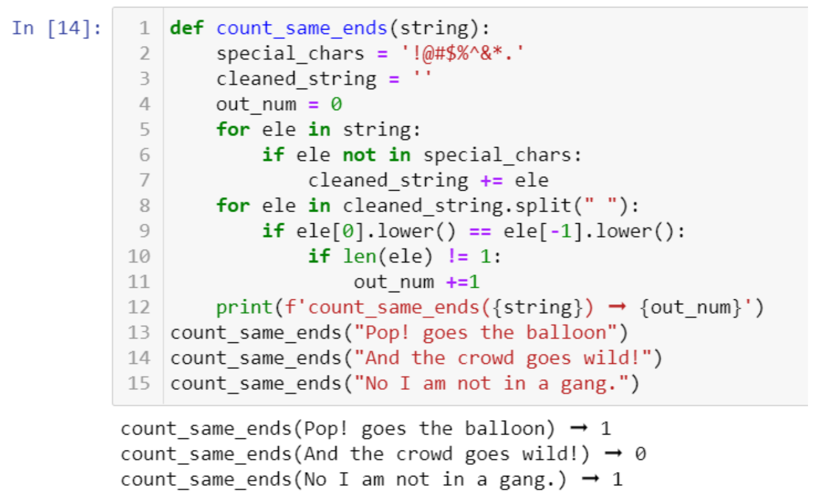
1. Given a sentence, return the number of words which have the same first and last letter.

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:



```
In [14]:
def count_same_ends(string):
    special_chars = '!@#$%^&*.'
    cleaned_string = ''
    out_num = 0
    for ele 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({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.")
```

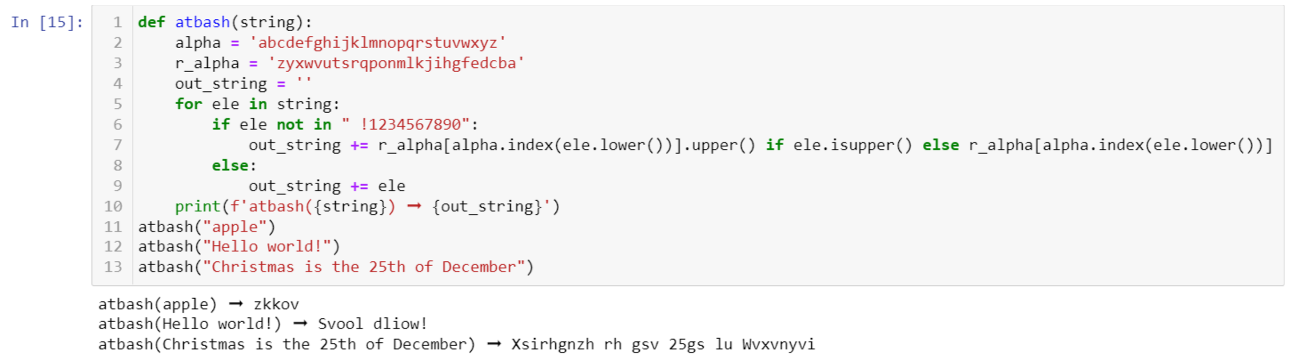
```
count_same_ends(Pop! goes the balloon) ➞ 1
count_same_ends(And the crowd goes wild!) ➞ 0
count_same_ends(No I am not in a gang.) ➞ 1
```

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"

Ans:



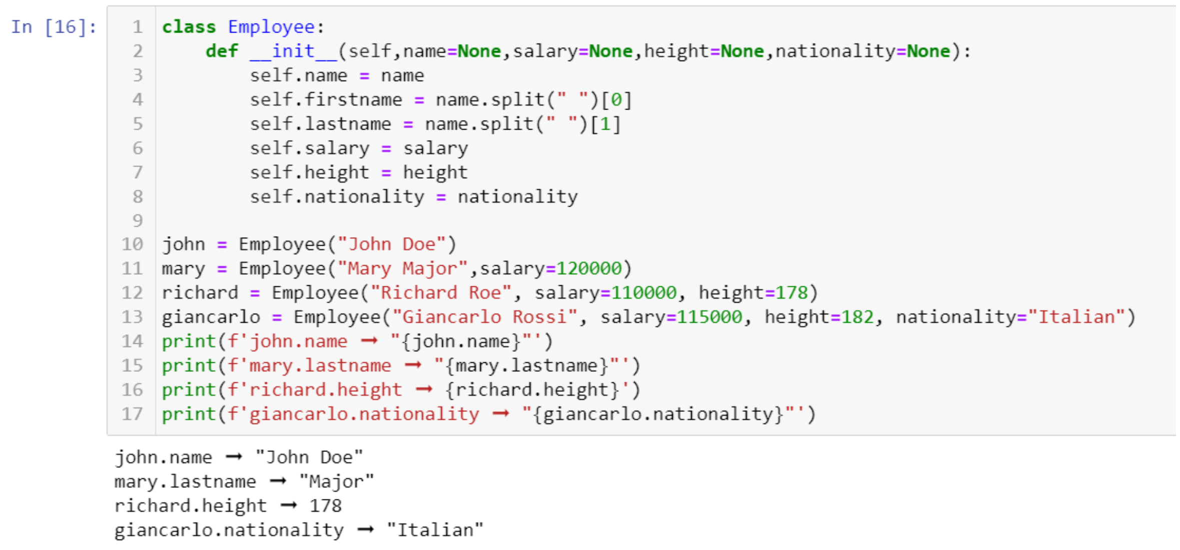
```
In [15]:
def atbash(string):
    alpha = 'abcdefghijklmnopqrstuvwxyz'
    r_alpha = 'zyxwvutsrqponmlkjihgfedcba'
    out_string = ''
    for ele in string:
        if ele not in " !1234567890":
            out_string += r_alpha[alpha.index(ele.lower())].upper() if ele.isupper() else r_alpha[alpha.index(ele.lower())]
        else:
            out_string += ele
    print(f'atbash({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"
Ans:



In [16]:
```python
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]]

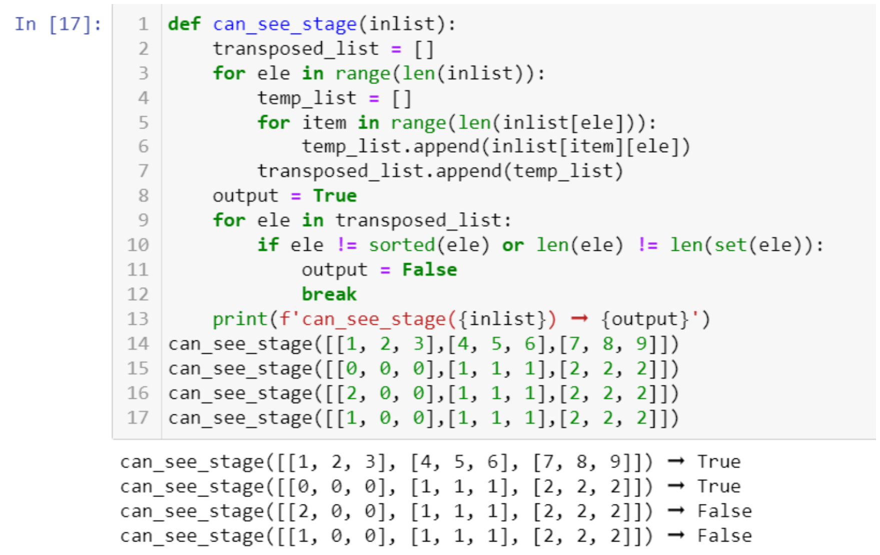
# The 10 is directly in front of the 6 and blocking its view.

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.

Ans:



```
In [17]:
def can_see_stage(inlist):
    transposed_list = []
    for ele in range(len(inlist)):
        temp_list = []
        for item in range(len(inlist[ele])):
            temp_list.append(inlist[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({inlist}) ➞ {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.

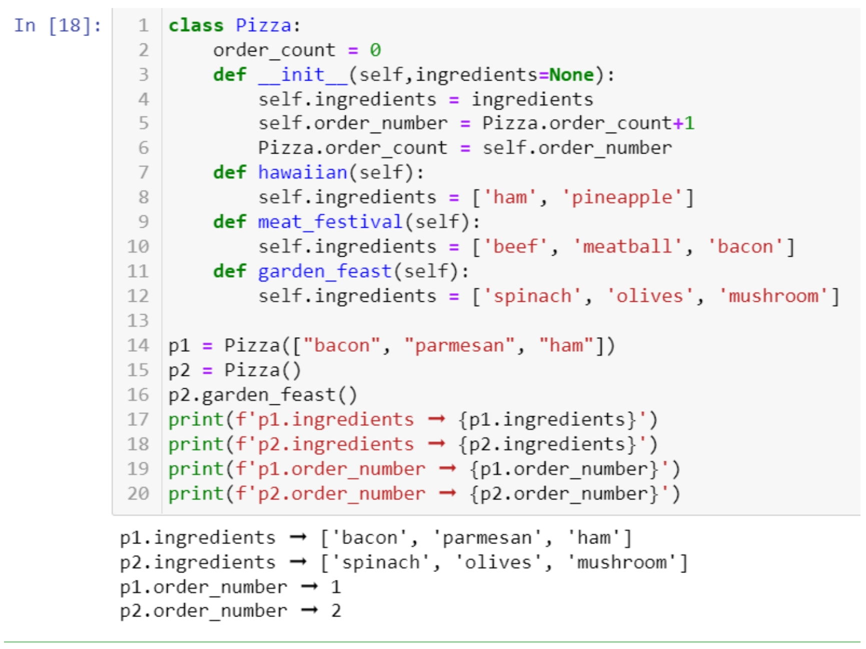
You should also make it so that its possible to choose a ready made pizza flavour rather than typing out the ingredients manually! As well as creating this Pizza class, hard-code the following pizza flavours.

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

Ans:



In [18]:

```python
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
```