**Python Advance Programming Assignment-01**

**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

A list of lists containing 2 \*s, a O, and a !!! would equal (0 + 5 + 5 + 3 - 5) 8.

If the final score is negative, return 0 (e.g. 3 \*s, 3 !!s, 2 !!!s and a X would be (0 + 5 + 5 + 5 - 3 - 3 - 3 - 5 - 5 + 1) -3, so return 0.

Examples

check\_score([ ["\*", "!"], ["!!", "X"] ]) ➞ 2

check\_score([ ["!!!", "O", "!"], ["X", "\*", "!!!"], ["!!", "X", "O"] ]) ➞ 0

**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.**

Examples

combinations(2, 3) ➞ 6

combinations(3, 7, 4) ➞ 84

combinations(2, 3, 4, 5) ➞ 120

In [47]:

**def** combinations(**\***args):

mul**=**1

**for** i **in** args:

mul**=**mul**\***i

**return** mul

print(combinations(2, 3))

print(combinations(3, 7, 4))

print(combinations(2, 3, 4, 5))

6

84

120

**3. Create a function that takes a string as an argument and returns the Morse code equivalent.**

Examples

encode\_morse("EDABBIT CHALLENGE") ➞ ". -.. .- -... -... .. - -.-. .... .- .-.. .-.. . -. --. ."

encode\_morse("HELP ME !") ➞ ".... . .-.. .--. -- . -.-.--"

This dictionary can be used for coding:

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': '--..', ' ': ' ', '0': '-----', '1': '.----', '2': '..---', '3': '...--', '4': '....-', '5': '.....', '6': '-....', '7': '--...', '8': '---..', '9': '----.', '&': '.-...', "'": '.----.', '@': '.--.-.', ')': '-.--.-', '(': '-.--.', ':': '---...', ',': '--..--', '=': '-...-', '!': '-.-.--', '.': '.-.-.-', '-': '-....-', '+': '.-.-.', '"': '.-..-.', '?': '..--..', '/': '-..-.'

}

In [44]:

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': '--..', ' ': ' ', '0': '-----',

'1': '.----', '2': '..---', '3': '...--', '4': '....-', '5': '.....', '6': '-....',

'7': '--...', '8': '---..', '9': '----.', '&': '.-...', "'": '.----.', '@': '.--.-.',

')': '-.--.-', '(': '-.--.', ':': '---...', ',': '--..--', '=': '-...-', '!': '-.-.--',

'.': '.-.-.-', '-': '-....-', '+': '.-.-.', '"': '.-..-.', '?': '..--..', '/': '-..-.'

}

**def** encode\_morse(string):

l**=**{}

**for** i **in** string:

l[i]**=**char\_to\_dots[i]

**return** ''**.**join(l**.**values())

In [46]:

encode\_morse("EDABBIT CHALLENGE")

Out[46]:

'.-...--.....- -.-......-..-.--.'

In [45]:

encode\_morse('HELP ME !')

Out[45]:

'......-...--. ---.-.--'

**4.Write a function that takes a number and returns True if it's 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.**

Examples

prime(7) ➞ True

prime(56963) ➞ True

prime(5151512515524) ➞ False

In [63]:

**def** prime(number):

count**=**1

**if** number**>**1:

**for** i **in** range(2,number**+**1):

**if** number**%i**==0:

count**+=**1

**if** count**==**2:

**return** **True**

**return** **False**

In [ ]:

**def** prime(number):

count**=**1

**if** number**>**1:

**for** i **in** range(2,number**+**1):

**if** number**%i**==0:

count**+=**1

**if** count**==**2:

**return** **True**

**return** **False**

In [64]:

prime(7)

Out[64]:

True

In [65]:

prime(56963)

Out[65]:

True

**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.

Examples

to\_boolean\_list("deep") ➞ [False, True, True, False]

**deep converts to 0110**

**d is the 4th alphabet - 0**

**e is the 5th alphabet - 1**

**e is the 5th alphabet - 1**

**p is the 16th alphabet - 0**

to\_boolean\_list("loves") ➞ [False, True, False, True, True]

to\_boolean\_list("tesh") ➞ [False, True, True, False]

In [106]:

**import** string **as** s

**def** to\_boolean\_list(string):

letters**=**s**.**ascii\_letters

d**=**{}

l**=**[]

**for** i **in** string:

d[i]**=**letters**.**index(i)

**for** i **in** d**.**values():

**if** i**%2**==0:

l**.**append(**True**)

**else**:

l**.**append(**False**)

**return** l

In [107]:

to\_boolean('loves')

Out[107]:

[False, True, False, True, True]

In [108]:

to\_boolean\_list("tesh")

Out[108]:

[False, True, True, False]