**1. Given a sentence, return the number of words which have the same first and last letter.**

**Examples:**  
count\_same\_ends("Pop! goes the balloon") ➞ 1  
count\_same\_ends("And the crowd goes wild!") ➞ 0  
count\_same\_ends("No I am not in a gang.") ➞ 1

In [1]:

**def** count\_same\_ends(in\_string):

special\_chars **=** '!@#$%^&\*.'

cleaned\_string **=** ''

out\_num **=** 0

**for** ele **in** in\_string:

**if** ele **not** **in** special\_chars:

cleaned\_string **+=** ele

**for** ele **in** cleaned\_string**.**split(" "):

**if** ele[0]**.**lower() **==** ele[**-**1]**.**lower():

**if** len(ele) **!=** 1:

out\_num **+=**1

print(f'count\_same\_ends({in\_string}) ➞ {out\_num}')

count\_same\_ends("Pop! goes the balloon")

count\_same\_ends("And the crowd goes wild!")

count\_same\_ends("No I am not in a gang.")

count\_same\_ends(Pop! goes the balloon) ➞ 1

count\_same\_ends(And the crowd goes wild!) ➞ 0

count\_same\_ends(No I am not in a gang.) ➞ 1

**2. The Atbash cipher is an encryption method in which each letter of a word is replaced with its "mirror" letter in the alphabet: A <=> Z; B <=> Y; C <=> X; etc.**

Create a function that takes a string and applies the Atbash cipher to it.

**Examples:**  
atbash("apple") ➞ "zkkov"  
atbash("Hello world!") ➞ "Svool dliow!"  
atbash("Christmas is the 25th of December") ➞ "Xsirhgnzh rh gsv 25gs lu Wvxvnyvi"

In [2]:

**def** atbash(in\_string):

alpha **=** 'abcdefghijklmnopqrstuvwxyz'

r\_alpha **=** 'zyxwvutsrqponmlkjihgfedcba'

out\_string **=** ''

**for** ele **in** in\_string:

**if** ele **not** **in** " !1234567890":

out\_string **+=** r\_alpha[alpha**.**index(ele**.**lower())]**.**upper() **if** ele**.**isupper() **else** r\_alpha[alpha**.**index(ele**.**lower())]

**else**:

out\_string **+=** ele

print(f'atbash({in\_string}) ➞ {out\_string}')

atbash("apple")

atbash("Hello world!")

atbash("Christmas is the 25th of December")

atbash(apple) ➞ zkkov

atbash(Hello world!) ➞ Svool dliow!

atbash(Christmas is the 25th of December) ➞ Xsirhgnzh rh gsv 25gs lu Wvxvnyvi

**3. Create a class Employee that will take a full name as argument, as well as a set of none, one or more keywords. Each instance should have a name and a lastname attributes plus one more attribute for each of the keywords, if any.**

**Examples:**  
john = Employee("John Doe")  
mary = Employee("Mary Major", salary=120000)  
richard = Employee("Richard Roe", salary=110000, height=178)  
giancarlo = Employee("Giancarlo Rossi", salary=115000, height=182, nationality="Italian")

john.name ➞ "John"  
mary.lastname ➞ "Major"  
richard.height ➞ 178  
giancarlo.nationality ➞ "Italian"

In [3]:

**class** Employee:

**def** \_\_init\_\_(self,name**=None**,salary**=None**,height**=None**,nationality**=None**):

self**.**name **=** name

self**.**firstname **=** name**.**split(" ")[0]

self**.**lastname **=** name**.**split(" ")[1]

self**.**salary **=** salary

self**.**height **=** height

self**.**nationality **=** nationality

john **=** Employee("John Doe")

mary **=** Employee("Mary Major",salary**=**120000)

richard **=** Employee("Richard Roe", salary**=**110000, height**=**178)

giancarlo **=** Employee("Giancarlo Rossi", salary**=**115000, height**=**182, nationality**=**"Italian")

print(f'john.name ➞ "{john**.**name}"')

print(f'mary.lastname ➞ "{mary**.**lastname}"')

print(f'richard.height ➞ {richard**.**height}')

print(f'giancarlo.nationality ➞ "{giancarlo**.**nationality}"')

john.name ➞ "John Doe"

mary.lastname ➞ "Major"

richard.height ➞ 178

giancarlo.nationality ➞ "Italian"

**4. Create a function that determines whether each seat can "see" the front-stage. A number can "see" the front-stage if it is strictly greater than the number before it.**

Everyone can see the front-stage in the example below:

# FRONT STAGE [[1, 2, 3, 2, 1, 1], [2, 4, 4, 3, 2, 2], [5, 5, 5, 5, 4, 4], [6, 6, 7, 6, 5, 5]]

# Starting from the left, the 6 > 5 > 2 > 1, so all numbers can see.  
# 6 > 5 > 4 > 2 - so all numbers can see, etc.

Not everyone can see the front-stage in the example below:

# FRONT STAGE [[1, 2, 3, 2, 1, 1], [2, 4, 4, 3, 2, 2], [5, 5, 5, 10, 4, 4], [6, 6, 7, 6, 5, 5]]

# The 10 is directly in front of the 6 and blocking its view.

The function should return True if every number can see the front-stage, and False if even a single number cannot.

**Examples:**  
can\_see\_stage([[1, 2, 3],[4, 5, 6],[7, 8, 9]]) ➞ True  
can\_see\_stage([[0, 0, 0],[1, 1, 1],[2, 2, 2]]) ➞ True  
can\_see\_stage([[2, 0, 0],[1, 1, 1],[2, 2, 2]]) ➞ False  
can\_see\_stage([[1, 0, 0],[1, 1, 1],[2, 2, 2]]) ➞ False

# Number must be strictly smaller than  
# the number directly behind it.

In [4]:

**def** can\_see\_stage(in\_list):

transposed\_list **=** []

**for** ele **in** range(len(in\_list)):

temp\_list **=** []

**for** item **in** range(len(in\_list[ele])):

temp\_list**.**append(in\_list[item][ele])

transposed\_list**.**append(temp\_list)

output **=** **True**

**for** ele **in** transposed\_list:

**if** ele **!=** sorted(ele) **or** len(ele) **!=** len(set(ele)):

output **=** **False**

**break**

print(f'can\_see\_stage({in\_list}) ➞ {output}')

can\_see\_stage([[1, 2, 3],[4, 5, 6],[7, 8, 9]])

can\_see\_stage([[0, 0, 0],[1, 1, 1],[2, 2, 2]])

can\_see\_stage([[2, 0, 0],[1, 1, 1],[2, 2, 2]])

can\_see\_stage([[1, 0, 0],[1, 1, 1],[2, 2, 2]])

can\_see\_stage([[1, 2, 3], [4, 5, 6], [7, 8, 9]]) ➞ True

can\_see\_stage([[0, 0, 0], [1, 1, 1], [2, 2, 2]]) ➞ True

can\_see\_stage([[2, 0, 0], [1, 1, 1], [2, 2, 2]]) ➞ False

can\_see\_stage([[1, 0, 0], [1, 1, 1], [2, 2, 2]]) ➞ False

**5. Create a Pizza class with the attributes order\_number and ingredients (which is given as a list). Only the ingredients will be given as input.**

You should also make it so that its possible to choose a ready made pizza flavour rather than typing out the ingredients manually! As well as creating this Pizza class, hard-code the following pizza flavours.

| **Name** | **Ingredients** |
| --- | --- |
| hawaiian | ham, pineapple |
| meat\_festival | beef, meatball, bacon |
| garden\_feast | spinach, olives, mushroom |

**Examples:**

p1 = Pizza(["bacon", "parmesan", "ham"]) # order 1  
p2 = Pizza.garden\_feast() # order 2  
p1.ingredients ➞ ["bacon", "parmesan", "ham"]  
p2.ingredients ➞ ["spinach", "olives", "mushroom"]  
p1.order\_number ➞ 1  
p2.order\_number ➞ 2

In [5]:

**class** Pizza:

order\_count **=** 0

**def** \_\_init\_\_(self,ingredients**=None**):

self**.**ingredients **=** ingredients

self**.**order\_number **=** Pizza**.**order\_count**+**1

Pizza**.**order\_count **=** self**.**order\_number

**def** hawaiian(self):

self**.**ingredients **=** ['ham', 'pineapple']

**def** meat\_festival(self):

self**.**ingredients **=** ['beef', 'meatball', 'bacon']

**def** garden\_feast(self):

self**.**ingredients **=** ['spinach', 'olives', 'mushroom']

p1 **=** Pizza(["bacon", "parmesan", "ham"])

p2 **=** Pizza()

p2**.**garden\_feast()

print(f'p1.ingredients ➞ {p1**.**ingredients}')

print(f'p2.ingredients ➞ {p2**.**ingredients}')

print(f'p1.order\_number ➞ {p1**.**order\_number}')

print(f'p2.order\_number ➞ {p2**.**order\_number}')

p1.ingredients ➞ ['bacon', 'parmesan', 'ham']

p2.ingredients ➞ ['spinach', 'olives', 'mushroom']

p1.order\_number ➞ 1

p2.order\_number ➞ 2