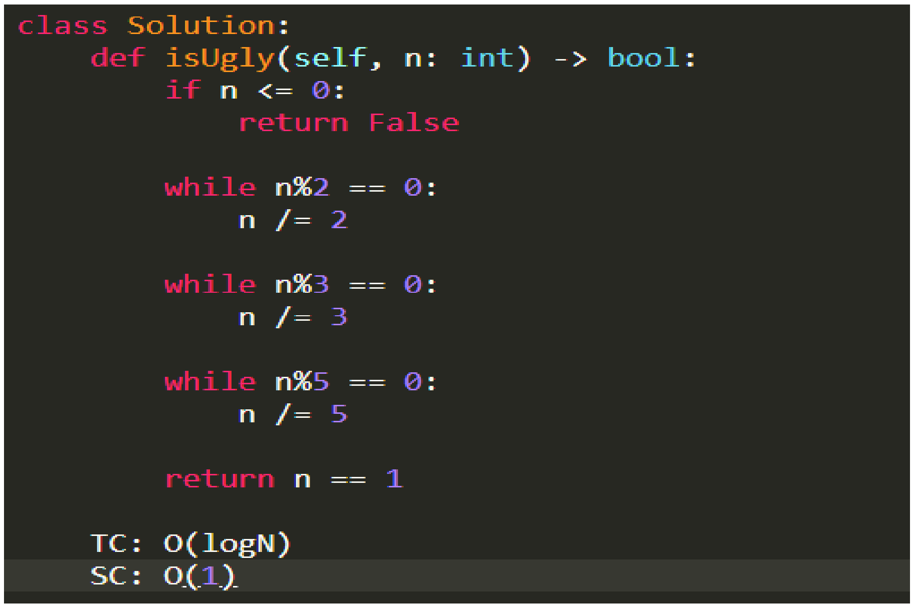
Brute force:
-> using hash map, hash the characters of original string and then iterate on other string and check if these characters are present in hash map and the length of both strings is same.



```
class Solution:
    def isAnagram(self, s: str, t: str) -> bool:
        original_letters = {}
        original_length = 0
        for letter in s:
            original_letters[letter] = original_letters.get(letter, 0) + 1
            original_length+=1

        new_length = 0
        for letter in t:
            if not original_letters.get(letter):
                return False

            original_letters[letter] -=1
            new_length += 1

        if new_length != original_length:
            return False
        return True

    # TC: O(N + N)
    # SC: O(N)
```

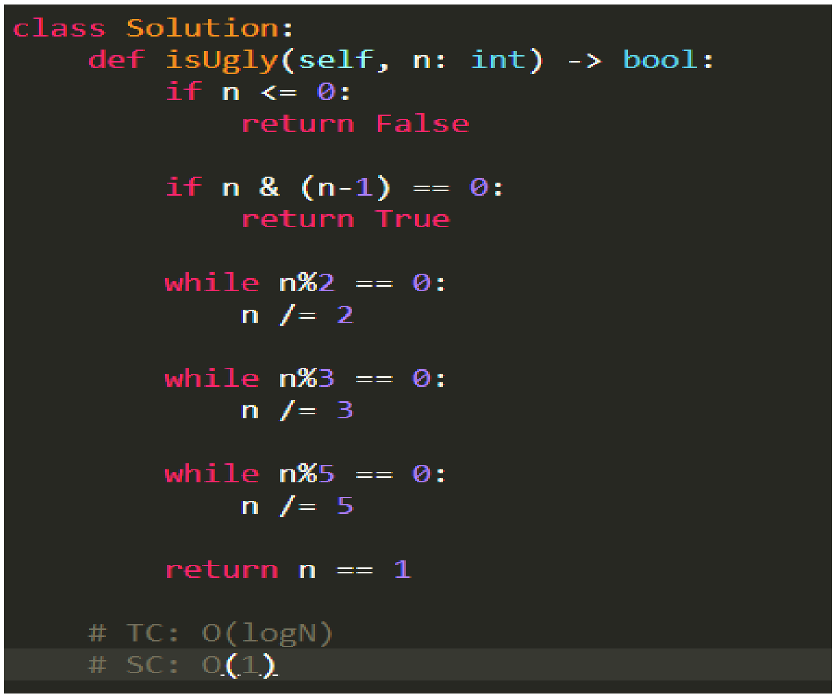
4. https://leetcode.com/problems/ugly-number/

Brute force:
-> check if no. is divisible by 2 then keep on dividing it until not possible, same for 3 and 5.
-> in the end if no. is 1, then true otherwise False



```
class Solution:
    def isUgly(self, n: int) -> bool:
        if n <= 0:
            return False

        while n%2 == 0:
            n /= 2

        while n%3 == 0:
            n /= 3

        while n%5 == 0:
            n /= 5

        return n == 1

    TC: O(logN)
    SC: O(1)
```

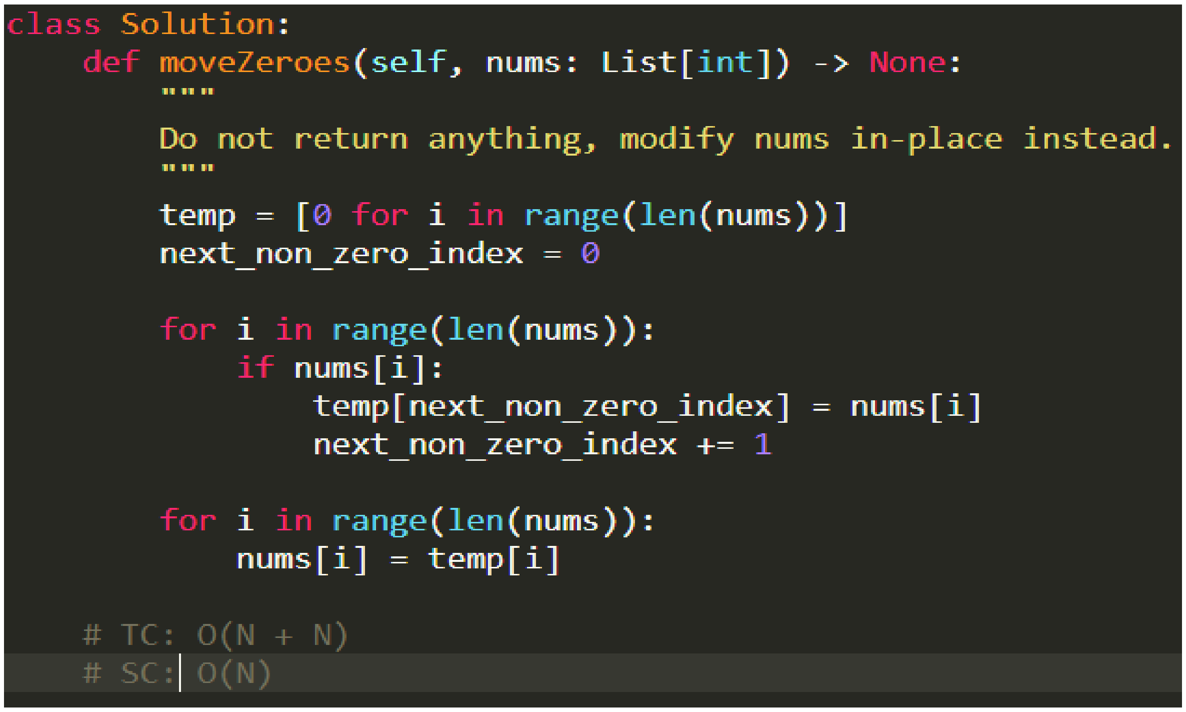
Better:
-> for a number which is a power of 2, save iterations for such numbers.



```
class Solution:
    def isUgly(self, n: int) -> bool:
        if n <= 0:
            return False

        if n & (n-1) == 0:
            return True

        while n%2 == 0:
            n /= 2

        while n%3 == 0:
            n /= 3

        while n%5 == 0:
            n /= 5

        return n == 1

    # TC: O(logN)
    # SC: O(1)
```

5. https://leetcode.com/problems/move-zeroes/

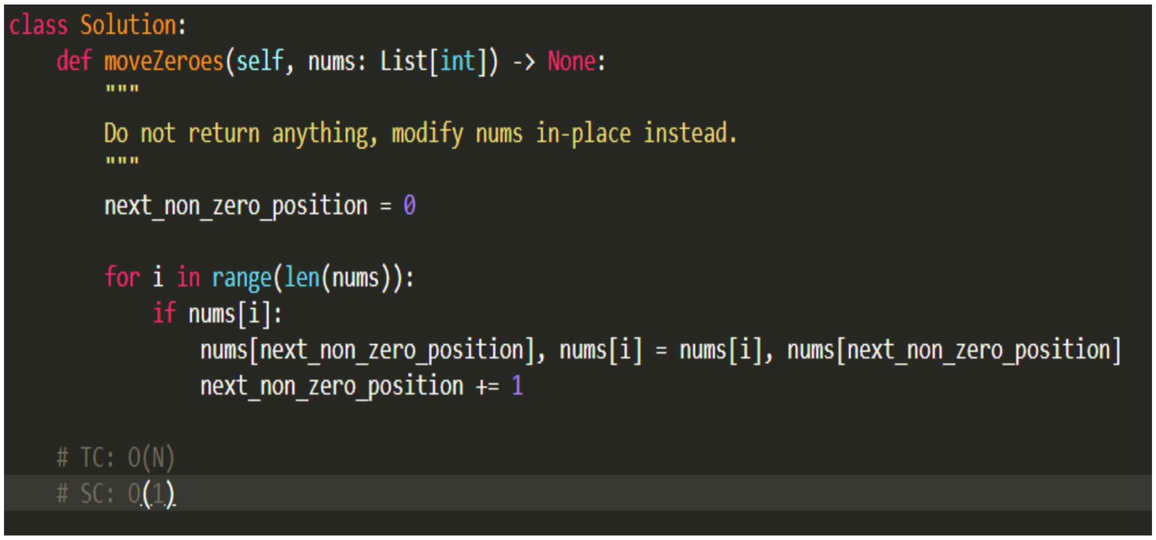
Brute:
-> maintain a temporary array, place all the non zero element in that array and copy them back to the original array.



```python
class Solution:
    def moveZeroes(self, nums: List[int]) -> None:
        """
        Do not return anything, modify nums in-place instead.
        """
        temp = [0 for i in range(len(nums))]
        next_non_zero_index = 0

        for i in range(len(nums)):
            if nums[i]:
                temp[next_non_zero_index] = nums[i]
                next_non_zero_index += 1

        for i in range(len(nums)):
            nums[i] = temp[i]

    # TC: O(N + N)
    # SC: O(N)
```

Optimal:

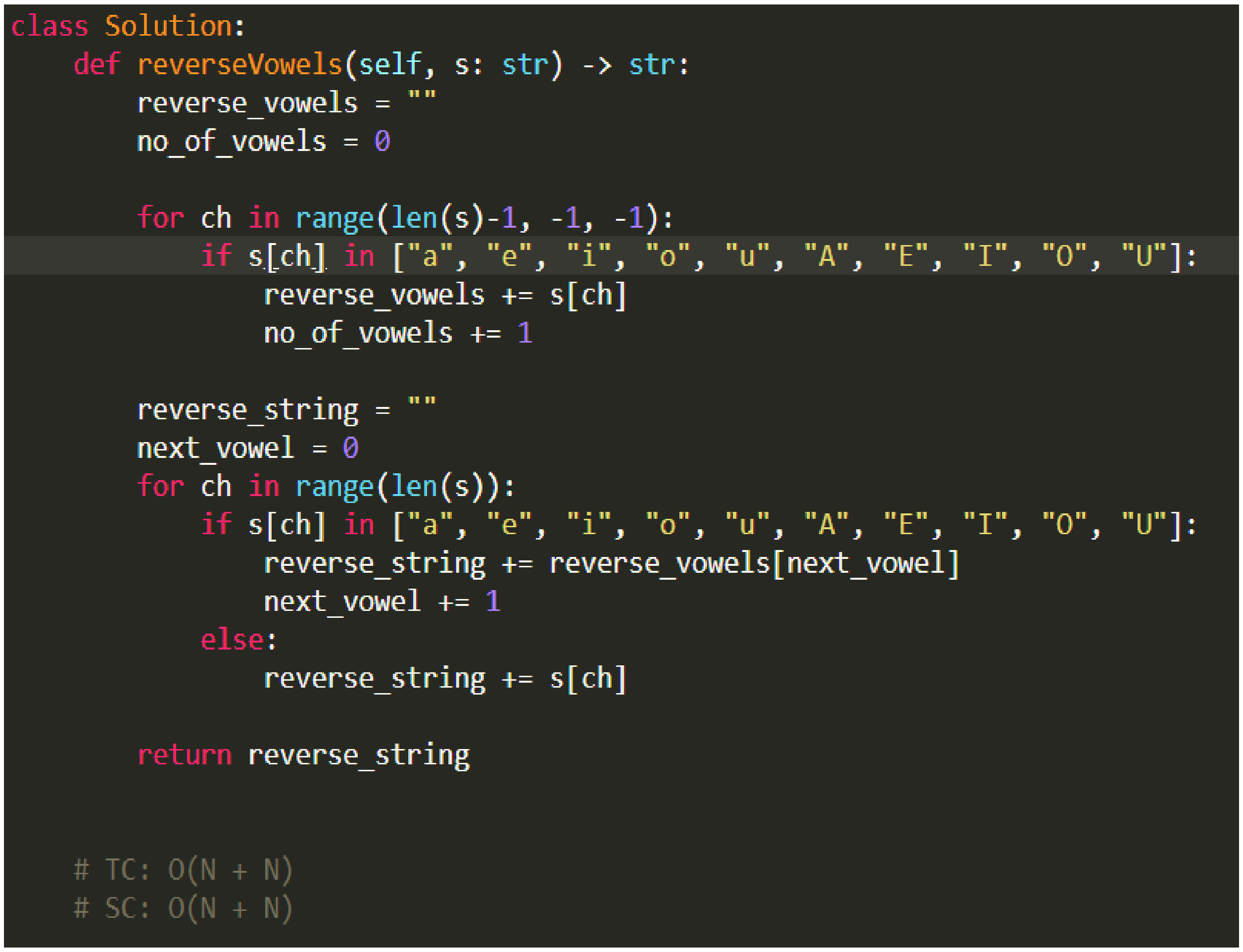
-> maintain a variable which signifies the position for a non zero element, as a non zero element is placed at this position, increment it by 1.



```python
class Solution:
    def moveZeroes(self, nums: List[int]) -> None:
        """
        Do not return anything, modify nums in-place instead.
        """
        next_non_zero_position = 0

        for i in range(len(nums)):
            if nums[i]:
                nums[next_non_zero_position], nums[i] = nums[i], nums[next_non_zero_position]
                next_non_zero_position += 1

    # TC: O(N)
    # SC: O(1)
```

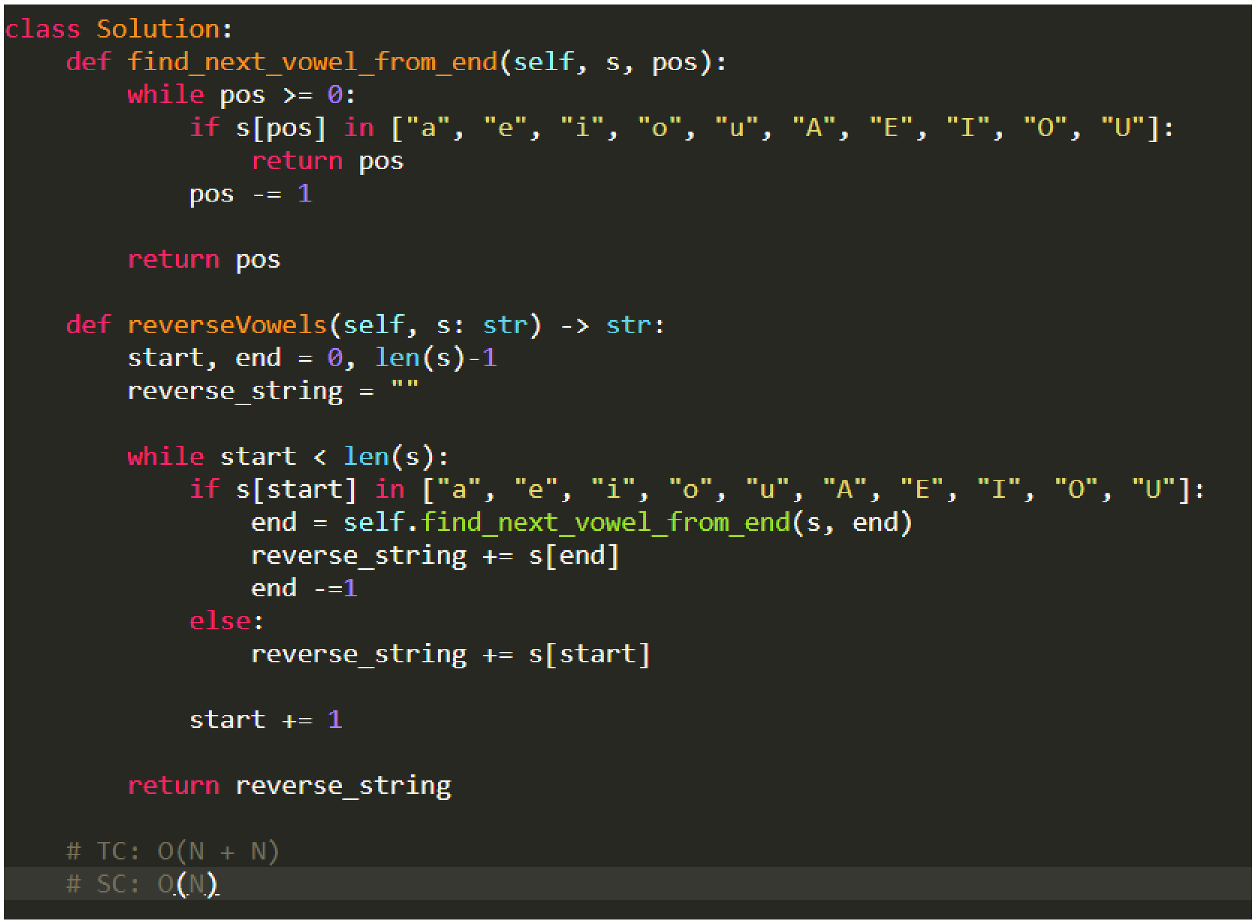
6. https://leetcode.com/problems/reverse-vowels-of-a-string/

Brute force:
-> maintain a string in which vowels are stored in reverse order.
-> again traverse the string, if vowels is encountered, pick a vowel from above string and place that character in the resultant string.



```python
class Solution:
    def reverseVowels(self, s: str) -> str:
        reverse_vowels = ""
        no_of_vowels = 0

        for ch in range(len(s)-1, -1, -1):
            if s[ch] in ["a", "e", "i", "o", "u", "A", "E", "I", "O", "U"]:
                reverse_vowels += s[ch]
                no_of_vowels += 1

        reverse_string = ""
        next_vowel = 0
        for ch in range(len(s)):
            if s[ch] in ["a", "e", "i", "o", "u", "A", "E", "I", "O", "U"]:
                reverse_string += reverse_vowels[next_vowel]
                next_vowel += 1
            else:
                reverse_string += s[ch]

        return reverse_string


    # TC: O(N + N)
    # SC: O(N + N)
```

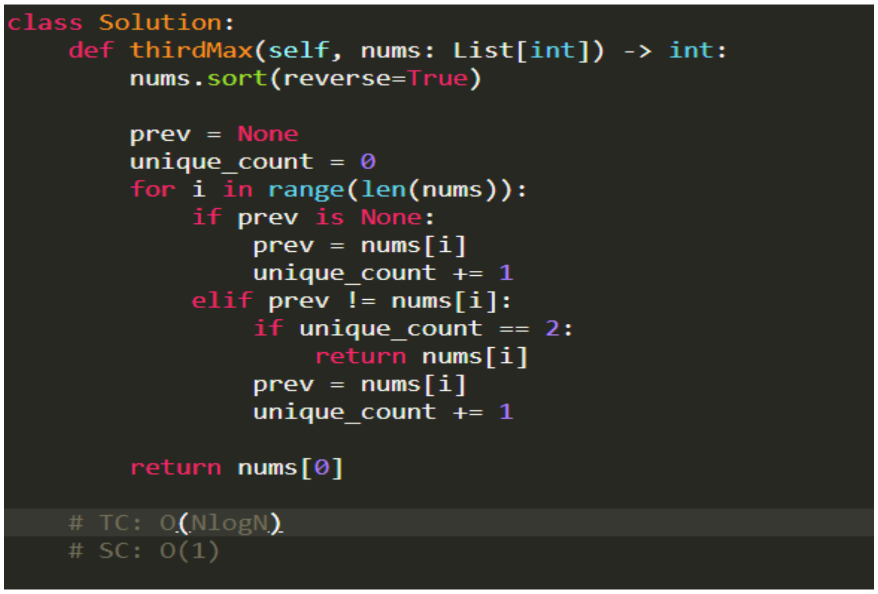
Better:
-> maintain two pointers, start and end, end will move to next vowel from end.
-> as start encounters a vowel, a method will update the end to the vowel from end, place character at end and continue this process.



```python
class Solution:
    def find_next_vowel_from_end(self, s, pos):
        while pos >= 0:
            if s[pos] in ["a", "e", "i", "o", "u", "A", "E", "I", "O", "U"]:
                return pos
            pos -= 1

        return pos

    def reverseVowels(self, s: str) -> str:
        start, end = 0, len(s)-1
        reverse_string = ""

        while start < len(s):
            if s[start] in ["a", "e", "i", "o", "u", "A", "E", "I", "O", "U"]:
                end = self.find_next_vowel_from_end(s, end)
                reverse_string += s[end]
                end -=1
            else:
                reverse_string += s[start]

            start += 1

        return reverse_string

    # TC: O(N + N)
    # SC: O(N)
```

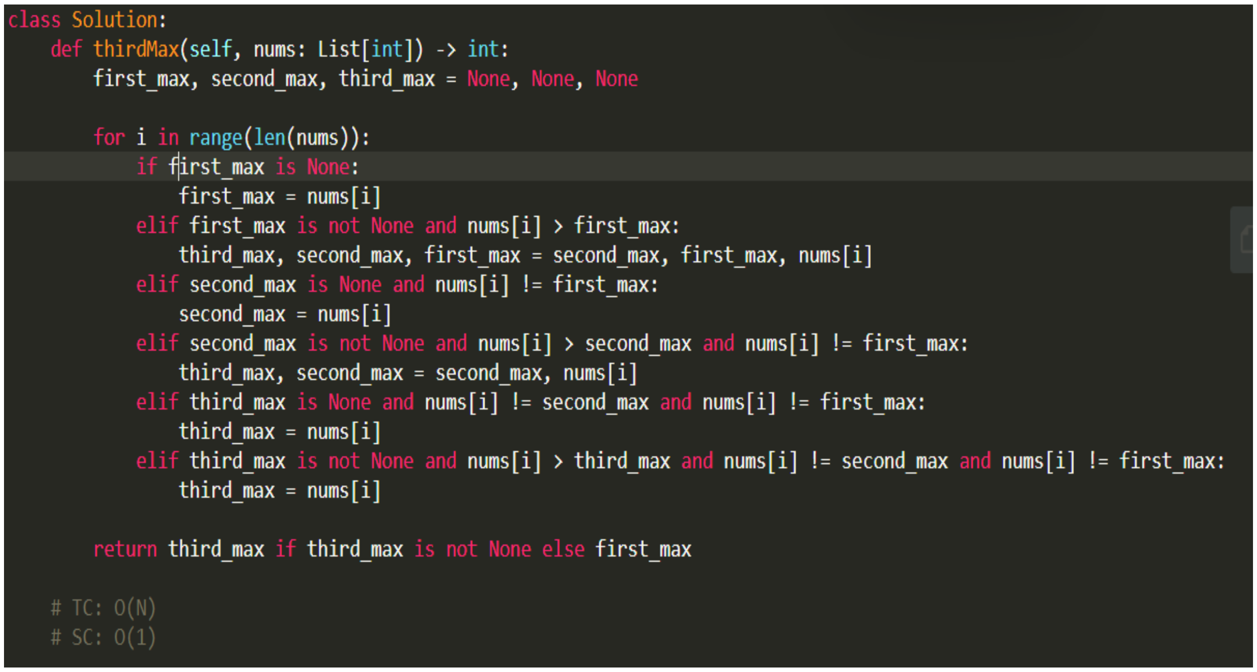
7. https://leetcode.com/problems/third-maximum-number/

Brute force:
-> sort the array in descending order and find the third unique number from beginning



```python
class Solution:
    def thirdMax(self, nums: List[int]) -> int:
        nums.sort(reverse=True)

        prev = None
        unique_count = 0
        for i in range(len(nums)):
            if prev is None:
                prev = nums[i]
                unique_count += 1
            elif prev != nums[i]:
                if unique_count == 2:
                    return nums[i]
                prev = nums[i]
                unique_count += 1

        return nums[0]

    # TC: O(NlogN)
    # SC: O(1)
```

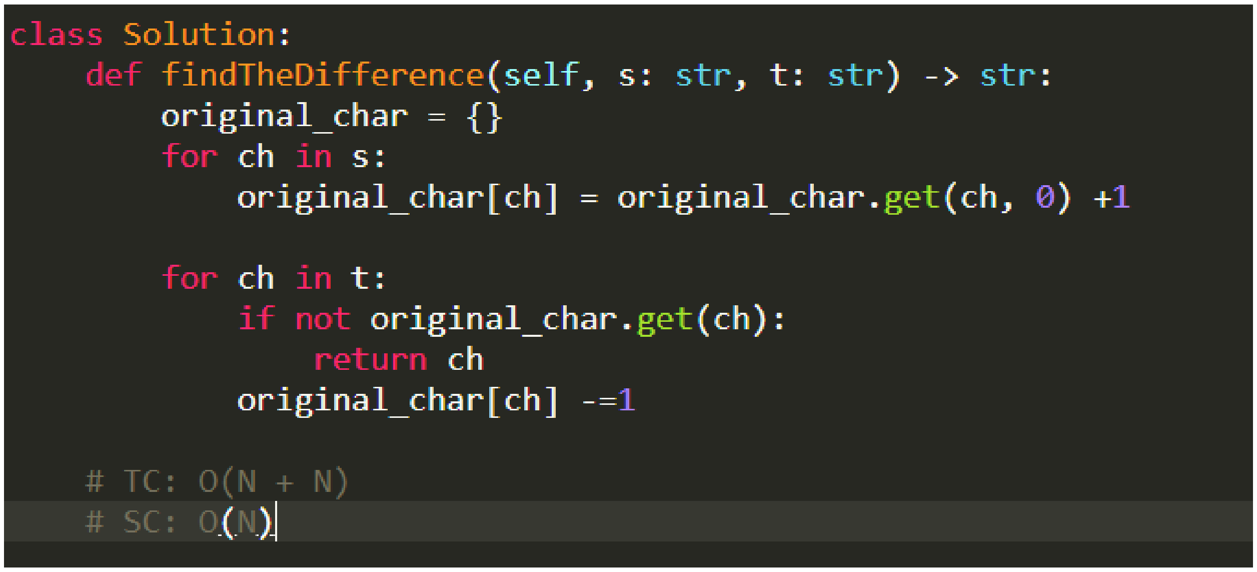
Better:
-> maintain three pointers first_max, second_max and third_max.
-> iterate on array and accordingly assign values to each pointer but make sure that there cannot be same value for any two pointers.



```python
class Solution:
    def thirdMax(self, nums: List[int]) -> int:
        first_max, second_max, third_max = None, None, None

        for i in range(len(nums)):
            if first_max is None:
                first_max = nums[i]
            elif first_max is not None and nums[i] > first_max:
                third_max, second_max, first_max = second_max, first_max, nums[i]
            elif second_max is None and nums[i] != first_max:
                second_max = nums[i]
            elif second_max is not None and nums[i] > second_max and nums[i] != first_max:
                third_max, second_max = second_max, nums[i]
            elif third_max is None and nums[i] != second_max and nums[i] != first_max:
                third_max = nums[i]
            elif third_max is not None and nums[i] > third_max and nums[i] != second_max and nums[i] != first_max:
                third_max = nums[i]

        return third_max if third_max is not None else first_max

    # TC: O(N)
    # SC: O(1)
```

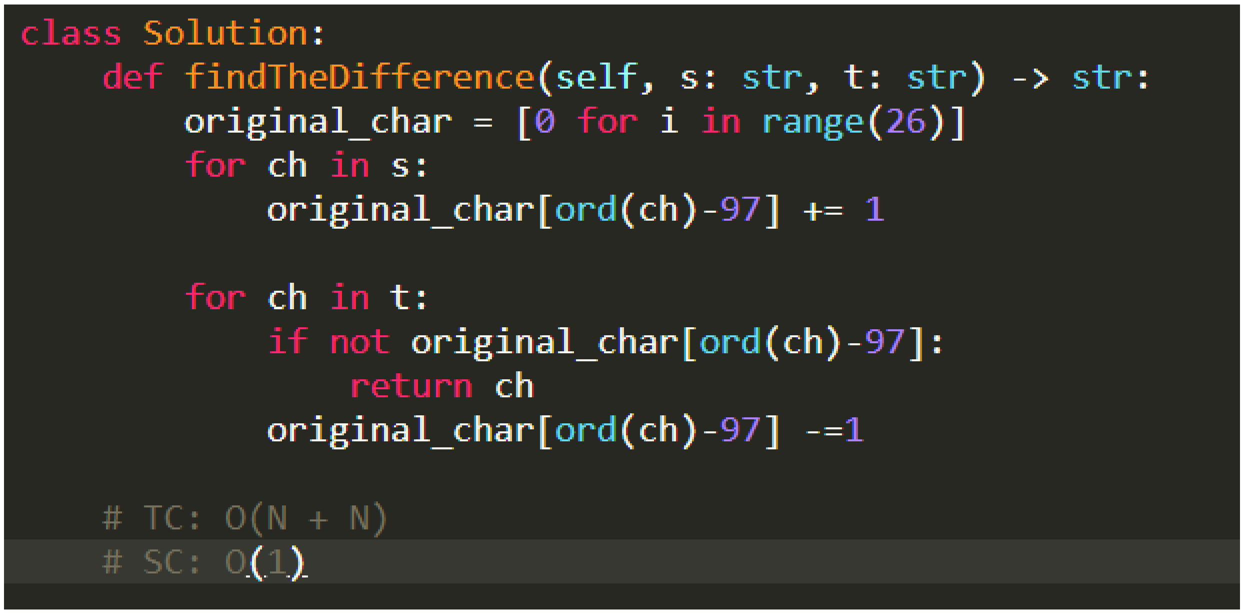
8. https://leetcode.com/problems/find-the-difference/

Brute force:
-> maintain a hashmap or array, storing the count of original characters in s.
-> traverse through it, and if any character not found in hash map or their count is 0, return that character.



```python
class Solution:
    def findTheDifference(self, s: str, t: str) -> str:
        original_char = {}
        for ch in s:
            original_char[ch] = original_char.get(ch, 0) +1

        for ch in t:
            if not original_char.get(ch):
                return ch
            original_char[ch] -=1

    # TC: O(N + N)
    # SC: O(N)
```

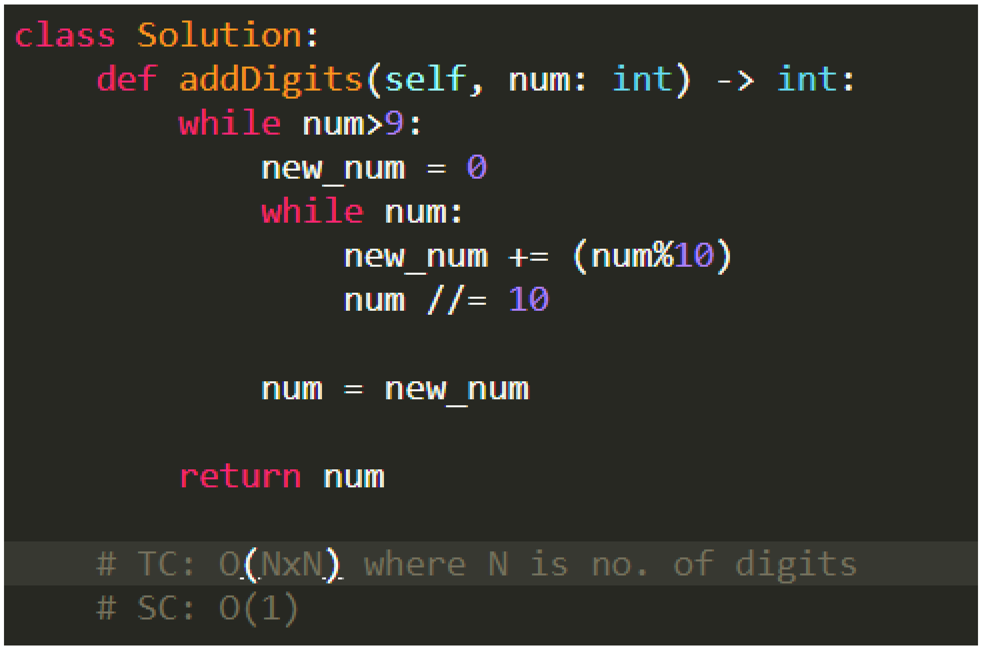
-> better:
Instead of hash map, we can use fixed size array to hash characters.



```python
class Solution:
    def findTheDifference(self, s: str, t: str) -> str:
        original_char = [0 for i in range(26)]
        for ch in s:
            original_char[ord(ch)-97] += 1

        for ch in t:
            if not original_char[ord(ch)-97]:
                return ch
            original_char[ord(ch)-97] -=1

    # TC: O(N + N)
    # SC: O(1)
```

9. https://leetcode.com/problems/add-digits/

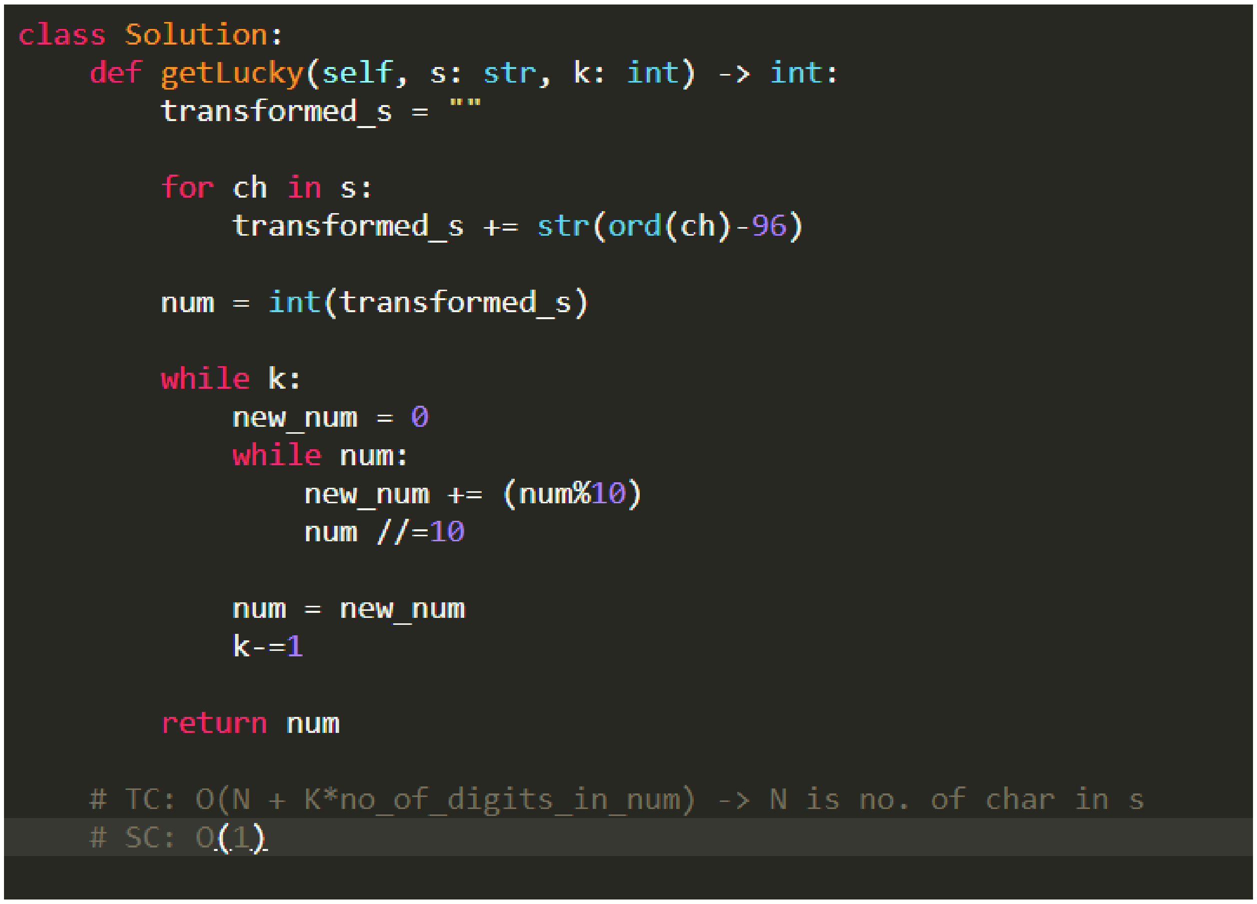
Brute force:
-> keep on adding digits of number until number becomes less than 10



```
class Solution:
    def addDigits(self, num: int) -> int:
        while num>9:
            new_num = 0
            while num:
                new_num += (num%10)
                num //= 10

            num = new_num

        return num

    # TC: O(NxN) where N is no. of digits
    # SC: O(1)
```

10. https://leetcode.com/problems/sum-of-digits-of-string-after-convert/

Brute force:
-> convert all char into int and form a new string, then start adding its digit and for some new number, keep forming new number k times.



```python
class Solution:
    def getLucky(self, s: str, k: int) -> int:
        transformed_s = ""

        for ch in s:
            transformed_s += str(ord(ch)-96)

        num = int(transformed_s)

        while k:
            new_num = 0
            while num:
                new_num += (num%10)
                num //=10

            num = new_num
            k-=1

        return num

    # TC: O(N + K*no_of_digits_in_num) -> N is no. of char in s
    # SC: O(1)
```