[Online Python Challenges - Python Principles](https://pythonprinciples.com/challenges/)

* Write a function named capital\_indexes. The function takes a single parameter, which is a string. Your function should return a list of all the indexes in the string that have capital letters.For example, calling capital\_indexes("HeLlO") should return the list [0, 2, 4].

Ans: def capital\_indexes(s):

return [pos for pos, char in enumerate(s) if char.isupper()]

* Write a function named mid that takes a string as its parameter. Your function should extract and return the middle letter. If there is no middle letter, your function should return the empty string.For example, mid("abc") should return "b" and mid("aaaa") should return "".

Ans: def mid(s):

return "" if len(s)%2==0 else s[len(s)//2]

* The aim of this challenge is, given a dictionary of people's online status, to count the number of people who are online.For example, consider the following dictionary:

statuses = {

"Alice": "online",

"Bob": "offline",

"Eve": "online",

}

In this case, the number of people online is 2.Write a function named online\_count that takes one parameter. The parameter is a dictionary that maps from strings of names to the string "online" or "offline", as seen above.Your function should return the number of people who are online.

Ans: def online\_count(s):

return list(s.values()).count("online")

* Define a function, random\_number, that takes no parameters. The function must generate a random integer between 1 and 100, both inclusive, and return it.Calling the function multiple times should (usually) return different numbers.For example, calling random\_number() some times might first return 42, then 63, then 1.

Ans: import random

def random\_number():

return random.randint(1,100)

* Write a function named only\_ints that takes two parameters. Your function should return **True** if both parameters are integers, and **False** otherwise.For example, calling only\_ints(1, 2) should return **True**, while calling only\_ints("a", 1) should return **False**.

Ans: def only\_ints(s1,s2):

return (type(s1)==type(s2)==int)

* The goal of this challenge is to analyze a string to check if it contains two of the same letter in a row. For example, the string "hello" has l twice in a row, while the string "nono" does not have two identical letters in a row.Define a function named double\_letters that takes a single parameter. The parameter is a string. Your function must return **True** if there are two identical letters in a row in the string, and **False** otherwise.

Ans: def double\_letters(s):

return any([a == b for a, b in zip(s, s[1:])])

* Write a function named add\_dots that takes a string and adds "." in between each letter. For example, calling add\_dots("test") should return the string "t.e.s.t".Then, below the add\_dots function, write another function named remove\_dots that removes all dots from a string. For example, calling remove\_dots("t.e.s.t") should return "test".If both functions are correct, calling remove\_dots(add\_dots(string)) should return back the original string for any string.(You may assume that the input to add\_dots does not itself contain any dots.)

Ans: def add\_dots(s):

return ".".join(s)

def remove\_dots(s):

return s.replace(".","")

* Define a function named count that takes a single parameter. The parameter is a string. The string will contain a single word divided into syllables by hyphens, such as these:

"ter-min-a-tor"

Your function should count the number of syllables and return it.For example, the call count("ho-tel") should return 2.

Ans: def count(s):

return s.count("-")+1

* Two strings are anagrams if you can make one from the other by rearranging the letters.Write a function named is\_anagram that takes two strings as its parameters. Your function should return **True** if the strings are anagrams, and **False** otherwise.For example, the call is\_anagram("typhoon", "opython") should return **True** while the call is\_anagram("Alice", "Bob") should return **False**.

Ans: def is\_anagram(s1,s2):

return sorted(s1)==sorted(s2)

* Write a function that takes a list of lists and flattens it into a one-dimensional list.Name your function flatten. It should take a single parameter and return a list.For example, calling:flatten([[1, 2], [3, 4]]).Should return the list:[1, 2, 3, 4]

Ans: def flatten(nested\_list):

return [item for sublist in nested\_list for item in sublist]

* Define a function named largest\_difference that takes a list of numbers as its only parameter.Your function should compute and return the difference between the largest and smallest number in the list.For example, the call largest\_difference([1, 2, 3]) should return 2 because 3 - 1 is 2.You may assume that no numbers are smaller or larger than -100 and 100.

Ans: def largest\_difference(s):

return max(s)-min(s)

* Define a function named div\_3 that returns **True** if its single integer parameter is divisible by 3 and **False** otherwise.For example, div\_3(6) is **True** because 6/3 does not leave any remainder. However div\_3(5) is **False** because 5/3 leaves 2 as a remainder.

Ans: def div\_3(n):

return n%3==0

* Here's the backstory for this challenge: imagine you're writing a tic-tac-toe game, where the board looks like this:
* 1: X | O | X
* -----------
* 2: | |
* -----------
* 3: O | |
* A B C
* The board is represented as a 2D list:
* board = [
* ["X", "O", "X"],
* [" ", " ", " "],
* ["O", " ", " "],
* ]

Imagine if your user enters "C1" and you need to see if there's an X or O in that cell on the board. To do so, you need to translate from the string "C1" to row 0 and column 2 so that you can check board[row][column].Your task is to write a function that can translate from strings of length 2 to a tuple (row, column). Name your function get\_row\_col; it should take a single parameter which is a string of length 2 consisting of an uppercase letter and a digit.For example, calling get\_row\_col("A3") should return the tuple (2, 0) because A3 corresponds to the row at index 2 and column at index 0in the board.

Ans: def get\_row\_col(s):

return (int(s[1])-1,0 if s[0]=="A" else 1 if s[0]=="B" else 2)

* A string is a palindrome when it is the same when read backwards.For example, the string "bob" is a palindrome. So is "abba". But the string "abcd" is not a palindrome, because "abcd" != "dcba".Write a function named palindrome that takes a single string as its parameter. Your function should return **True** if the string is a palindrome, and **False** otherwise

Ans: def palindrome(s):

return s==s[::-1]

* Define a function named up\_down that takes a single number as its parameter. Your function return a tuple containing two numbers; the first should be one lower than the parameter, and the second should be one higher.For example, calling up\_down(5) should return (4, 6).

Ans: def up\_down(n):

return ((n-1,n+1))

* The goal of this challenge is to analyze a binary string consisting of only zeros and ones. Your code should find the biggest number of consecutive zeros in the string.For example, given the string:"1001101000110"The biggest number of consecutive zeros is 3.Define a function named consecutive\_zeros that takes a single parameter, which is the string of zeros and ones. Your function should return the number described above.

Ans: def consecutive\_zeros(s):

return max([len(i) for i in s.split("1")])

* Define a function named all\_equal that takes a list and checks whether all elements in the list are the same.For example, calling all\_equal([1, 1, 1]) should return **True**.

Ans: def all\_equal(s):

return all([i==j for i,j in zip(s,s[1:])])

* Define a function named triple\_and that takes three parameters and returns **True** only if they are all **True** and **False** otherwise.

Ans: def triple\_and(s1,s2,s3):

return all([s1,s2,s3])

* Define a function named convert that takes a list of numbers as its only parameter and returns a list of each number converted to a string.For example, the call convert([1, 2, 3]) should return ["1", "2", "3"].What makes this tricky is that your function body must only contain **a single line of code**.

Ans: def convert(s):

return [str(i) for i in s]

* The built-in zip function "zips" two lists. Write your own implementation of this function.Define a function named zap. The function takes two parameters, a and b. These are lists.Your function should return a list of tuples. Each tuple should contain one item from the a list and one from b.You may assume a and b have equal lengths.If you don't get it, think of a zipper.For example:zap( [0, 1, 2, 3], [5, 6, 7, 8])

Should return:[(0, 5), (1, 6), (2, 7), (3, 8)]

Ans: def zap(a,b):

return [(a[i],b[i]) for i in range(len(a))]

* The aim of this challenge is to write code that can analyze code submissions. We'll simplify things a lot to not make this too hard.Write a function named validate that takes code represented as a string as its only parameter.Your function should check a few things:
* the code must contain the **def** keyword
  + otherwise return "missing def"
* the code must contain the : symbol
  + otherwise return "missing :"
* the code must contain ( and ) for the parameter list
  + otherwise return "missing paren"
* the code must not contain ()
  + otherwise return "missing param"
* the code must contain four spaces for indentation
  + otherwise return "missing indent"
* the code must contain validate
  + otherwise return "wrong name"
* the code must contain a **return** statement
  + otherwise return "missing return"

If all these conditions are satisfied, your code should return **True**.Here comes the twist: your solution must return **True** when validating itself.

Ans: def validate(s):

return "missing def" if "def" not in s else "missing :" if ":" not in s else "missing paren" if "(" not in s or ")" not in s else "missing param" if "()" in s.split(":")[0] else "missing indent" if "\n " not in s else "wrong name" if "validate" not in s else "missing return" if "return" not in s else True

* Define a function named list\_xor. Your function should take three parameters: n, list1 and list2.Your function must return whether n is exclusively in list1 or list2.In other words, if n is in both lists or in none of the lists, return **False**. If n is in only one of the lists, return **True**.For example:

list\_xor(1, [1, 2, 3], [4, 5, 6]) == **True**

list\_xor(1, [0, 2, 3], [1, 5, 6]) == **True**

list\_xor(1, [1, 2, 3], [1, 5, 6]) == **False**

list\_xor(1, [0, 0, 0], [4, 5, 6]) == **False**

Ans: def list\_xor(n,list1,list2):

return (n in list1) ^ (n in list2)

* Define a function param\_count that takes a variable number of parameters. The function should return the number of arguments it was called with.For example, param\_count() should return 0, while param\_count(2, 3, 4) should return 3.

Ans: def param\_count(\*args):

return len(args)

* Write a function named format\_number that takes a non-negative number as its only parameter.Your function should convert the number to a string and add commas as a thousands separator.For example, calling format\_number(1000000) should return "1,000,000".

Ans: def format\_number(n):

return format(n,",")