In [1]:
```
#Q)1). Write a function that takes a list of lists and returns the value of all of the
#symbols in it, where each symbol adds or takes something from the total
#score. Symbol values:
# = 5, O = 3, X = 1, ! = -1, !! = -3, !!! = -5
```

In [2]:
```
# dictionary containing symbols and their respective value as per question
check_dict = {
    '#' : 5,
    'O' :3,
    'X' : 1,
    '!' : -1,
    '!!' : -3,
    '!!!' : -5
}
```

In [3]:
```
#Function to implement the given question
def check_score(list_of_lists):
    sum =0
    try:
        for l in list_of_lists:
            for char in l:
                sum= sum+ check_dict[char]

        print(sum)
    except Exception as e:
        print(e)
```

In [5]:
```
#Example1:
row, coloumn = map(int, input("Enter the row and coloumn\n").split())
list_of_lists = []
for x in range(row):
    m = []
    for y in range(coloumn):
        characther = input("Enter the character ['#','O','X','!','!!','!!!']")
        m.append(characther)

    list_of_lists.append(m)
```

```
Enter the row and coloumn
2 2
Enter the character ['#','O','X','!','!!','!!!']#
Enter the character ['#','O','X','!','!!','!!!']O
Enter the character ['#','O','X','!','!!','!!!']X
Enter the character ['#','O','X','!','!!','!!!']X
```

In [6]:
```
list_of_lists
```

Out[6]:
```
[['#', '!'], ['!!', 'X']]
```

In [7]:
```
check_score(list_of_lists)
```

```
2
```

In [8]:
```
#Example 2:
row, coloumn = map(int, input("Enter the row and coloumn\n").split())
list_of_lists = []
for x in range(row):
    m = []
    for y in range(coloumn):
        characther = input("Enter the character ['#','O','X','!','!!','!!!']")
        m.append(characther)

    list_of_lists.append(m)
```

```
Enter the row and coloumn
3 3
Enter the character ['#','O','X','!','!!','!!!']!!!
Enter the character ['#','O','X','!','!!','!!!']!!!
Enter the character ['#','O','X','!','!!','!!!']!
Enter the character ['#','O','X','!','!!','!!!']X
Enter the character ['#','O','X','!','!!','!!!']#
Enter the character ['#','O','X','!','!!','!!!']!!!
Enter the character ['#','O','X','!','!!','!!!']!!!
Enter the character ['#','O','X','!','!!','!!!']X
Enter the character ['#','O','X','!','!!','!!!']O
```

In [9]:
```
list_of_lists
```

Out[9]:
```
[['!!!', 'O', '!'], ['X', '#', '!!!'], ['!!!', 'X', 'O']]
```

In [10]:
```
check_score(list_of_lists)
```

```
-1
```

In [ ]:
```

```

In [11]:
```
#Q)2. Create a function that takes a variable number of arguments, each
#argument representing the number of items in a group, and returns the
#number of permutations (combinations) of items that you could get by taking
#one item from each group.

from itertools import permutations #python module to perform permutations
def num_of_permutations(variable_number_arguments):
    total_perm = 1 #intilize the total permutations by 1
    for item in  variable_number_arguments:
        perm = len(permutations(range(item),1)) # function to implement using permutations class
        total = 0
        for i in perm:
            total = total+1                         #calculates the total number of permutations in a single argu
        total_perm = total_perm * total             #gives the total number of permutations


    print(f'The total number of permutations are {total_perm}')
```

In [13]:
```
#Example1: (2,3)
variable_number_arguments =  map(int, input("Enter the variable number of arguments\n").split())
num_of_permutations(variable_number_arguments)
```

```
Enter the variable number of arguments
2 3
The total number of permutations are 6
```

In [14]:
```
#Example2: (3,7,4):
variable_number_arguments =  map(int, input("Enter the variable number of arguments\n").split())
num_of_permutations(variable_number_arguments)
```

```
Enter the variable number of arguments
3 7 4
The total number of permutations are 84
```

In [15]:
```
#Example3:(2,3,4,5):
variable_number_arguments =  map(int, input("Enter the variable number of arguments\n").split())
num_of_permutations(variable_number_arguments)
```

```
Enter the variable number of arguments
2 3 4 5
The total number of permutations are 120
```

In [ ]:
```

```

In [16]:
```
#Q3) Create a function that takes a string as an argument and returns the Morse code equvialent.
```

In [17]:
```
char_to_dots = { 'A':'.-', 'B':'-...',
                 'C':'-.-.', 'D':'-..', 'E':'.',
                 'F':'..-.', 'G':'--.', 'H':'....',
                 'I':'..', 'J':'.---', 'K':'-.-',
                 'L':'.-..', 'M':'--', 'N':'-.',
                 'O':'---', 'P':'.--.', 'Q':'--.-',
                 'R':'.-.', 'S':'...', 'T':'-',
                 'U':'..-', 'V':'...-', 'W':'.--',
                 'X':'-..-', 'Y':'-.--', 'Z':'--..',
                 '1':'.----', '2':'..---', '3':'...--',
                 '4':'....-', '5':'.....', '6':'-....',
                 '7':'--...', '8':'---..', '9':'----.',
                 '0':'-----', ',':'--..--', '.':'.-.-.-',
                 '?':'..--..', '/':'-..-.', '-':'-....-',
                 '(':'-.--.', ')':'-.--.-', ' ':' ' }
```

In [18]:
```
def Morse_code(string):
    try:
        for char in string:
            #print(char)
            print(char_to_dots[char], end=' ')

    except Exception as e:
        print(e)
```

In [19]:
```
#Example1 ("HELP ME!")
string = input("Enter the string to convert into Morse code\n")
Morse_code(string)
```

```
Enter the string to convert into Morse code
HELP ME!
.... . .-.. .--. -- . --..--
```

In [20]:
```
#Example 2( "ECHABIT CHALLENGE")
string = input("Enter the string to convert into Morse code\n")
Morse_code(string)
```

```
Enter the string to convert into Morse code
ECHABIT CHALLENGE
. -.-. .... .- -... .. -   -.-. .... .- .-.. .-.. . -. --. .
```

In [ ]:
```

```

In [21]:
```
#Q4)Write a function that takes a number and returns True if it is a prime; False
#otherwise. The number can be 2^64-1 (2 to the power of 63, not XOR) . With
#the standard technique it would be O(2^64-1), which is much too large for the
#10 second time limit.
```

In [23]:
```
#Here the logic is any composite number is divisble more than twice by the numbers between [1-9].
#and prime number is divisible less than or equal twice by the numbers between [1-9]..so the range is (1,10)
def prime_number(number):
    try:
        k=0
        for i in range(1,10):
            if(number%i==0):
                k+=1
                if(k>2):
                    print(False)
                    break
        if(k==1 or k==2):
            print(True)



    except Exception as e:
        print(e)
```

In [43]:
```
#Example1: 7
number = int(input("Enter a number\n"))
prime_number(number)
```

```
Enter a number
7
True
```

In [44]:
```
#Example2: 56963
number = int(input("Enter a number\n"))
prime_number(number)
```

```
Enter a number
56963
True
```

In [45]:
```
#Example3: 515151251552 4
number = int(input("Enter a number\n"))
prime_number(number)
```

```
Enter a number
515151251552 4
False
```

In [46]:
```
#Example4:999999000001
number = int(input("Enter a number "))
prime_number(number)
```

```
Enter a number 999999000001
True
```

In [ ]:
```

```

In [48]:
```
#Q)5. Create a function that converts a word to a bitstring and then to a boolean
#list based on the following criteria:
#1. Locate the position of the letter in the English alphabet (from 1 to 26).
#2. Odd positions will be represented as 1 and 0 otherwise.
#3. Convert the represented positions to boolean values, 1 for True and 0
#for False.
#4. Store the conversions into an array.
```

In [49]:
```
alphabet = {
        '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'
    }
```

In [78]:
```
def word_to_bitstring(word):
    try:
        list1 = []
        for char in word:
            if(int(alphabet[char])%2==0):
                list1.append('0')


            else:
                list1.append('1')
        print(''.join(list1))
        return ''.join(list1) #Making list into a string by using joint function
    except Exception as e:
        print(e)

def bitstrings_to_boolean(bitstring):
    try:
        list2 = []
        for bit in bitstring:
            if bit=='0':
                list2.append(False)
            else:
                list2.append(True)

        print(list2)
    except Exception as e:
        print(e)
```

In [79]:
```
#Example1 : deep
word = input("Enter a word to convert into boolean table\n")
bitstring = word_to_bitstring(word)
bitstrings_to_boolean(bitstring)
```

```
Enter a word to convert into boolean table
deep
0110
[False, True, True, False]
```

In [80]:
```
#Example2 : loves
word = input("Enter a word to convert into boolean table\n")
bitstring = word_to_bitstring(word)
bitstrings_to_boolean(bitstring)
```

```
Enter a word to convert into boolean table
loves
01011
[False, True, False, True, True]
```

In [81]:
```
#Example3 : teah
word = input("Enter a word to convert into boolean table\n")
bitstring = word_to_bitstring(word)
bitstrings_to_boolean(bitstring)
```

```
Enter a word to convert into boolean table
teah
0110
[False, True, True, False]
```

In [ ]:
```

```