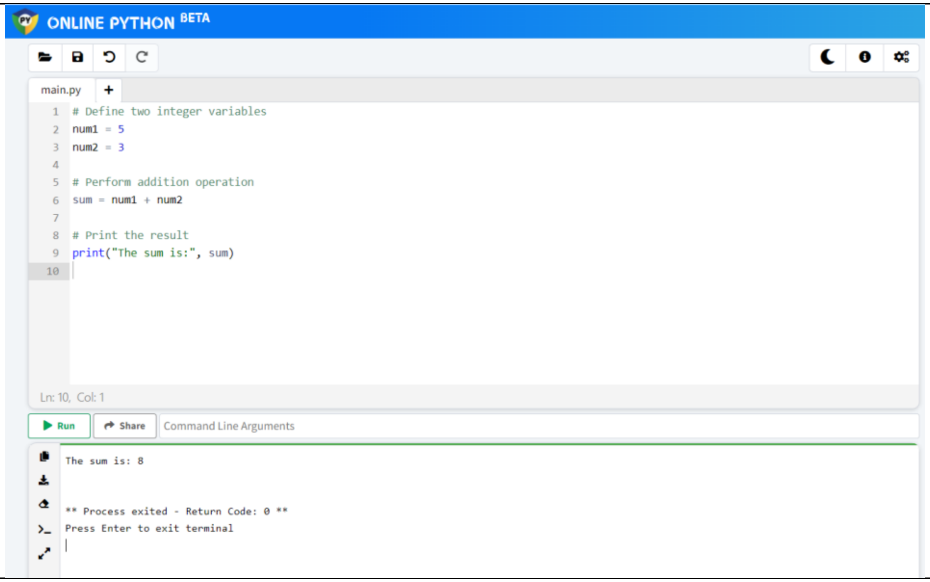
Koda ģenerēšana – Meta AI

Instruction	Output	Koda rezultāti

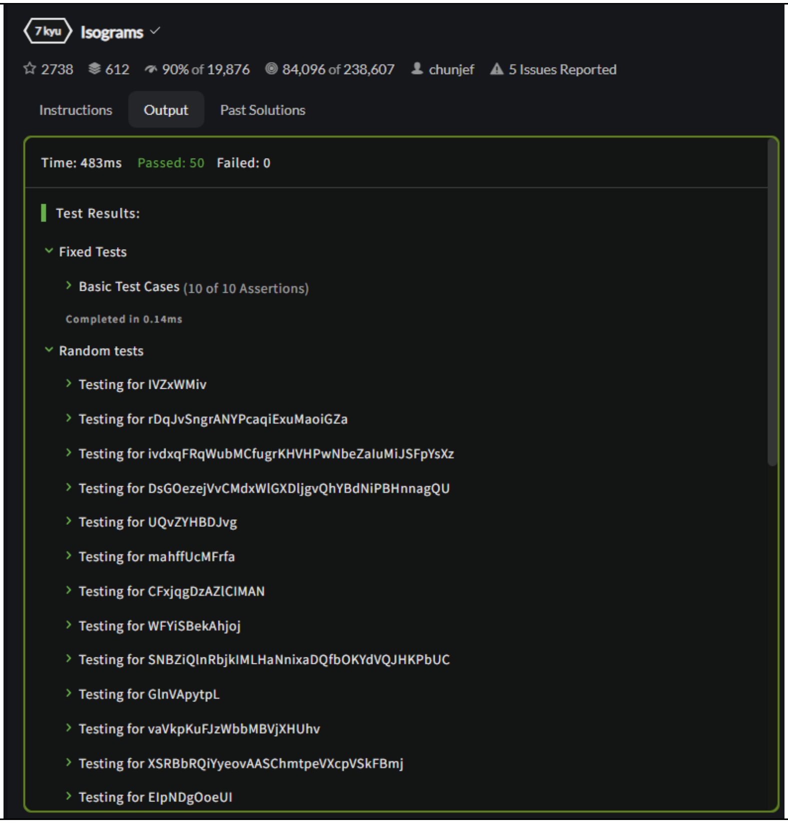
Please generate Python code that defines two integer variables and then does a simple addition operation.	# Define two integer variables
	num1 = 5
	num2 = 3
	
	# Perform addition operation
	sum = num1 + num2
	
	# Print the result
	print("The sum is:", sum)
		

```
# Define two integer variables
num1 = 5
num2 = 3

# Perform addition operation
sum = num1 + num2

# Print the result
print("The sum is:", sum)
```

```
The sum is: 8


** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

An isogram is a word that has no repeating letters, consecutive or non-consecutive. Implement a function that determines whether a string that contains only letters is an isogram. Assume the empty string is an isogram. Ignore letter case.

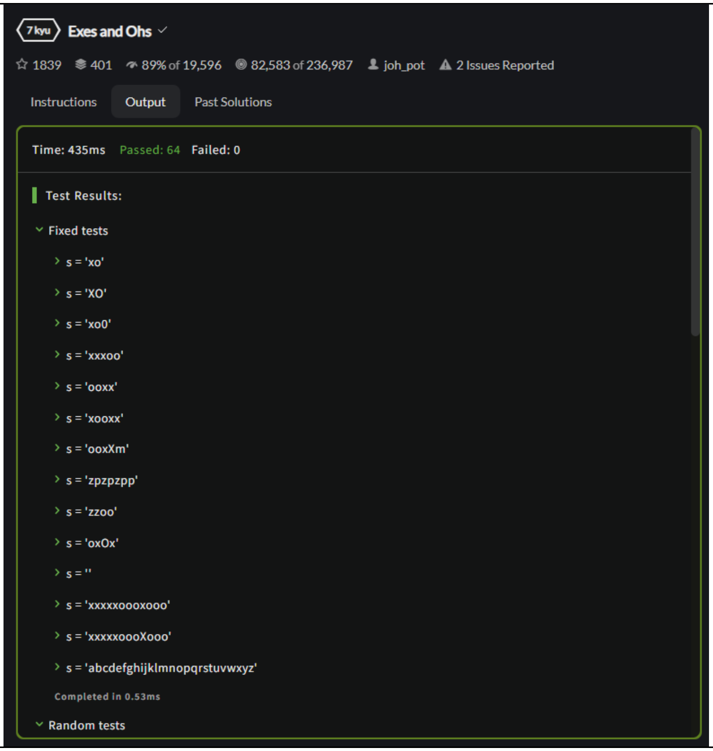
```python
def is_isogram(string):
    """
    Determine if a string is an isogram (has no repeating letters).

    Args:
        string (str): The input string.

    Returns:
        bool: True if the string is an isogram, False otherwise.
    """
    # Convert the string to lowercase to ignore letter case
    string = string.lower()
    # Create a set from the string, which automatically removes duplicates
    unique_letters = set(string)
    # Check if the length of the set is equal to the length of the original string
    return len(unique_letters) == len(string)
```



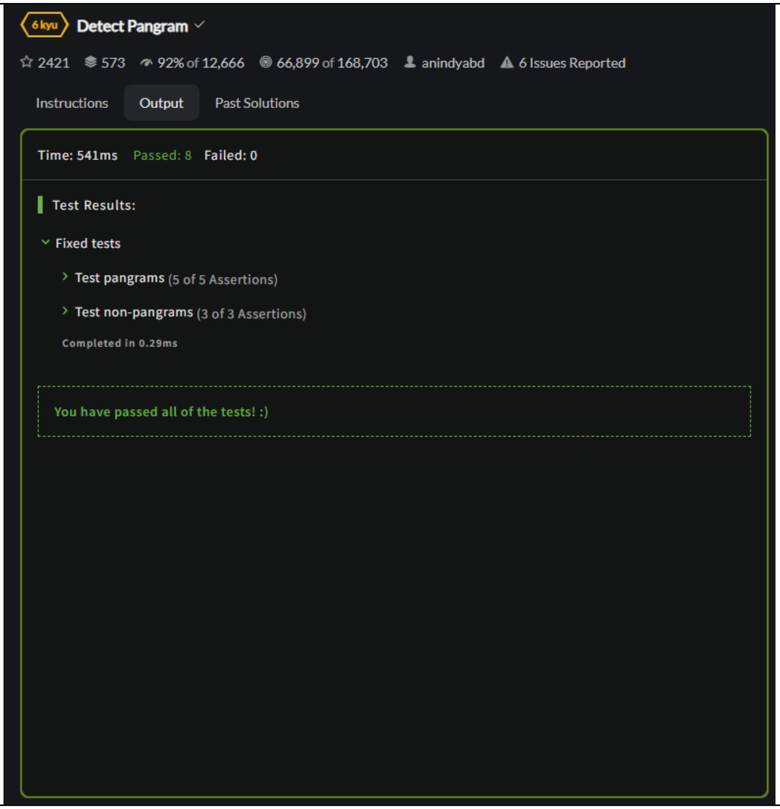
```
Time: 483ms   Passed: 50   Failed: 0

Test Results:
  Fixed Tests
    Basic Test Cases (10 of 10 Assertions)
    Completed in 0.14ms
  Random tests
    Testing for IVZxWMiv
    Testing for rDqJvSngrANYPcaqiExuMaoiGZa
    Testing for ivdxqFRqWubMCfugrKHVHPwNbeZaIuMiJSFpYsXz
    Testing for DsGOezejVvCMdxWlGXDljgvQhYBdNiPBHnnagQU
    Testing for UQvZYHBDJvg
    Testing for mahffUcMFrfa
    Testing for CFxjqgDzAZlCIMAN
    Testing for WFYiSBekAhjoj
    Testing for SNBZiQlnRbjkIMLHaNnixaDQfbOKYdVQJHKPbUC
    Testing for GlnVApytpL
    Testing for vaVkpKuFJzWbbMBVjXHUhv
    Testing for XSRBbRQiYyeovAASChmtpeVXcpVSkFBmj
    Testing for ElpNDgOoeUI
```

Check to see if a string has the same amount of 'x's and 'o's. The method must return a boolean and be case insensitive. The string can contain any char.

```
def has_equal_x_and_o(string):
    """
    Check if a string has the same amount of 'x's and 'o's.

    Args:
        string (str): The input string.

    Returns:
        bool: True if the string has equal 'x's and 'o's, False otherwise.
    """
    # Convert the string to lowercase to ignore case
    string = string.lower()
    # Count the number of 'x's and 'o's
    x_count = string.count('x')
    o_count = string.count('o')
    # Return True if the counts are equal, False otherwise
    return x_count == o_count
```

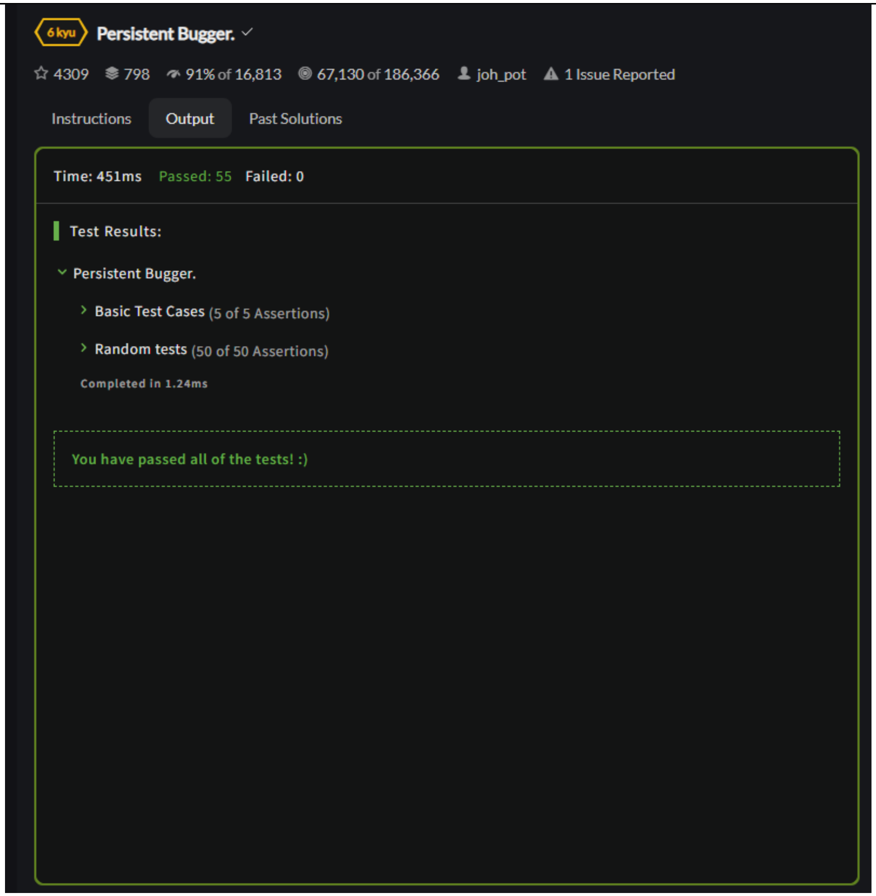


```
Time: 435ms   Passed: 64   Failed: 0

Test Results:
  Fixed tests
    s = 'xo'
    s = 'XO'
    s = 'xo0'
    s = 'xxxoo'
    s = 'ooxx'
    s = 'xooxx'
    s = 'ooxXm'
    s = 'zpzpzpp'
    s = 'zzoo'
    s = 'oxOx'
    s = ''
    s = 'xxxxxoooxooo'
    s = 'xxxxxoooXooo'
    s = 'abcdefghijklmnopqrstuvwxyz'
    Completed in 0.53ms
  Random tests
```

A pangram is a sentence that contains every single letter of the alphabet at least once. For example, the sentence "The quick brown fox jumps over the lazy dog" is a pangram, because it uses the letters A-Z at least once (case is irrelevant).
Given a string, detect whether or not it is a pangram. Return True if it is, False if not. Ignore numbers and punctuation
	def is_pangram(string):
	    """
	    Detect whether a string is a pangram (contains every letter of the alphabet at least once).
	
	    Args:
	        string (str): The input string.
	
	    Returns:
	        bool: True if the string is a pangram, False otherwise.
	    """
	    # Convert the string to lowercase and remove non-alphabetic characters
	    alphabet = "abcdefghijklmnopqrstuvwxyz"
	    string = "".join(filter(str.isalpha, string.lower()))
	    # Check if every letter of the alphabet is in the string
	    return set(alphabet) <= set(string)
			
			6 kyu Detect Pangram
			2421 573 92% of 12,666 66,899 of 168,703 anindyabd 6 Issues Reported
			Instructions Output Past Solutions
			Time: 541ms Passed: 8 Failed: 0
			Test Results:
			Fixed tests
			Test pangrams (5 of 5 Assertions)
			Test non-pangrams (3 of 3 Assertions)
			Completed in 0.29ms
			You have passed all of the tests! :)

Write a function, persistence, that takes in a positive parameter num and returns its multiplicative persistence, which is the number of times you must multiply the digits in num until you reach a single digit.

```python
def persistence(num):
    """
    Calculate the multiplicative persistence of a number.

    Args:
        num (int): A positive integer.

    Returns:
        int: The multiplicative persistence of the number.
    """
    # Convert the number to a string to easily access its digits
    num_str = str(num)
    # Initialize the persistence counter
    persistence_count = 0
    # Loop until the number is reduced to a single digit
    while len(num_str) > 1:
        # Initialize the product of the digits
        product = 1
        # Calculate the product of the digits
        for digit in num_str:
            product *= int(digit)
        # Update the number string and increment the persistence counter
        num_str = str(product)
```



Persistent Bugger.

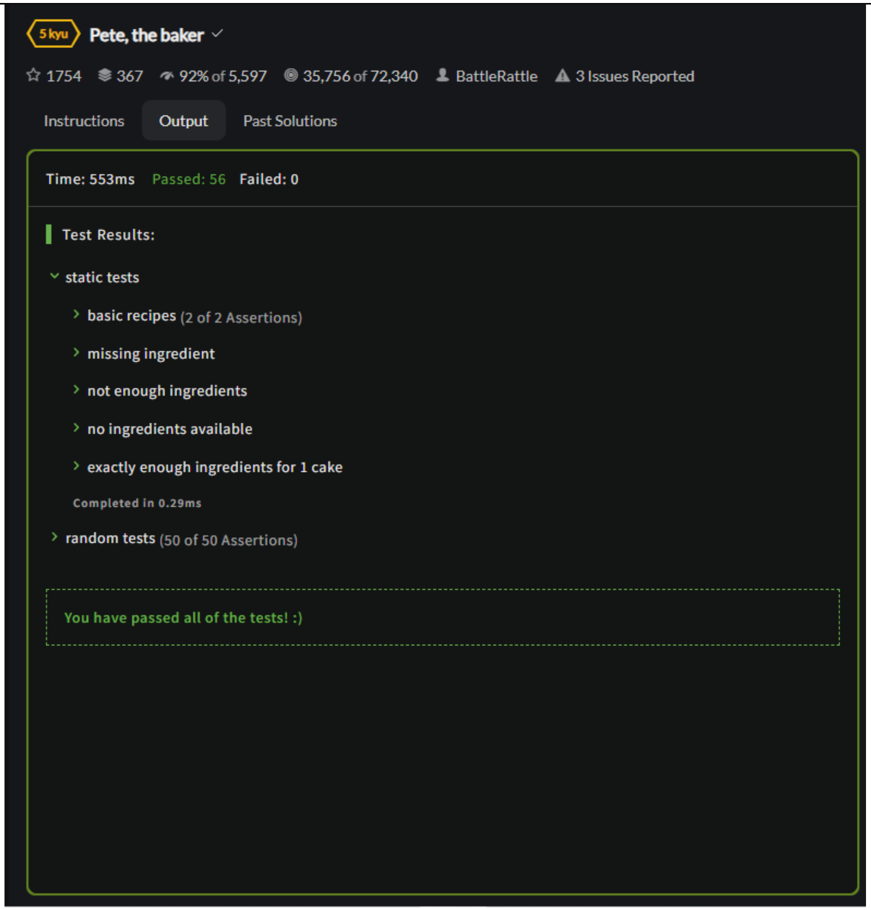
☆ 4309 798 91% of 16,813 67,130 of 186,366 joh_pot ⚠ 1 Issue Reported

```
Time: 451ms   Passed: 55   Failed: 0

Test Results:
  Persistent Bugger.
    > Basic Test Cases (5 of 5 Assertions)
    > Random tests (50 of 50 Assertions)
    Completed in 1.24ms

You have passed all of the tests! :)
```

```
        persistence_count += 1
    # Return the persistence
count
    return persistence_count
```

Write a function cakes(), which takes the recipe (object) and the available ingredients (also an object) and returns the maximum number of cakes Pete can bake (integer). For simplicity there are no units for the amounts (e.g. 1 lb of flour or 200 g of sugar are simply 1 or 200). Ingredients that are not present in the objects, can be considered as 0.

```
def cakes(recipe, available):
    """
    Calculate the maximum number
of cakes that can be baked.

    Args:
        recipe (dict): A dictionary
containing the recipe (e.g. {"flour":
500, "sugar": 200, "eggs": 1}).
        available (dict): A dictionary
containing the available ingredients
(e.g. {"flour": 1200, "sugar": 1200,
"eggs": 5, "milk": 3000}).

    Returns:
        int: The maximum number of
cakes that can be baked.
    """
    # Initialize the maximum number
of cakes to infinity
    max_cakes = float('inf')
    # Iterate over the ingredients in
the recipe
    for ingredient, amount in
recipe.items():
        # Get the available amount of
the ingredient (default to 0 if not
present)
        available_amount =
available.get(ingredient, 0)
        # Calculate the number of
cakes that can be baked with this
ingredient
```



Time: 553ms Passed: 56 Failed: 0

Test Results:

static tests
- basic recipes (2 of 2 Assertions)
- missing ingredient
- not enough ingredients
- no ingredients available
- exactly enough ingredients for 1 cake

Completed in 0.29ms

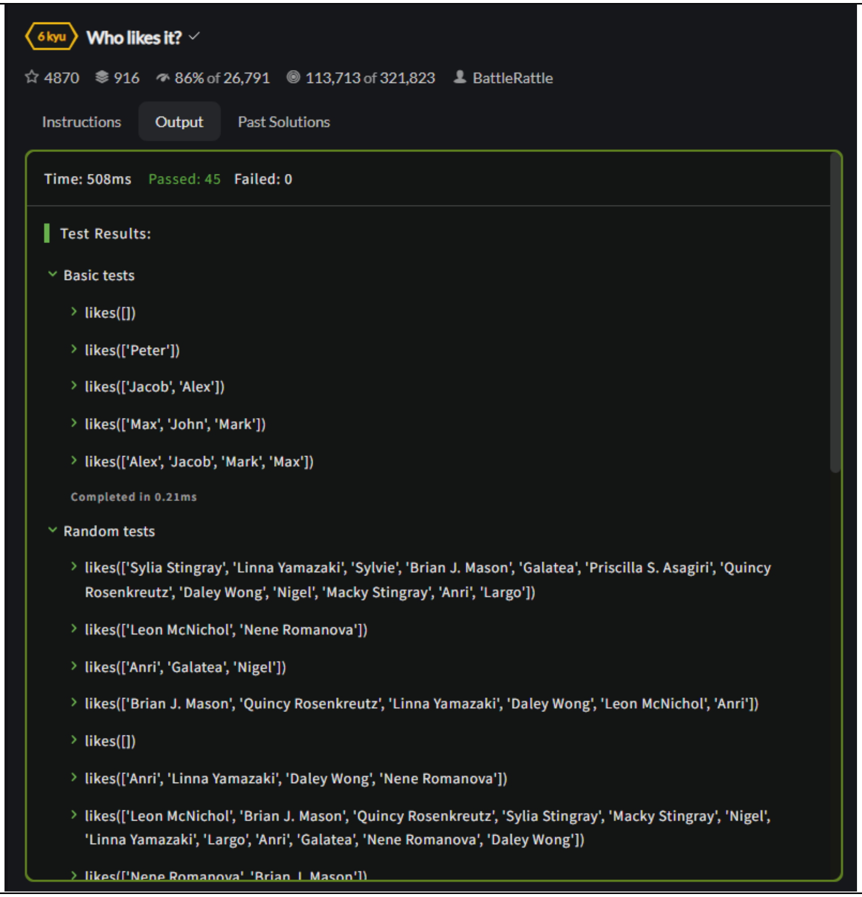
random tests (50 of 50 Assertions)

You have passed all of the tests! :)

```
        cakes_with_ingredient = available_amount // amount
        # Update the maximum number of cakes
        max_cakes = min(max_cakes, cakes_with_ingredient)
    # Return the maximum number of cakes
    return max_cakes
```

You probably know the "like" system from Facebook and other pages. People can "like" blog posts, pictures or other items. We want to create the text that should be displayed next to such an item.
Implement the function which takes an array containing the names of people that like an item. It must return the display text as shown in the examples

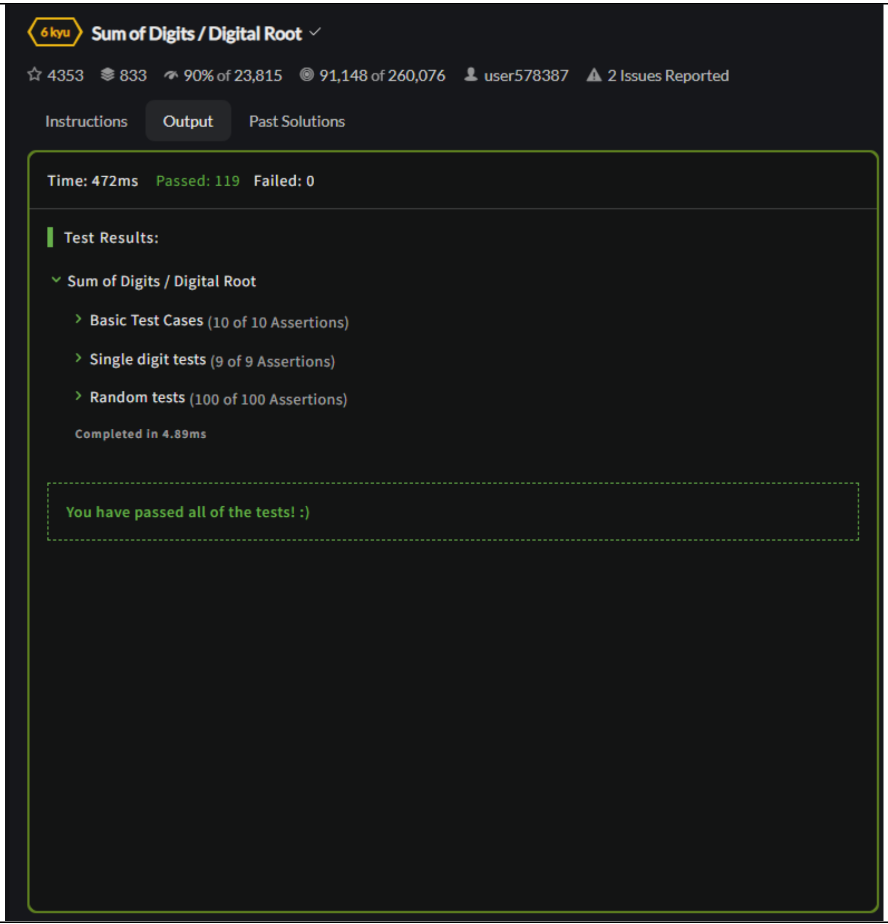
```
def likes(names):
    """
    Generate the display text for the
"like" system.

    Args:
        names (list): A list of names of
people who like the item.

    Returns:
        str: The display text.
    """
    if len(names) == 0:
        return "no one likes this"
    elif len(names) == 1:
        return "{} likes
this".format(names[0])
    elif len(names) == 2:
        return "{} and {} like
this".format(names[0], names[1])
    elif len(names) == 3:
        return "{}, {} and {} like
this".format(names[0], names[1],
names[2])
    else:
        return "{}, {} and {} others like
this".format(names[0], names[1],
len(names) - 2)
```



```
Time: 508ms   Passed: 45   Failed: 0

Test Results:
  Basic tests
    likes([])
    likes(['Peter'])
    likes(['Jacob', 'Alex'])
    likes(['Max', 'John', 'Mark'])
    likes(['Alex', 'Jacob', 'Mark', 'Max'])
    Completed in 0.21ms
  Random tests
    likes(['Sylia Stingray', 'Linna Yamazaki', 'Sylvie', 'Brian J. Mason', 'Galatea', 'Priscilla S. Asagiri', 'Quincy
    Rosenkreutz', 'Daley Wong', 'Nigel', 'Macky Stingray', 'Anri', 'Largo'])
    likes(['Leon McNichol', 'Nene Romanova'])
    likes(['Anri', 'Galatea', 'Nigel'])
    likes(['Brian J. Mason', 'Quincy Rosenkreutz', 'Linna Yamazaki', 'Daley Wong', 'Leon McNichol', 'Anri'])
    likes([])
    likes(['Anri', 'Linna Yamazaki', 'Daley Wong', 'Nene Romanova'])
    likes(['Leon McNichol', 'Brian J. Mason', 'Quincy Rosenkreutz', 'Sylia Stingray', 'Macky Stingray', 'Nigel',
    'Linna Yamazaki', 'Largo', 'Anri', 'Galatea', 'Nene Romanova', 'Daley Wong'])
    likes(['Nene Romanova', 'Brian J. Mason'])
```

Digital root is the recursive sum of all the digits in a number.
Given n, take the sum of the digits of n. If that value has more than one digit, continue reducing in this way until a single-digit number is produced. The input will be a non-negative integer.

```
def digital_root(n):
    """
    Calculate the digital root of a number.

    Args:
        n (int): A non-negative integer.

    Returns:
        int: The digital root of the number.
    """
    while n > 9:
        n = sum(int(digit) for digit in str(n))
    return n
```



Time: 472ms Passed: 119 Failed: 0

Test Results:

Sum of Digits / Digital Root
- Basic Test Cases (10 of 10 Assertions)
- Single digit tests (9 of 9 Assertions)
- Random tests (100 of 100 Assertions)

Completed in 4.89ms

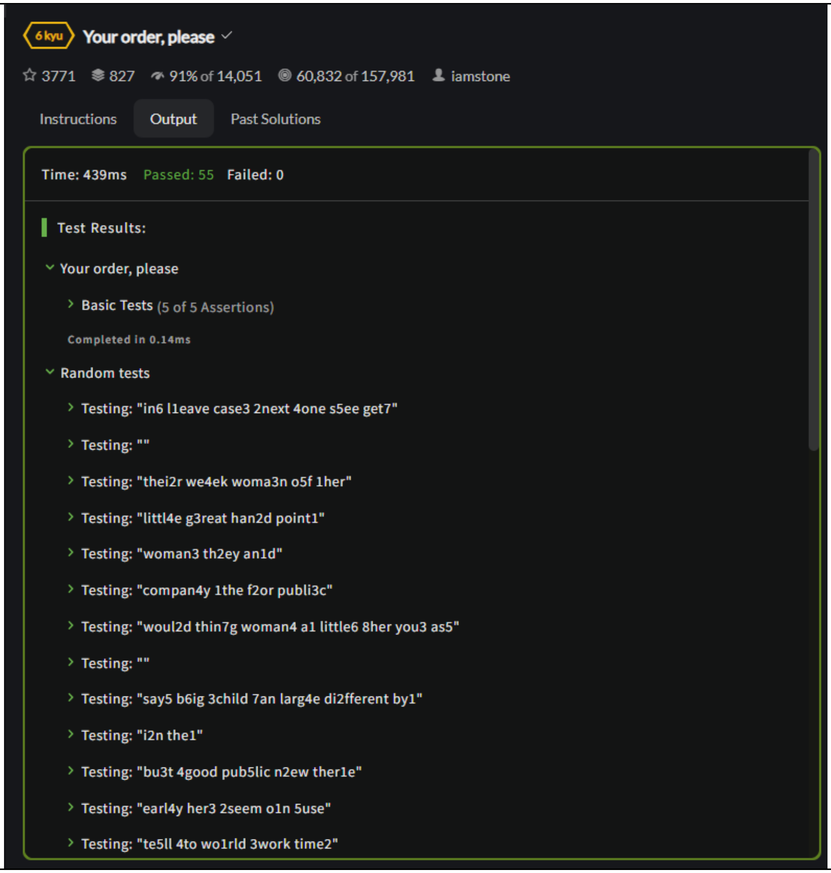
You have passed all of the tests! :)

Your task is to sort a given string. Each word in the string will contain a single number. This number is the position the word should have in the result.

Note: Numbers can be from 1 to 9. So 1 will be the first word (not 0).

If the input string is empty, return an empty string. The words in the input String will only contain valid consecutive numbers.

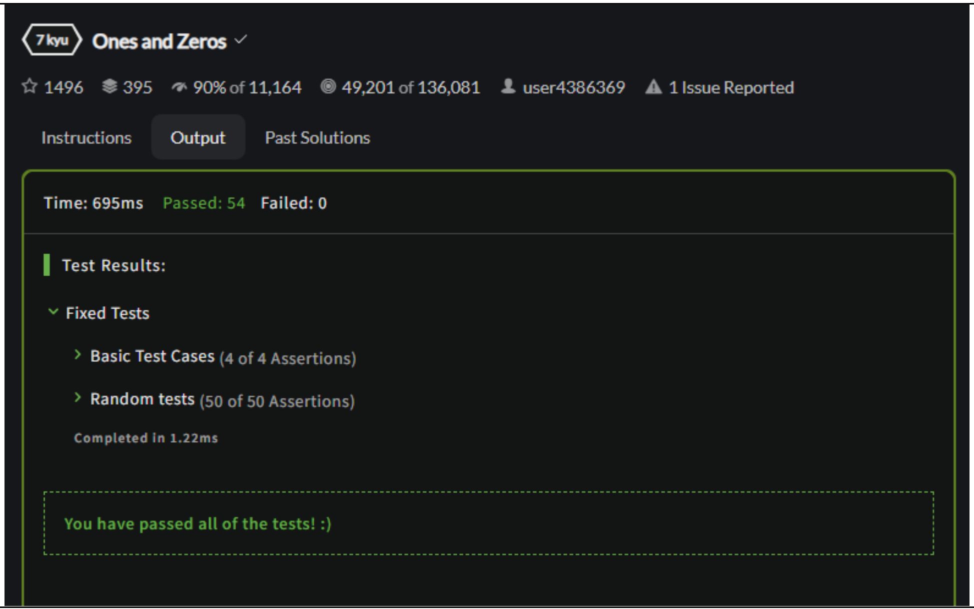
```
def order_words(s):
    """
    Sort a string based on the
numbers in each word.

    Args:
        s (str): The input string.

    Returns:
        str: The sorted string.
    """
    if not s:
        return ""
    words = s.split()
    sorted_words = [""] * len(words)
    for word in words:
        for char in word:
            if char.isdigit():
                sorted_words[int(char) -
1] = word
                break
    return " ".join(sorted_words)
```



```
Time: 439ms   Passed: 55   Failed: 0

Test Results:
  Your order, please
    > Basic Tests (5 of 5 Assertions)
    Completed in 0.14ms
  Random tests
    > Testing: "in6 l1eave case3 2next 4one s5ee get7"
    > Testing: ""
    > Testing: "thei2r we4ek woma3n o5f 1her"
    > Testing: "littl4e g3reat han2d point1"
    > Testing: "woman3 th2ey an1d"
    > Testing: "compan4y 1the f2or publi3c"
    > Testing: "woul2d thin7g woman4 a1 little6 8her you3 as5"
    > Testing: ""
    > Testing: "say5 b6ig 3child 7an larg4e di2fferent by1"
    > Testing: "i2n the1"
    > Testing: "bu3t 4good pub5lic n2ew ther1e"
    > Testing: "earl4y her3 2seem o1n 5use"
    > Testing: "te5ll 4to wo1rld 3work time2"
```

Given an array of ones and zeroes, convert the equivalent binary value to an integer. Eg: [0, 0, 0, 1] is treated as 0001 which is the binary representation of 1.

```
def binary_array_to_number(arr):
    """
    Convert an array of binary digits to an integer.

    Args:
        arr (list): A list of binary digits (0s and 1s).

    Returns:
        int: The equivalent integer value.
    """
    return int("".join(map(str, arr)), 2)
```



7 kyu **Ones and Zeros**

☆ 1496 · 395 · 90% of 11,164 · 49,201 of 136,081 · user4386369 · ⚠ 1 Issue Reported

Time: 695ms Passed: 54 Failed: 0

Test Results:

Fixed Tests

Basic Test Cases (4 of 4 Assertions)

Random tests (50 of 50 Assertions)

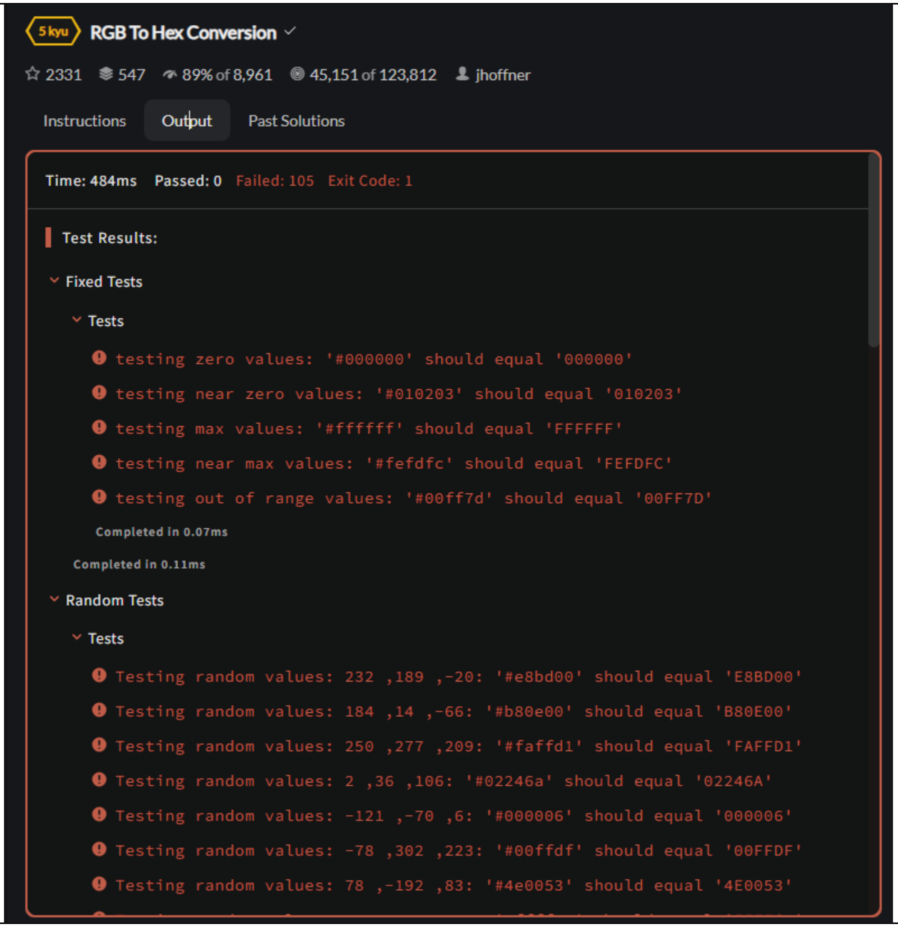
Completed in 1.22ms

You have passed all of the tests! :)

The rgb function is incomplete. Complete it so that passing in RGB decimal values will result in a hexadecimal representation being returned. Valid decimal values for RGB are 0 - 255. Any values that fall out of that range must be rounded to the closest valid value.

Note: Your answer should always be 6 characters long, the shorthand with 3 will not work here.

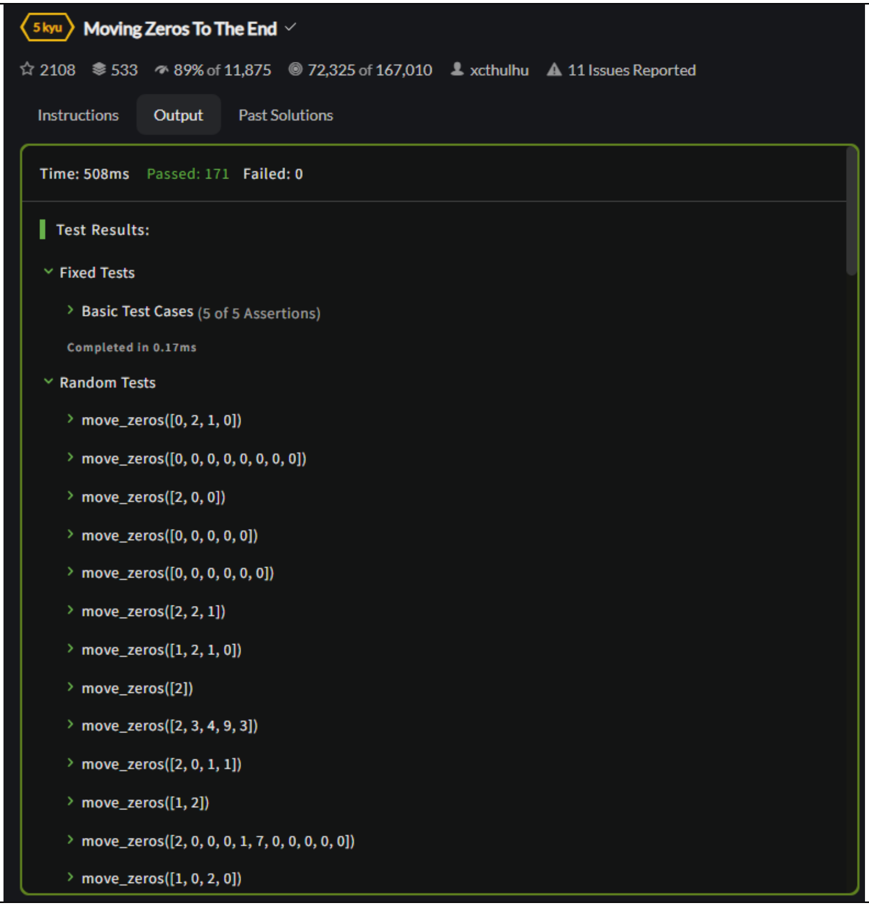
```python
def rgb(r, g, b):
    """
    Convert RGB decimal values to a hexadecimal representation.

    Args:
        r (int): The red value (0-255).
        g (int): The green value (0-255).
        b (int): The blue value (0-255).

    Returns:
        str: The hexadecimal representation of the RGB values.
    """
    # Ensure values are within the valid range (0-255)
    r = min(max(r, 0), 255)
    g = min(max(g, 0), 255)
    b = min(max(b, 0), 255)

    # Convert decimal values to hexadecimal
    hex_r = "{:02x}".format(r)
    hex_g = "{:02x}".format(g)
    hex_b = "{:02x}".format(b)

    # Return the hexadecimal representation
    return "#" + hex_r + hex_g + hex_b
```



```
Time: 484ms   Passed: 0   Failed: 105   Exit Code: 1

Test Results:
  Fixed Tests
    Tests
      testing zero values: '#000000' should equal '000000'
      testing near zero values: '#010203' should equal '010203'
      testing max values: '#ffffff' should equal 'FFFFFF'
      testing near max values: '#fefdfc' should equal 'FEFDFC'
      testing out of range values: '#00ff7d' should equal '00FF7D'
      Completed in 0.07ms
    Completed in 0.11ms
  Random Tests
    Tests
      Testing random values: 232 ,189 ,-20: '#e8bd00' should equal 'E8BD00'
      Testing random values: 184 ,14 ,-66: '#b80e00' should equal 'B80E00'
      Testing random values: 250 ,277 ,209: '#faffd1' should equal 'FAFFD1'
      Testing random values: 2 ,36 ,106: '#02246a' should equal '02246A'
      Testing random values: -121 ,-70 ,6: '#000006' should equal '000006'
      Testing random values: -78 ,302 ,223: '#00ffdf' should equal '00FFDF'
      Testing random values: 78 ,-192 ,83: '#4e0053' should equal '4E0053'
```

Write an algorithm that takes an array and moves all of the zeros to the end, preserving the order of the other elements.

```python
def move_zeros_to_end(arr):
    """
    Move all zeros to the end of an array, preserving the order of other elements.

    Args:
        arr (list): The input array.

    Returns:
        list: The modified array with zeros moved to the end.
    """
    # Separate the array into two lists: non-zeros and zeros
    non_zeros = [x for x in arr if x != 0]
    zeros = [x for x in arr if x == 0]

    # Combine the non-zeros and zeros lists
    return non_zeros + zeros
```



Time: 508ms Passed: 171 Failed: 0

Test Results:

Fixed Tests

Basic Test Cases (5 of 5 Assertions)

Completed in 0.17ms

Random Tests

move_zeros([0, 2, 1, 0])

move_zeros([0, 0, 0, 0, 0, 0, 0, 0])

move_zeros([2, 0, 0])

move_zeros([0, 0, 0, 0, 0])

move_zeros([0, 0, 0, 0, 0, 0])

move_zeros([2, 2, 1])

move_zeros([1, 2, 1, 0])

move_zeros([2])

move_zeros([2, 3, 4, 9, 3])

move_zeros([2, 0, 1, 1])

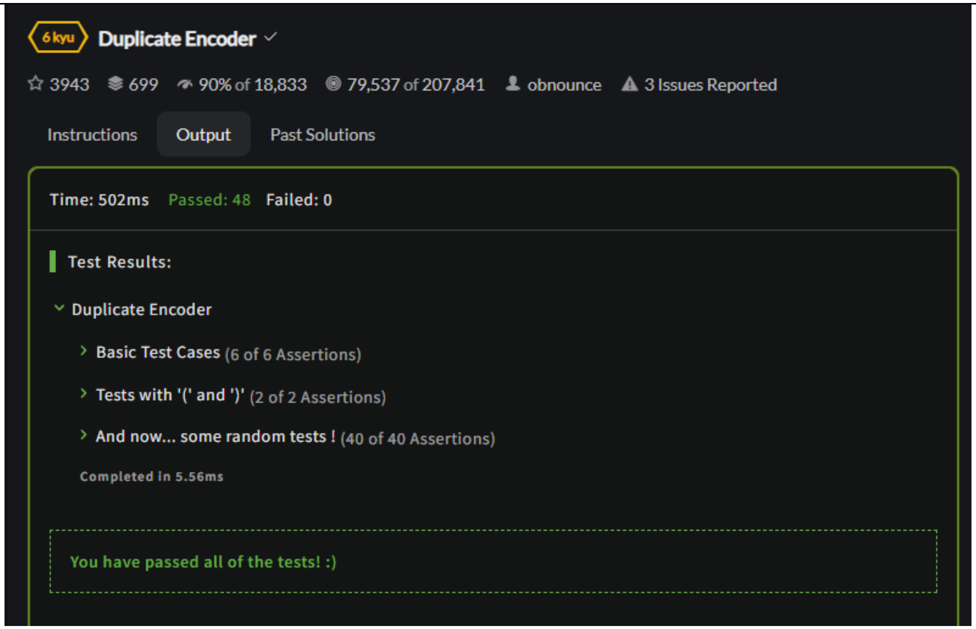
move_zeros([1, 2])

move_zeros([2, 0, 0, 0, 1, 7, 0, 0, 0, 0, 0])

move_zeros([1, 0, 2, 0])

The goal of this exercise is to convert a string to a new string where each character in the new string is "(" if that character appears only once in the original string, or ")" if that character appears more than once in the original string. Ignore capitalization when determining if a character is a duplicate.

```
def duplicate_encode(word):
    """
    Convert a string to a new string where each character is "(" if it appears only once,
    or ")" if it appears more than once.

    Args:
        word (str): The input string.

    Returns:
        str: The modified string with "(" and ")" characters.
    """
    word = word.lower()
    count_dict = {}
    for char in word:
        if char in count_dict:
            count_dict[char] += 1
        else:
            count_dict[char] = 1
    result = ""
    for char in word:
        if count_dict[char] == 1:
            result += "("
        else:
            result += ")"
    return result
```



Time: 502ms Passed: 48 Failed: 0

Test Results:

Duplicate Encoder

Basic Test Cases (6 of 6 Assertions)

Tests with '(' and ')' (2 of 2 Assertions)

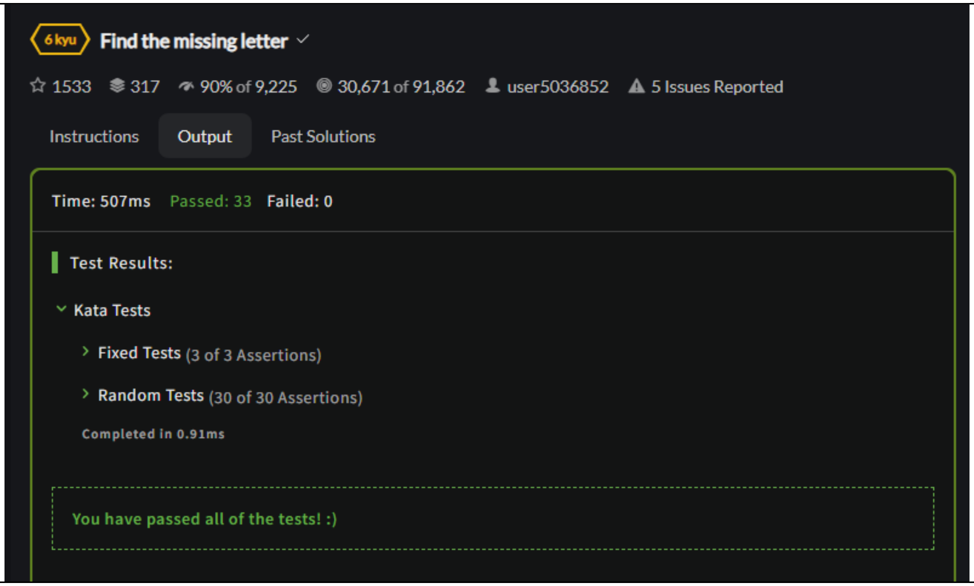
And now... some random tests ! (40 of 40 Assertions)

Completed in 5.56ms

You have passed all of the tests! :)

Write a method that takes an array of consecutive (increasing) letters as input and that returns the missing letter in the array. You will always get an valid array. And it will be always exactly one letter be missing. The length of the array will always be at least 2. The array will always contain letters in only one case.

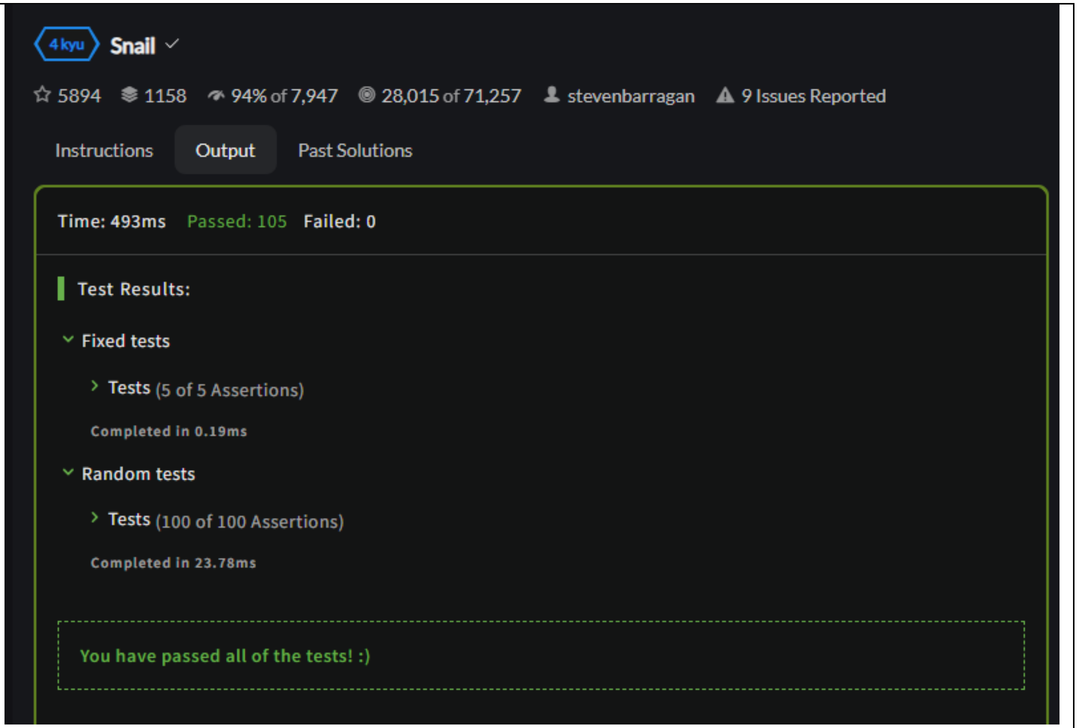
```
def find_missing_letter(chars):
    """
    Find the missing letter in an array of consecutive letters.

    Args:
        chars (list): The input array of consecutive letters.

    Returns:
        str: The missing letter.
    """
    for i in range(len(chars) - 1):
        if ord(chars[i + 1]) - ord(chars[i]) > 1:
            return chr(ord(chars[i]) + 1)
```



```
Time: 507ms   Passed: 33   Failed: 0

Test Results:
  Kata Tests
    Fixed Tests (3 of 3 Assertions)
    Random Tests (30 of 30 Assertions)
    Completed in 0.91ms

You have passed all of the tests! :)
```

Given an n x n array, return the array elements arranged from outermost elements to the middle element, traveling clockwise.

```
def arrange_clockwise(arr):
    """
    Arrange the array elements from
outermost to the middle element,
traveling clockwise.

    Args:
        arr (list): The input 2D array.

    Returns:
        list: The arranged array
elements.
    """
    result = []
    while arr:
        # Append the first row
        result += arr.pop(0)
        # Append the last column
        if arr and arr[0]:
            for row in arr:
                result.append(row.pop())
        # Append the last row in
reverse order
        if arr:
            result += arr.pop()[::-1]
        # Append the first column in
reverse order
        if arr and arr[0]:
            for row in arr[::-1]:
                result.append(row.pop(0))
    return result
```



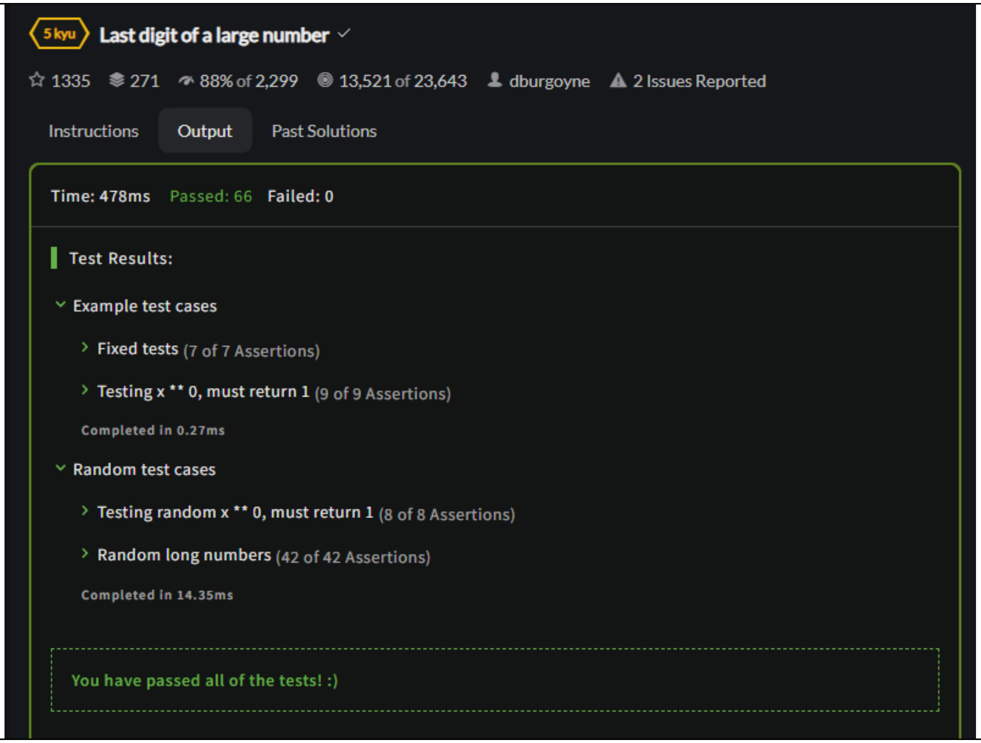
```
Time: 493ms   Passed: 105   Failed: 0

Test Results:
  Fixed tests
    Tests (5 of 5 Assertions)
    Completed in 0.19ms
  Random tests
    Tests (100 of 100 Assertions)
    Completed in 23.78ms

You have passed all of the tests! :)
```

Define a function that takes in two non-negative integers �a and �b and returns the last decimal digit of ��ab. Note that �a and �b may be very large!

For example, the last decimal digit of 9797 is 99, since 97=478296997=4782969. The last decimal digit of (2200)2300(2200)2300, which has over 10921092 decimal digits, is 66. Also, please take 0000 to be 11.

You may assume that the input will always be valid.

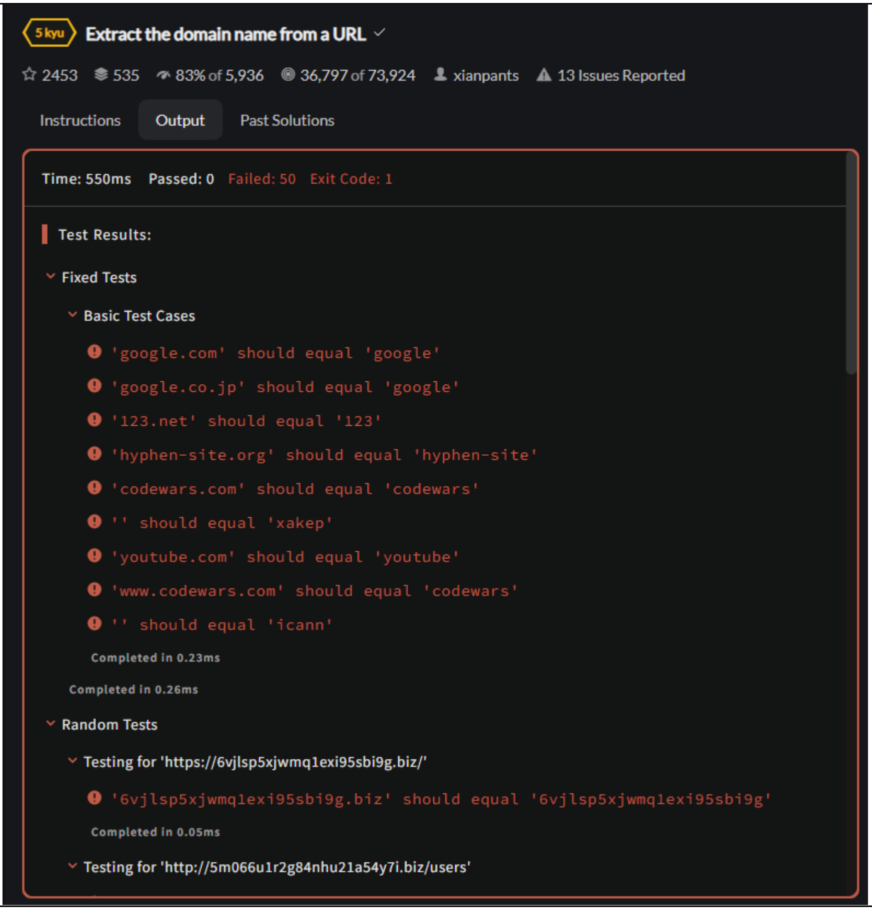
```
def last_decimal_digit(a, b):
    """
    Return the last decimal digit of a^b.

    Args:
        a (int): The base number.
        b (int): The exponent.

    Returns:
        int: The last decimal digit of a^b.
    """
    # Calculate the last decimal digit using modular arithmetic
    return pow(a, b, 10)
```



```
Time: 478ms   Passed: 66   Failed: 0

Test Results:
  Example test cases
    Fixed tests (7 of 7 Assertions)
    Testing x ** 0, must return 1 (9 of 9 Assertions)
    Completed in 0.27ms
  Random test cases
    Testing random x ** 0, must return 1 (8 of 8 Assertions)
    Random long numbers (42 of 42 Assertions)
    Completed in 14.35ms

You have passed all of the tests! :)
```

Write a function that when given a URL as a string, parses out just the domain name and returns it as a string.

```python
import urllib.parse

def extract_domain(url):
    """
    Extract the domain name from a URL.

    Args:
        url (str): The input URL.

    Returns:
        str: The domain name.
    """
    parsed_uri = urllib.parse.urlparse(url)
    domain = '{uri.netloc}'.format(uri=parsed_uri)
    return domain
```



```
Time: 550ms   Passed: 0   Failed: 50   Exit Code: 1

Test Results:
  Fixed Tests
    Basic Test Cases
      'google.com' should equal 'google'
      'google.co.jp' should equal 'google'
      '123.net' should equal '123'
      'hyphen-site.org' should equal 'hyphen-site'
      'codewars.com' should equal 'codewars'
      '' should equal 'xakep'
      'youtube.com' should equal 'youtube'
      'www.codewars.com' should equal 'codewars'
      '' should equal 'icann'
      Completed in 0.23ms
    Completed in 0.26ms
  Random Tests
    Testing for 'https://6vjlsp5xjwmq1exi95sbi9g.biz/'
      '6vjlsp5xjwmq1exi95sbi9g.biz' should equal '6vjlsp5xjwmq1exi95sbi9g'
      Completed in 0.05ms
    Testing for 'http://5m066u1r2g84nhu21a54y7i.biz/users'
```

Write two functions that convert a roman numeral to and from an integer value. Multiple roman numeral values will be tested for each function.

Modern Roman numerals are written by expressing each digit separately starting with the left most digit and skipping any digit with a value of zero. In Roman numerals:

1990 is

rendered: 1000=M, 900=CM, 90=XC; resulting in MCMXC

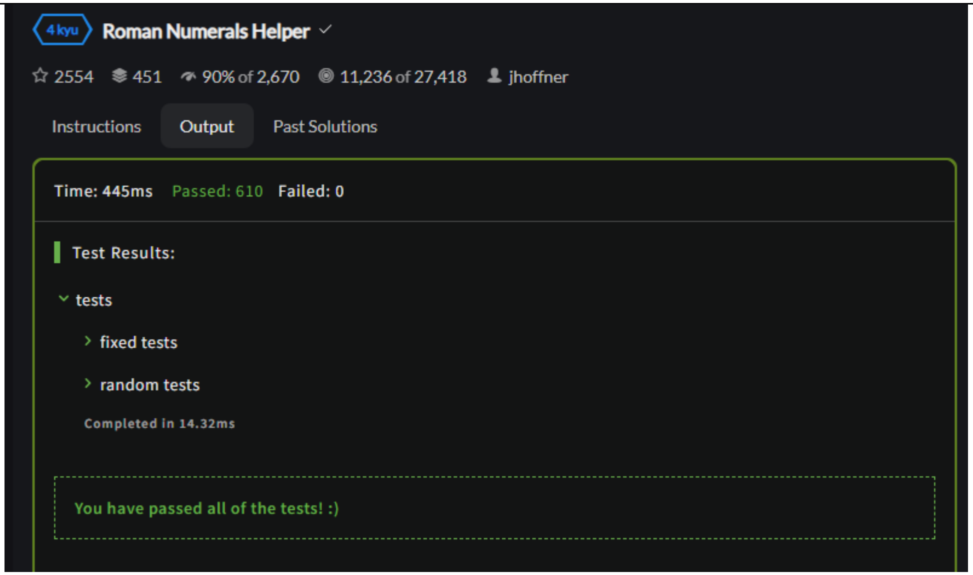
2008 is written as 2000=MM, 8=VIII; or MMVIII

1666 uses each Roman symbol in descending order: MDCLXVI.

Input range : 1 <= n < 4000

In this kata 4 should be represented as IV, NOT as IIII (the "watchmaker's four").

```
def roman_to_int(s):
  roman_numerals = {'I': 1, 'V': 5,
'X': 10, 'L': 50, 'C': 100, 'D': 500, 'M':
1000}
  int_val = 0
  for i in range(len(s)):
    if i > 0 and
roman_numerals[s[i]] >
roman_numerals[s[i - 1]]:
      int_val +=
roman_numerals[s[i]] - 2 *
roman_numerals[s[i - 1]]
    else:
      int_val +=
roman_numerals[s[i]]
  return int_val

def int_to_roman(num):
  val = [
    1000, 900, 500, 400,
    100, 90, 50, 40,
    10, 9, 5, 4,
    1
    ]
  syb = [
    "M", "CM", "D", "CD",
    "C", "XC", "L", "XL",
    "X", "IX", "V", "IV",
    "I"
    ]
  roman_num = ''
  i = 0
  while  num > 0:
    for _ in range(num // val[i]):
      roman_num += syb[i]
      num -= val[i]
    i += 1
```



Time: 445ms Passed: 610 Failed: 0

Test Results:

tests

fixed tests

random tests

Completed in 14.32ms

You have passed all of the tests! :)

```
    return roman_num
```

In this kata we want to convert a string into an integer. The strings simply represent the numbers in words.

Examples:

"one" => 1

"twenty" => 20

"two hundred forty-six" => 246

"seven hundred eighty-three thousand nine hundred and nineteen" => 783919

Additional Notes:

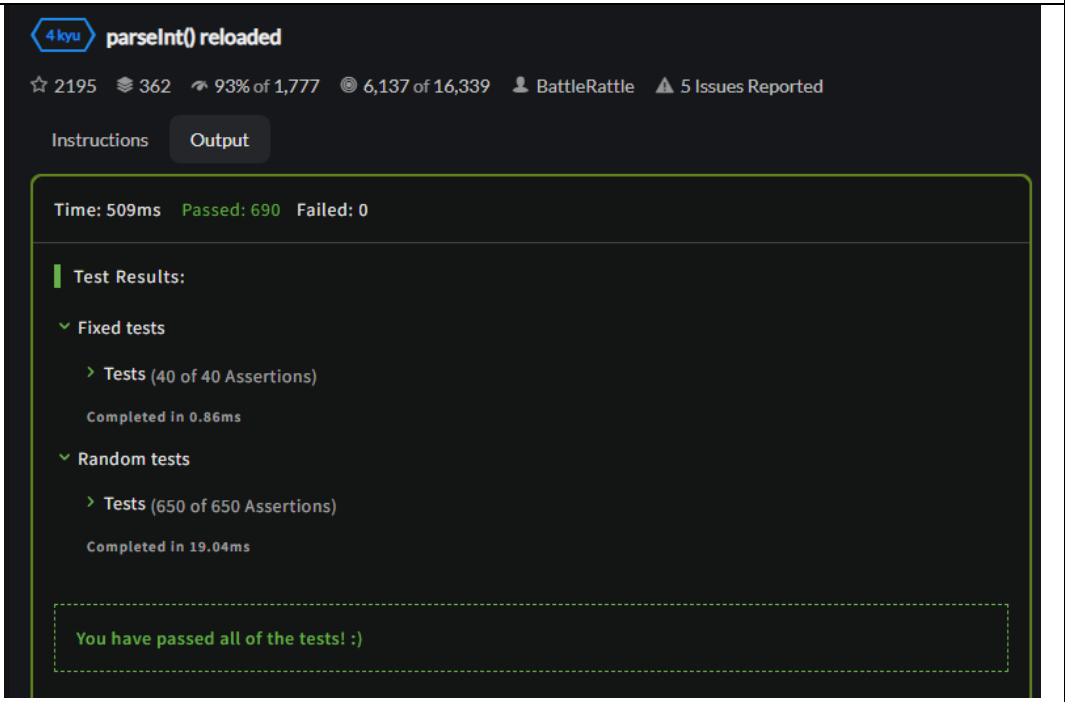
The minimum number is "zero" (inclusively)

The maximum number, which must be supported is 1 million (inclusively)

The "and" in e.g. "one hundred and twenty-four" is optional, in some cases it's present and in others it's not

All tested numbers are valid, you don't need to validate them

```
def word_to_num(s):
    num_dict = {
        "zero": 0, "one": 1, "two": 2, "three": 3, "four": 4, "five": 5,
        "six": 6, "seven": 7, "eight": 8, "nine": 9, "ten": 10,
        "eleven": 11, "twelve": 12, "thirteen": 13, "fourteen": 14,
        "fifteen": 15, "sixteen": 16, "seventeen": 17, "eighteen": 18,
        "nineteen": 19, "twenty": 20, "thirty": 30, "forty": 40,
        "fifty": 50, "sixty": 60, "seventy": 70, "eighty": 80,
        "ninety": 90, "hundred": 100, "thousand": 1000, "million": 1000000
    }
    s = s.replace("-", " ")
    s = s.replace(" and ", " ")
    words = s.split()
    num = 0
    temp = 0
    for word in words:
        if word in num_dict:
            if word == "hundred":
                temp *= num_dict[word]
            elif word == "thousand" or word == "million":
                num += temp * num_dict[word]
                temp = 0
            else:
                temp += num_dict[word]
    num += temp
    return num
```



Time: 509ms Passed: 690 Failed: 0

Test Results:

Fixed tests

Tests (40 of 40 Assertions)

Completed in 0.86ms

Random tests

Tests (650 of 650 Assertions)

Completed in 19.04ms

You have passed all of the tests! :)

Your task in order to complete this Kata is to write a function which formats a duration, given as a number of seconds, in a human-friendly way.

The function must accept a non-negative integer. If it is zero, it just returns "now". Otherwise, the duration is expressed as a combination

of years, days, hours, minutes and seconds.

It is much easier to understand with an example:

* For seconds = 62, your function should return

"1 minute and 2 seconds"

* For seconds = 3662, your function should return

"1 hour, 1 minute and 2 seconds"

For the purpose of this Kata, a year is 365 days and a day is 24 hours.

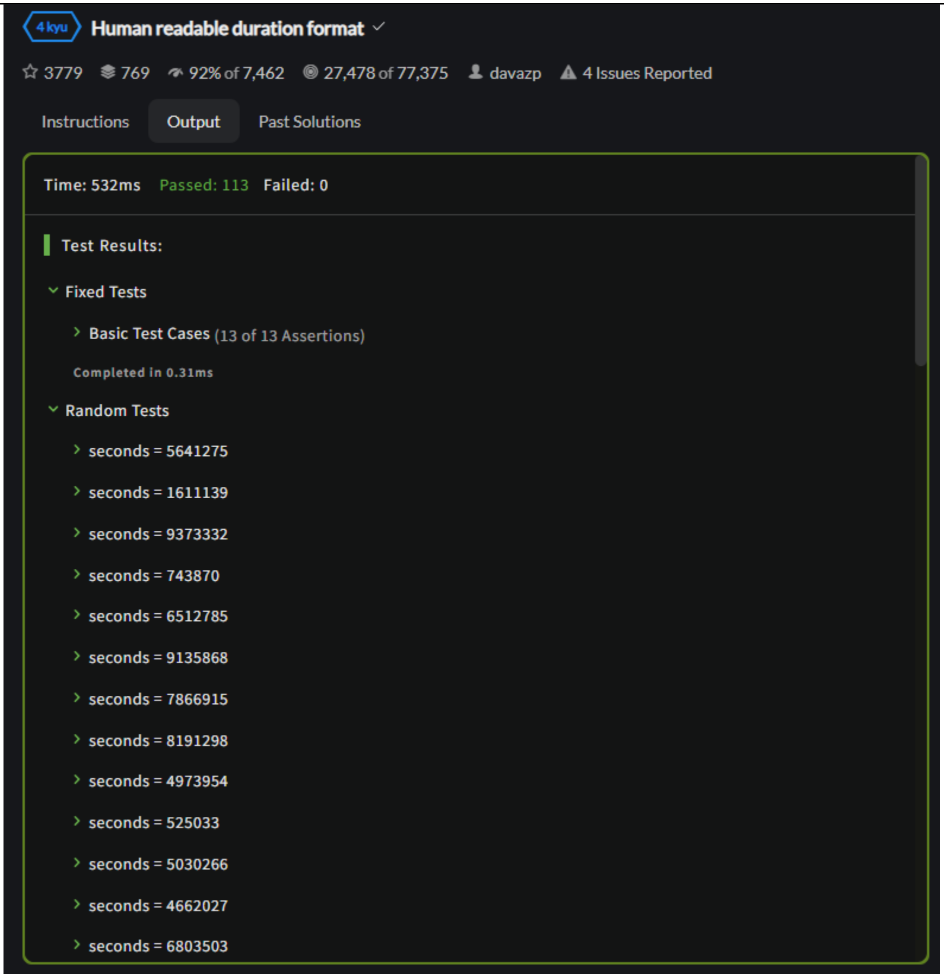
Note that spaces are important.

Detailed rules

The resulting expression is made of components like 4 seconds, 1 year, etc. In general, a positive integer and one of the valid units of time, separated by a space. The unit of time is used in plural if the integer is greater than 1.

The components are separated by a comma and a space (", "). Except the last

```
def format_duration(seconds):
    if seconds == 0:
        return "now"
    components = []
    years, seconds = divmod(seconds, 31536000)
    if years > 0:
        components.append("{} year{}".format(years, "" if years == 1 else "s"))
    days, seconds = divmod(seconds, 86400)
    if days > 0:
        components.append("{} day{}".format(days, "" if days == 1 else "s"))
    hours, seconds = divmod(seconds, 3600)
    if hours > 0:
        components.append("{} hour{}".format(hours, "" if hours == 1 else "s"))
    minutes, seconds = divmod(seconds, 60)
    if minutes > 0:
        components.append("{} minute{}".format(minutes, "" if minutes == 1 else "s"))
    if seconds > 0:
        components.append("{} second{}".format(seconds, "" if seconds == 1 else "s"))
    if len(components) == 1:
        return components[0]
    last_component = components.pop()
```



```
Time: 532ms   Passed: 113   Failed: 0

Test Results:
  ˅ Fixed Tests
      › Basic Test Cases (13 of 13 Assertions)
      Completed in 0.31ms
  ˅ Random Tests
      › seconds = 5641275
      › seconds = 1611139
      › seconds = 9373332
      › seconds = 743870
      › seconds = 6512785
      › seconds = 9135868
      › seconds = 7866915
      › seconds = 8191298
      › seconds = 4973954
      › seconds = 525033
      › seconds = 5030266
      › seconds = 4662027
      › seconds = 6803503
```

component, which is separated by " and ", just like it would be written in English. A more significant units of time will occur before than a least significant one. Therefore, 1 second and 1 year is not correct, but 1 year and 1 second is. Different components have different unit of times. So there is not repeated units like in 5 seconds and 1 second. A component will not appear at all if its value happens to be zero. Hence, 1 minute and 0 seconds is not valid, but it should be just 1 minute. A unit of time must be used "as much as possible". It means that the function should not return 61 seconds, but 1 minute and 1 second instead. Formally, the duration specified by of a component must not be greater than any valid more significant unit of time.	`return ", ".join(components) + " and " + last_component`	