# Assignment 19

#### 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"

In [1]:

**def** checker\_board(grid\_size,in\_one,in\_two):  
 **if** in\_one **!=** in\_two:  
 input **=** [in\_one,in\_two]  
 output **=** []  
 **for** ele\_1 **in** range(grid\_size):  
 output**.**append([])  
 **for** ele\_2 **in** range(grid\_size):  
 output[ele\_1]**.**append(input[(ele\_1**+**ele\_2)**%2**])  
 **else**:  
 output **=** 'Invalid'  
 print(f'checker\_board{grid\_size,in\_one,in\_two} ➞ {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

In [2]:

**def** almost\_palindrome(in\_string):  
 in\_string\_rev **=** in\_string[::**-**1]  
 count **=** 0  
 **for** ele **in** range(len(in\_string)):  
 **if** in\_string[ele] **!=** in\_string\_rev[ele]:  
 count **+=**1  
 output **=** **True** **if** count **==** 2 **else** **False**  
 print(f'almost\_palindrome({in\_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**

In [3]:

**def** prime\_numbers(in\_num):  
 out\_num **=** 0  
 out\_list **=** [2,3]  
 **for** ele **in** range(1,in\_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  
 out\_list**.**append(ele)   
 **for** top **in** out\_list:  
 **for** bottom **in** out\_list:  
 **if** top **!=** bottom:  
 **if** top**%bottom** == 0:  
 out\_num **-=** 1  
 print(f'prime\_numbers({in\_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"]**

In [4]:

**def** after\_n\_days(in\_list,in\_num):  
 week\_dict **=** {0:'Sunday',1:'Monday',2:'Tuesday',3:'Wednesday',4:'Thursday',5:'Friday',6:'Saturday'}  
 week\_days\_no **=** list(week\_dict**.**keys())  
 week\_days\_name **=** list(week\_dict**.**values())  
 output **=** []  
 **for** ele **in** in\_list:  
 output**.**append(week\_dict[(week\_days\_name**.**index(ele)**+**in\_num)**%7**])  
 print(f'after\_n\_days{in\_list,in\_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.

In [5]:

**def** is\_correct\_aliases(in\_list\_one, in\_list\_two):  
 output **=** **False**  
 **if** len(in\_list\_one) **==** len(in\_list\_two):  
 **for** ele **in** range(len(in\_list\_one)):  
 **if** in\_list\_one[ele]**.**split(" ")[0][0] **==** in\_list\_two[ele]**.**split(" ")[0][0] **==** in\_list\_two[ele]**.**split(" ")[1][0]:  
 output **=** **True**  
 **else**:  
 output **=** **False**  
 **break**  
 print(f'is\_correct\_aliases{in\_list\_one,in\_list\_two}➞{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", "Notable Nickel"])

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', 'Notable Nickel'])➞True