GitHub Link : https://notebooks.githubusercontent.com/view/ipynb?browser=chrome&color\_mode=auto&commit=a052498eff1c1c5b4d6d5633fc0c91f2539fef80&device=unknown&enc\_url=68747470733a2f2f7261772e67697468756275736572636f6e74656e742e636f6d2f6b617274686562616e732f694e6575726f6e2f613035323439386566663163316335623464366435363333666330633931663235333966656638302f4164762e50726f676d25323041737369676e253230312e6970796e62&logged\_in=false&nwo=karthebans%2FiNeuron&path=Adv.Progm+Assign+1.ipynb&platform=android&repository\_id=360934072&repository\_type=Repository&version=99

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

**def** check\_score(a):

symbol\_values **=** {'#':5, 'O' : 3, 'X' : 1, '!' : **-**1, '!!': **-**3, '!!!' : **-**5}

l1 **=** []

sum1 **=** 0

**for** i **in** a:

**for** j **in** i:

**if** j **in** symbol\_values:

l1**.**append(symbol\_values[j])

**for** num **in** l1:

sum1**+=**num

print(sum1)

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

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

prod **=** 1

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

prod**\*=**i

print(prod)

combinations(2,3)

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': '----.',

'&': '.-...', "'": '.----.', '@': '.--.-.', ')': '-.--.-', '(': '-.--.',

':': '---...', ',': '--..--', '=': '-...-', '!': '-.-.--', '.': '.-.-.-',

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

}

Ans : **def** encode\_morse(a):

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': '----.', '&': '.-...', "'": '.----.', '@': '.--.-.',

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

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

b **=** []

**for** j **in** a:

**for** i **in** char\_to\_dots**.**keys():

**if** j **==** i:

x **=** char\_to\_dots**.**get(i)

b**.**append(x)

c **=** ''**.**join(b)

print(c)

encode\_morse("EDABBIT CHALLENGE")

.-...--...-.....--.-......-.-...-...-.--..

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

Ans : **def** prime(a):

*#a = 5151512515524*

l1 **=** []

**for** i **in** range(1, a**+**1):

**if** (a**%i**) == 0:

l1**.**append(i)

**if** len(l1)**>**4:

**break**

**if** len(l1)**>**2:

print(**False**)

**else**:

print(**True**)

prime(7)

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]

Ans : **def** to\_boolean\_list(a) :

alphabets **=** { "a":1,'b':0,'c':1,"d":0,"e":1,"f":0,"g":1,"h":0,"i":1,"j":0,"k":1,"l":0,'m':1,"n":0,

"o":1,"p":0,"q":1,"r":0,"s":1,"t":0,"u":1,"v":0,"w":1,"x":0,"y":1,"z":0 }

*#a = "deep"*

x **=** []

**for** i **in** a:

**for** j **in** alphabets**.**keys():

**if** i **==** j:

d **=** alphabets**.**get(j)

x**.**append(d)

l1 **=** []

**for** i **in** x:

**if** i **==** 1:

l1**.**append(**True**)

**else**:

l1**.**append(**False**)

print(l1)

to\_boolean\_list("deep")

[False, True, True, False]