1. Create a checker board generator, which takes as inputs n and 2 elements to generate an n x n checkerboard with those two elements as alternating squares.

Examples:

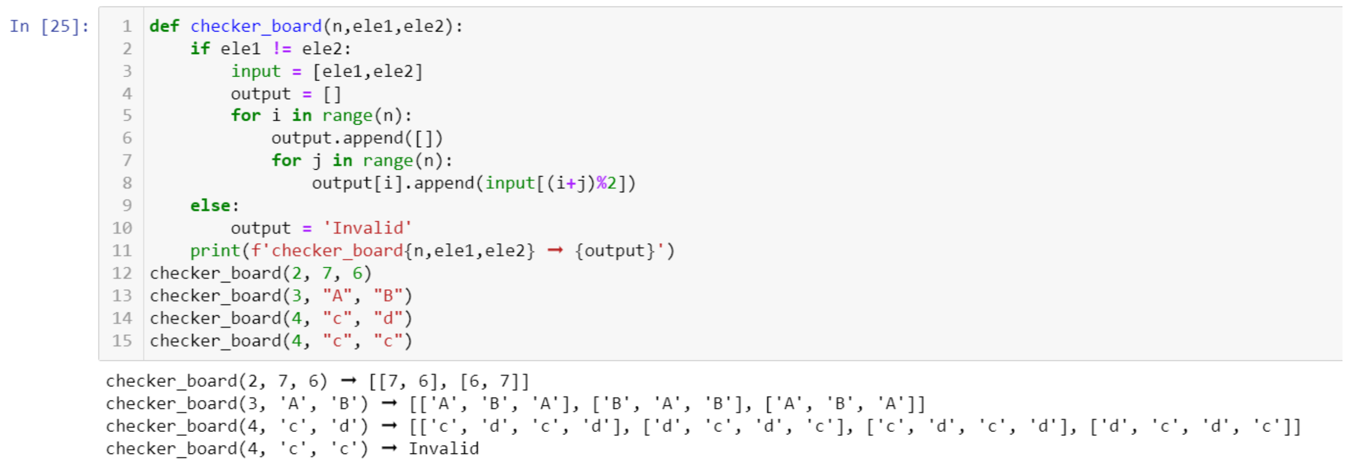
checker_board(2, 7, 6) ➞ [
  [7, 6],
  [6, 7]
]

checker_board(3, "A", "B") ➞ [
  ["A", "B", "A"],
  ["B", "A", "B"],
  ["A", "B", "A"]
]

checker_board(4, "c", "d") ➞ [
  ["c", "d", "c", "d"],
  ["d", "c", "d", "c"],
  ["c", "d", "c", "d"],
  ["d", "c", "d", "c"]
]

checker_board(4, "c", "c") ➞ "invalid"

**Ans**:



```
In [25]:
def checker_board(n,ele1,ele2):
    if ele1 != ele2:
        input = [ele1,ele2]
        output = []
        for i in range(n):
            output.append([])
            for j in range(n):
                output[i].append(input[(i+j)%2])
    else:
        output = 'Invalid'
    print(f'checker_board{n,ele1,ele2} ➞ {output}')
checker_board(2, 7, 6)
checker_board(3, "A", "B")
checker_board(4, "c", "d")
checker_board(4, "c", "c")
```

```
checker_board(2, 7, 6) ➞ [[7, 6], [6, 7]]
checker_board(3, 'A', 'B') ➞ [['A', 'B', 'A'], ['B', 'A', 'B'], ['A', 'B', 'A']]
checker_board(4, 'c', 'd') ➞ [['c', 'd', 'c', 'd'], ['d', 'c', 'd', 'c'], ['c', 'd', 'c', 'd'], ['d', 'c', 'd', 'c']]
checker_board(4, 'c', 'c') ➞ Invalid
```

2. A string is an almost-palindrome if, by changing only one character, you can make it a palindrome. Create a function that returns True if a string is an almost-palindrome and False otherwise.

Examples:

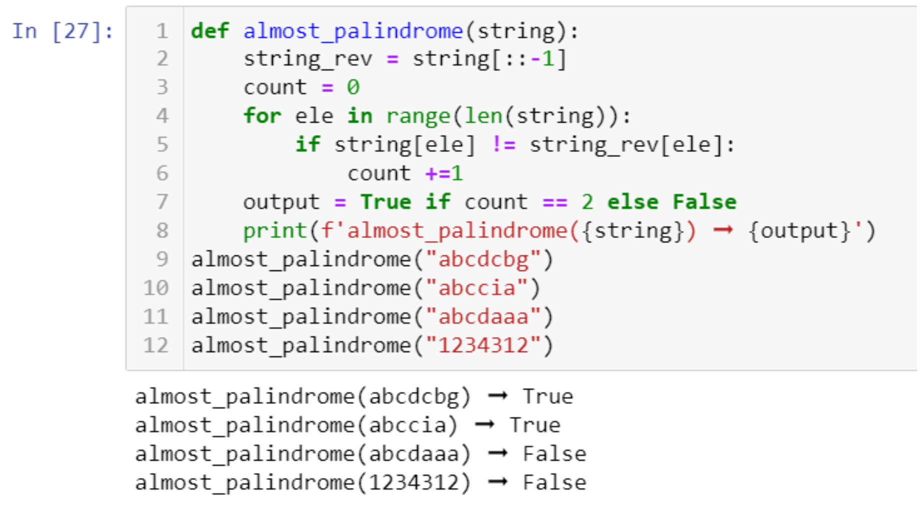
almost_palindrome("abcdcbg") ➞ True
# Transformed to "abcdcba" by changing "g" to "a".

almost_palindrome("abccia") ➞ True
# Transformed to "abccba" by changing "i" to "b".

almost_palindrome("abcdaaa") ➞ False
# Can't be transformed to a palindrome in exactly 1 turn.

almost_palindrome("1234312") ➞ False

**Ans**:



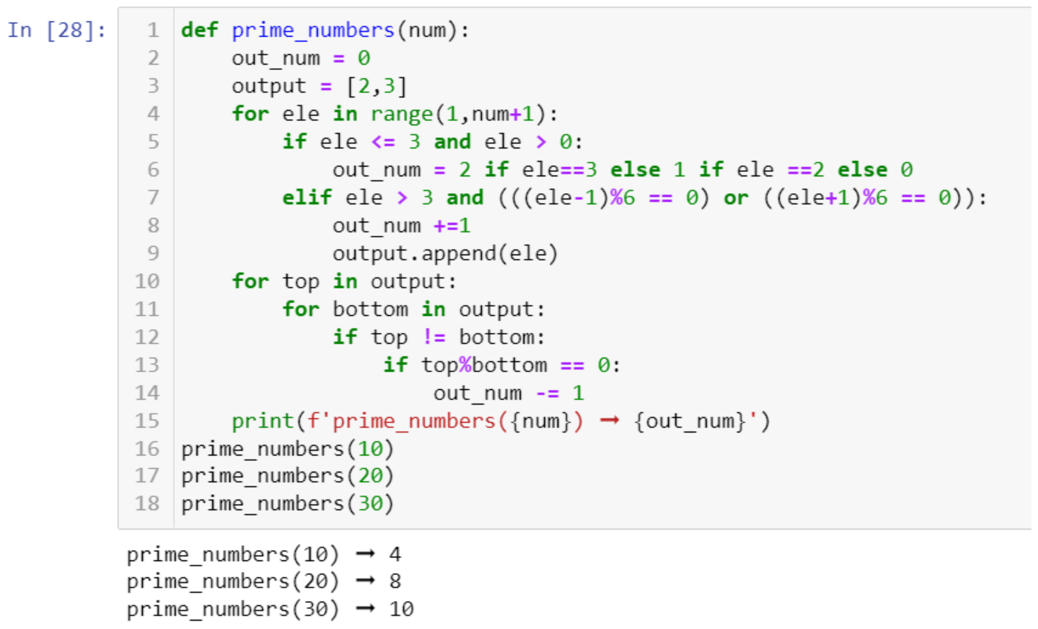
```
In [27]:
def almost_palindrome(string):
    string_rev = string[::-1]
    count = 0
    for ele in range(len(string)):
        if string[ele] != string_rev[ele]:
            count +=1
    output = True if count == 2 else False
    print(f'almost_palindrome({string}) ➞ {output}')
almost_palindrome("abcdcbg")
almost_palindrome("abccia")
almost_palindrome("abcdaaa")
almost_palindrome("1234312")
```

```
almost_palindrome(abcdcbg) ➞ True
almost_palindrome(abccia) ➞ True
almost_palindrome(abcdaaa) ➞ False
almost_palindrome(1234312) ➞ False
```

3. Create a function that finds how many prime numbers there are, up to the given integer.

Examples:
prime_numbers(10) ➞ 4 # 2, 3, 5 and 7
prime_numbers(20) ➞ 8 # 2, 3, 5, 7, 11, 13, 17 and 19
prime_numbers(30) ➞ 10 # 2, 3, 5, 7, 11, 13, 17, 19, 23 and 29

**Ans**:



```
In [28]:
def prime_numbers(num):
    out_num = 0
    output = [2,3]
    for ele in range(1,num+1):
        if ele <= 3 and ele > 0:
            out_num = 2 if ele==3 else 1 if ele ==2 else 0
        elif ele > 3 and (((ele-1)%6 == 0) or ((ele+1)%6 == 0)):
            out_num +=1
            output.append(ele)
    for top in output:
        for bottom in output:
            if top != bottom:
                if top%bottom == 0:
                    out_num -= 1
    print(f'prime_numbers({num}) ➞ {out_num}')
prime_numbers(10)
prime_numbers(20)
prime_numbers(30)
```

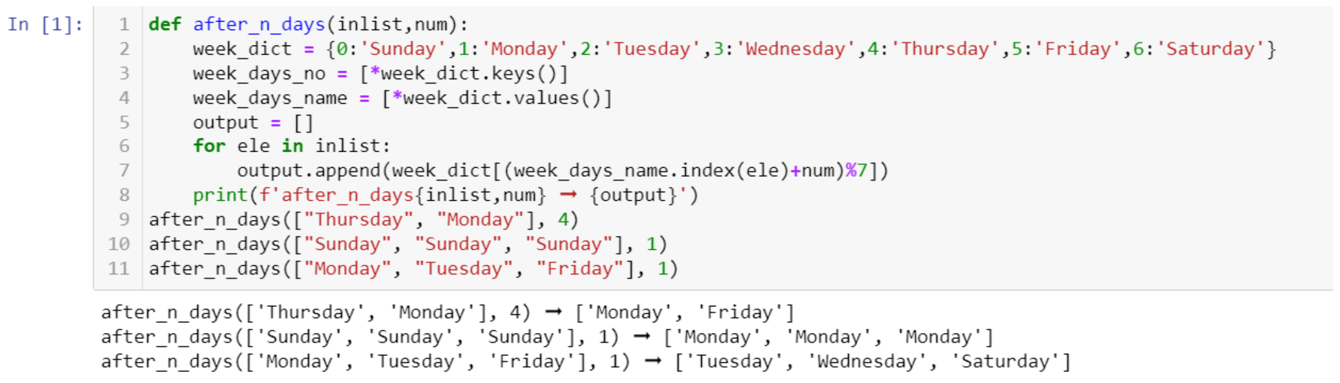
```
prime_numbers(10) ➞ 4
prime_numbers(20) ➞ 8
prime_numbers(30) ➞ 10
```

4. If today was Monday, in two days, it would be Wednesday.

Create a function that takes in a list of days as input and the number of days to increment by. Return a list of days after n number of days has passed.

Examples:
after_n_days(["Thursday", "Monday"], 4) ➞ ["Monday", "Friday"]
after_n_days(["Sunday", "Sunday", "Sunday"], 1) ➞ ["Monday", "Monday", "Monday"]
after_n_days(["Monday", "Tuesday", "Friday"], 1) ➞ ["Tuesday", "Wednesday", "Saturday"]

**Ans**:



```
In [1]:
def after_n_days(inlist,num):
    week_dict = {0:'Sunday',1:'Monday',2:'Tuesday',3:'Wednesday',4:'Thursday',5:'Friday',6:'Saturday'}
    week_days_no = [*week_dict.keys()]
    week_days_name = [*week_dict.values()]
    output = []
    for ele in inlist:
        output.append(week_dict[(week_days_name.index(ele)+num)%7])
    print(f'after_n_days{inlist,num} ➞ {output}')
after_n_days(["Thursday", "Monday"], 4)
after_n_days(["Sunday", "Sunday", "Sunday"], 1)
after_n_days(["Monday", "Tuesday", "Friday"], 1)
```

```
after_n_days(['Thursday', 'Monday'], 4) ➞ ['Monday', 'Friday']
after_n_days(['Sunday', 'Sunday', 'Sunday'], 1) ➞ ['Monday', 'Monday', 'Monday']
after_n_days(['Monday', 'Tuesday', 'Friday'], 1) ➞ ['Tuesday', 'Wednesday', 'Saturday']
```

5. You are in the process of creating a chat application and want to add an anonymous name feature. This anonymous name feature will create an alias that consists of two capitalized words beginning with the same letter as the users first name.

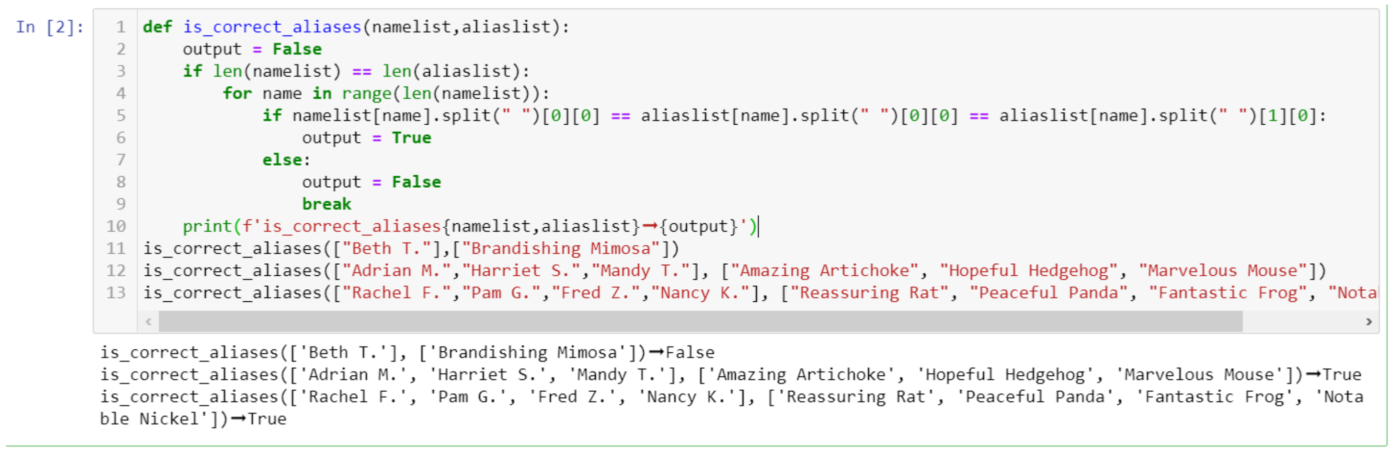
Create a function that determines if the list of users is mapped to a list of anonymous names correctly.

Examples:
is_correct_aliases(["Adrian M.", "Harriet S.", "Mandy T."], ["Amazing Artichoke", "Hopeful Hedgehog", "Marvelous Mouse"]) ➞ True

is_correct_aliases(["Rachel F.", "Pam G.", "Fred Z.", "Nancy K."], ["Reassuring Rat", "Peaceful Panda", "Fantastic Frog", "Notable Nickel"]) ➞ True

is_correct_aliases(["Beth T."], ["Brandishing Mimosa"]) ➞ False
# Both words in "Brandishing Mimosa" should begin with a "B" - "Brandishing Beaver" would do the trick.

**Ans**:



In [2]:

```python
def is_correct_aliases(namelist,aliaslist):
    output = False
    if len(namelist) == len(aliaslist):
        for name in range(len(namelist)):
            if namelist[name].split(" ")[0][0] == aliaslist[name].split(" ")[0][0] == aliaslist[name].split(" ")[1][0]:
                output = True
            else:
                output = False
                break
    print(f'is_correct_aliases{namelist,aliaslist}➞{output}')
is_correct_aliases(["Beth T."],["Brandishing Mimosa"])
is_correct_aliases(["Adrian M.","Harriet S.","Mandy T."], ["Amazing Artichoke", "Hopeful Hedgehog", "Marvelous Mouse"])
is_correct_aliases(["Rachel F.","Pam G.","Fred Z.","Nancy K."], ["Reassuring Rat", "Peaceful Panda", "Fantastic Frog", "Nota
```

```
is_correct_aliases(['Beth T.'], ['Brandishing Mimosa'])➞False
is_correct_aliases(['Adrian M.', 'Harriet S.', 'Mandy T.'], ['Amazing Artichoke', 'Hopeful Hedgehog', 'Marvelous Mouse'])➞True
is_correct_aliases(['Rachel F.', 'Pam G.', 'Fred Z.', 'Nancy K.'], ['Reassuring Rat', 'Peaceful Panda', 'Fantastic Frog', 'Nota
ble Nickel'])➞True
```