Exercises - Lab 4

Try the following in lab, or on your own.

1) In lecture you saw some version of a Caesar cipher which can take in a "key" or offset and encodes a string by offsetting each character in that string by the value of key. Implement a classic Caesar cipher encoding where the docstring is given below,

```
def classic_encode(s:str, key:int)-> str:
        """
        Takes in a string containing only lowercase a-z letters, s, and
encodes it based off an offset value
        key, following a classic Caesar cipher. See the examples below for
how to handle key values greater than 26

        Examples:
        >>> classic_encode("abc",3)
        "def"
        >>> classic_encode("xyz",5)
        "cde"
        >>> classic_encode("abc",26)
        "abc"
        >>> classic_encode("abc",53)
        "bcd"
        """
```

2) Implement the following function

```
def make_angry(s):
        """
        Creates and returns a new string equivalent to s, except all
lowercase letters are now uppercase,
        and all periods are replaced with exclamation points.
```

```
        Examples:
        >>> make_angry("I want some coffee.")
        "I WANT SOME COFFEE!"
        >>> make_angry("I'm not sure. Let's wait...")
        "I'M NOTE SURE! LET'S WAIT!!!"
        """
```

3) Implement the following function

```
def count_sentences(s):
        """
        Returns the number of sentences which occur in a string s. Assume a
sentence is any substring
        begins with a capital letter and completes the first time one of ".",
"!", or "?" occurs after said capital letter.

        Examples:
        >>> count_sentences("I like dogs. You like cats? I hate snakes!")
        3
        >>> count_sentences("i type like child!")
        0
        >>> count_sentences("Hmm, maybe...")
        1
        >>> count_sentences("this is a weIrd case?")
        1
        """
```

More questions to come...

4. Write a function that returns the maximum number of consecutive ``special symbols'' in a string. The special symbols are defined in a constant above the function definition.

```
SPECIAL_SYMBOLS = '!@#$%^&*()_+=[]?/'
def symbol_count(s):
    """(str) -> int
    Return the largest number of consecutive "special symbols" in the string
s.
```

```
    >>> symbol_count('c0mput3r')
    0
    >>> symbol_count('H! [here')
    1
    >>> symbol_count('h3!!&o world@#')
    3
    """
    # Your code goes here
```

5. Write the following Python function:

```
'''A "binary string" is a string composed only of the binary digits 0 and
1. Each character in the string represents a single "bit."

This question asks you to write a function that implements a binary
operation -- bitwise exclusive or (XOR) -- on two binary strings. For
two binary strings, the bitwise XOR is computed by performing XOR on
each pair of bits that have the same index. The XOR of two bits is
defined as follows:

XOR(0, 0) = 0
XOR(0, 1) = 1
XOR(1, 0) = 1
XOR(1, 1) = 0

You may assume that the two string arguments will have the same length.
'''


def string_xor(s1, s2):
    '''(str, str) -> str
    Return the bitwise XOR of two binary strings.
    Assumption: len(s1) == len(s2)

    >>> string_xor("", "")
    ''
    >>> string_xor("0", "0")
    '0'
    >>> string_xor("1", "0")
    '1'
    >>> string_xor("1011", "0010")
    '1001'
```

```
    '''

    # Implement this function
```

6. Write a function that given two strings A and B checks that for every character a in A there is a character b in B such that ord(a) + ord(b) > 200. Use the following function signature.

```
def foo(A:str, B:str) -> bool:
    """Checks for all a in A there is b in B such that ord(a) + ord(b) > 200.
    >>> foo("abc", "fgh")
    True
    >>> foo("abc", "def")
    False
    """
```

7. To speed up texting your friends, you are thinking of leaving out all vowels. Write a Python function `shorten` that takes in an arbitrary string and return your shortened string. Assume that your input will only ever be english letters and removes all uppercase and lowercase vowels.

```
shorten('The quick brown fox jumps over the lazy dog')
>>> 'Th qck brwn fx jmps vr th lzy dg'
```

Challenge Question

Each published book has a unique ISBN (International Standard Book Number), see Wikipedia for details. The ISBN is an example of an *error-detecting code*: it consists of nine digits (between 0 and 9) plus a one-digit *checksum* that serves for detecting if there is an error in the number.

A simple way to calculate checksums is by adding the decimal digits, for example the checksum of 53 would be 8. However, the ISBN checksum

weights each digit differently. If the nine digits are d1...d9 the checksum d10 is calculates as: d10=((1*d1)+(2*d2)+...+(9*d9)) % 11. Thus, the checksum d10 is a number between 0 and 10, with 10 being written as X (therefore, this last digit is an example of a base 11 number).

For example, if digits are 020103803, the checksum is (1*0)+(2*2)+...+(9*3) % 11=X (=10).

The complete ISBN is then written with dashes after the first digit, after the fourth digit, and before the checksum as ISBN 0-201-03803-X. The reason for digits having different weight is that a single transposition of two digits – a common error – can be detected.

For example, in ISBN 0-201-08303-X the digits 3 and 8 are transposed, the checksum is 5 rather than X, so this number is invalid.

Your Task:

1. Write a python function `is_ISBN(n)` that takes a string of 10 characters and tests whether `n` is a valid ISBN:

```
>>> isISBN('020103803X')
True
>>> isISBN('020108303X')
False
```

2. Write a python function `is_ISBN_dash(n)` that takes in an arbitrary string and tests whether `n` is a valid ISBN with dashes, using the function is_ISBN. For example:

```
>>> isISBNdash('0-201-03803-X')
True
>>> isISBNdash('0201-03803-X')
False
```

#pin