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

**Ans:**

def count\_same\_ends(string):

string=string.replace("!", "").replace(".", "")

splts=string.split(" ")

count=0

for splt in splts:

if len(splt)>1:

if splt[0].lower()==splt[len(splt)-1].lower():

count+=1

print(count)

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"

**Ans:**

def atbash(string):

dic={'A':1, 'B':2, 'C':3, 'D':4, 'E':5, 'F':6, 'G':7, 'H':8, 'I':9, 'J':10,

'K':11, 'L':12, 'M':13, 'N':14, 'O':15, 'P':16, 'Q':17, 'R':18, 'S':19, 'T':20,

'U':21, 'V':22, 'W':23, 'X':24, 'Y':25, 'Z':26}

exclude\_chars=[' ', '!', '.', '0', '1', '2', '3', '4', '5', '6', '7', '8', '9']

out\_str=""

for c in string:

if c not in exclude\_chars:

if c.isalpha():

val=dic[c.upper()]

key\_list=list(dic.keys())

values\_list=list(dic.values())

val=26-val+1

position=values\_list.index(val)

if c.islower():

out\_str+=key\_list[position].lower()

else:

out\_str+=key\_list[position]

else:

out\_str+=key\_list[position]

else:

out\_str+=c

print(out\_str)

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"

**Ans:**

class Employee():

def \_\_init\_\_(self, FullName, \*\*kwargs):

splt=FullName.split(" ")

self.name=splt[0]

self.lastname=splt[1]

for key, value in kwargs.items():

self.\_\_setattr\_\_(key,value)

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.

**Ans:**

def can\_see\_stage(lst):

rows=len(lst)

cols=len(lst[0])

result=True

for i in range(rows-1):

for j in range(cols):

if lst[i][j]<lst[i+1][j]:

continue

else:

result=False

break

print(result)

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

**Ans:**

dic={"order\_number":0}

class Pizza():

def \_\_init\_\_(self, ingredients):

dic["order\_number"]+=1

self.order\_number=dic["order\_number"]

self.ingredients=ingredients

def hawaiian():

return Pizza(['ham', 'pineapple'])

def meat\_festival():

return Pizza(['beef', 'meatball', 'bacon'])

def garden\_feast():

return Pizza(['spinach', 'olives', 'mushroom'])