Question1

Create a function that takes three parameters where:

* x is the start of the range (inclusive).
* y is the end of the range (inclusive).
* n is the divisor to be checked against.

Return an ordered list with numbers in the range that are divisible by the third parameter n. Return an empty list if there are no numbers that are divisible by n.

**Examples**

list\_operation(1, 10, 3) ➞ [3, 6, 9]

list\_operation(7, 9, 2) ➞ [8]

list\_operation(15, 20, 7) ➞ []

ANS: def list\_operation(x, y, n):

return [num for num in range(x, y+1) if num % n == 0]

# Test cases

print(list\_operation(1, 10, 3)) # ➞ [3, 6, 9]

print(list\_operation(7, 9, 2)) # ➞ [8]

print(list\_operation(15, 20, 7)) # ➞ []

Question2

Create a function that takes in two lists and returns True if the second list follows the first list by **one** element, and False otherwise. In other words, determine if the second list is the first list shifted to the right by 1.

**Examples**

simon\_says([1, 2], [5, 1]) ➞ True

simon\_says([1, 2], [5, 5]) ➞ False

simon\_says([1, 2, 3, 4, 5], [0, 1, 2, 3, 4]) ➞ True

simon\_says([1, 2, 3, 4, 5], [5, 5, 1, 2, 3]) ➞ False

**Notes**

* Both input lists will be of the same length, and will have a minimum length of 2.
* The values of the 0-indexed element in the second list and the n-1th indexed element in the first list do not matter.

ANS: def simon\_says(list1, list2):

# Compare elements from index 1 to the end of both lists

return list1[1:] == list2[:-1]

# Test cases

print(simon\_says([1, 2], [5, 1])) # ➞ True

print(simon\_says([1, 2], [5, 5])) # ➞ False

print(simon\_says([1, 2, 3, 4, 5], [0, 1, 2, 3, 4])) # ➞ True

print(simon\_says([1, 2, 3, 4, 5], [5, 5, 1, 2, 3])) # ➞ False

Question3

A group of friends have decided to start a secret society. The name will be the first letter of each of their names, sorted in alphabetical order.

Create a function that takes in a list of names and returns the name of the secret society.

### Examples

society\_name(["Adam", "Sarah", "Malcolm"]) ➞ "AMS"

society\_name(["Harry", "Newt", "Luna", "Cho"]) ➞ "CHLN"

society\_name(["Phoebe", "Chandler", "Rachel", "Ross", "Monica", "Joey"])

ANS: def society\_name(names):

return ''.join(sorted(name[0] for name in names))

# Test cases

print(society\_name(["Adam", "Sarah", "Malcolm"])) # ➞ "AMS"

print(society\_name(["Harry", "Newt", "Luna", "Cho"])) # ➞ "CHLN"

print(society\_name(["Phoebe", "Chandler", "Rachel", "Ross", "Monica", "Joey"])) # ➞ "CJMPRR"

Question4

An isogram is a word that has no duplicate letters. Create a function that takes a string and returns either True or False depending on whether or not it's an "isogram".

**Examples**

is\_isogram("Algorism") ➞ True

is\_isogram("PasSword") ➞ False

# Not case sensitive.

is\_isogram("Consecutive") ➞ False

**Notes**

* Ignore letter case (should not be case sensitive).
* All test cases contain valid one word strings.

Question5

Create a function that takes a string and returns True or False, depending on whether the characters are in order or not.

### Examples

is\_in\_order("abc") ➞ True

is\_in\_order("edabit") ➞ False

is\_in\_order("123") ➞ True

is\_in\_order("xyzz") ➞ True

### Notes

You don't have to handle empty strings.

ANS: To check if a string is an isogram (no duplicate letters), you can create the `is\_isogram` function. Here's the implementation:

def is\_isogram(word):

word = word.lower() # Convert the word to lowercase to ignore letter case

return len(set(word)) == len(word)

# Test cases

print(is\_isogram("Algorism")) # ➞ True

print(is\_isogram("PasSword")) # ➞ False

print(is\_isogram("Consecutive")) # ➞ False

The function `is\_isogram` first converts the input word to lowercase using `word.lower()` to make the check case-insensitive. Then, it compares the length of the word after converting it to a set (which removes duplicates) with the original word's length. If they are the same, the function returns `True`, indicating that the word is an isogram. Otherwise, it returns `False`.

Now, for the second part of the question, to check if the characters of a string are in order, you can create the `are\_characters\_in\_order` function as follows:

def are\_characters\_in\_order(s):

return list(s) == sorted(s)

# Test cases

print(are\_characters\_in\_order("abc")) # ➞ True

print(are\_characters\_in\_order("zyx")) # ➞ False

print(are\_characters\_in\_order("hello")) # ➞ False

The function `are\_characters\_in\_order` converts the string `s` into a list and compares it with the sorted version of `s`. If both lists are equal, it means the characters are in order, and the function returns `True`; otherwise, it returns `False`.