CS-UY 1114: Lab 7

Strings and Functions

You must get checked out by your lab CA prior to leaving early. **If you leave without being checked out, you will receive 0 credits for the lab.**

Restrictions

The Python structures that you use in this lab should be restricted to those you have **learned in lecture so far (topics learned in Rephactor may only be used if they have also been taught in lecture)**. Please check with your teaching assistants in case you are unsure whether something is or is not allowed!

Create a new python file for each of the following problems.

Your files should be named *lab[num]_q[num].py* similar to homework naming conventions.

Problem 1: *Function Fun!*

Write the outputs of the following code snippets by hand.

```
def mystery(num, string):
    secret = ""
    for char in string:
        secret += char * num
    return secret

def main():
    print(mystery(5, "hello"))
main()
```

```
def func_one(val):
    num = 10
    if val % 3 == 0:
        print(num)
        print("whoop")
        return True

def func_two(num):
    while num < 20:
        if func_one(num):
            print(num)
        num += 1
    print(num)

def main():
    func_two(13)
main()
```

Problem 2: *Reading With Numbers*

Often times when a word is misspelled, our brain can still make sense of what we're reading. This phenomenon is unofficially recognized as Typoglycemia. For this problem, you will test the limits of Typoglycemia by prompting the user for text and changing the user's input to swap some characters with numbers L1K3 TH15 (like this).

Part A: Defining the Helper Function

First define the `numberify(word)` function. This function accepts a string message and returns an *uppercase version* of this message. Specific characters will be swapped out for numbers as so:

```
A → 4
E → 3
I → 1
S → 5
T → 7
O → 0
```

```
"""
Returns a numberified version of the passed in string word
"""
def numberify(word):
    # Code here
```

Part B: Prompting for User Input

In the `main()`, prompt the user for a message to numberify. A word is numberified *only if* the word has a length greater than 3. You may assume each word will be separated by a single whitespace. You must use `numberify()` in your solution.

The following are examples of possible outputs:

```
Please enter a message to numberify: This message serves to
prove how our minds can do amazing things
Your numberified string is: 7H15 M3554G3 53RV35 TO PR0V3 HOW
OUR M1ND5 CAN DO 4M4Z1NG 7H1NG5
```

```
Please enter a message to numberify: In the beginning it was
hard but your mind is reading it automatically with out even
thinking about it
Your numberified string is: IN THE B3G1NN1NG IT WAS H4RD BUT
Y0UR M1ND IS R34D1NG IT 4U70M471C4LLY W17H OUT 3V3N 7H1NK1NG
4B0U7 IT
```

Problem 3: *Roman Code(r):*

You have been sent back in time, and are now working for the secret service in Ancient Rome. Currently, Rome is at war. One day, Roman soldiers intercepted a messenger delivering a written message to the enemy. However, the message seems to be encoded. As the brightest agent in the empire, you are tasked to find out what the message says. This is the message:

```
3Gn.olwo pd/Q gh5l!d pulAk c_kosk an2 moPn! .y\oausr?
3mqei,sd+ktcbe.KrkcmOpsne!se odYpqo>kulq fag pozrtks dftpqh
/ipaslk!dp4vm fkofwolp9 mjcnre
```

Before you start cracking the code, a fellow agent tells you that they managed to figure out how to decode the message from the captured messenger after some "light persuasion". According to the messenger, you can decoded the message by following these rules:

- The message is separated into multiple parts, with each part starting with a number. Let's call this number N.
- Within each part, starting from the first character after N, keep every N'th character and remove the rest. For example, `2hfik` would be `hi` (start with h and skip every other character).
- The original message stops after 100 letters have been processed in the encoded message (so you only have to decode the first 100 alphabetical characters of the message). If the 100 letter limit is reached in the middle of a part, you can ignore that entire part.

After learning this, like any genius tactician in Ancient Rome, you decide to write a computer program to tell you the decoded message. As a wise man once said, you can split this problem into 2 functions:

1. Write a function to decode each part. This function should take in the following parameters: the entire message, where the part starts, where the part ends, and what the number (N) is for that part, and returns a decoded string.
2. Write a function that splits the messages into different parts, and use the first function you wrote to decode the part. Remember that each part starts with a number. You also need to make sure to stop decoding after you have counted 100 letters. The function should return the entire decoded message.

Hints:

- For the first function, you are giving the start, end, and a step as parameters. What have you learned that might use all 3 of these?
- For the second function, you need to look at each character in the encoded message, so you would need to loop through the message. However, you need to stop after encountering 100 letters, so you must check if the character you are currently looking at is a letter in every step of the loop. Which of the different types of loops you've learned is most useful here?
- Since you are calling the first function in your second function, you need to obtain all the parameters for the first function. You can use the same message as the first parameter, but you will need to obtain the start, end, and step parameters. Think about which of these need to be initialized, and when they need to be updated.

If your code is correct, you should see a readable message being printed.

Here is a sample `main` function that you can use to test your code.

```
def main():
    """
    Test your code here.
    """
    message = "3Gn.olwo pd/Q gh5l!d pulAk c_kosk an2 moPn!
.y\oausr? 3mqei,sd+ktcbe.KrkcmOpsne!se odYpqo>kulq fag pozrtks
dftpqh /ipaslk!dp4vm fkofwolp9 mjcnre"
    print(decode_entire_msg(message))
```

Oh and one more thing. The agent tells you that the messenger said the message is from something called "CAs" to their students. Wonder what that could mean.

Problem 4: *All-Mighty Password*

Often when users create an account, the user's password should follow certain criteria in order for the password to be considered strong. Let's define a strong password as a password that:

- Is greater than or equal to 8 characters in length
- Contains at least 1 special character from the following: `"!", "@", "#", "*"`
- Contains at least 1 uppercase character
- Contains at least 1 lowercase character
- Contains at least 1 number

You're tasked with prompting a user to create a password and then verifying the password is a strong password.

Part A: Prompt the User For a Password

Define the `prompt_user_for_pw()` function which will prompt the user for an input of a password. The user's input is then returned by the function.

```
"""
Prompts for input for a password

Returns a string of the user's input
"""
def prompt_user_for_pw():
    # Code here
```

Part B: Validating the Password's Strength

Define the `is_strong_pw(pw)` function which will, given a string, determine if this string is considered a "strong password" according to the definition outlined above. This function will return a boolean reflecting whether the string is strong or not.

```
"""
Returns a bool indicating whether the given string is a strong
password according to the rules defined previously
"""
def is_strong_pw(pw):
    # Code here
```

Part C: Repeatedly Prompt For A Password

Lastly, in the `main()`, use `prompt_user_for_pw()` and `is_strong_pw(pw)` to first ask the user for a password. If the user's password is too weak, the program must repeatedly both display a message stating so and reprompt for a new password until a strong password is given. Be sure to follow the example output *exactly*.

The following is an example of a possible output:

```
Welcome!
Please create a password: bonk
Password too weak! Strong passwords must be greater than or
equal to 8 characters in length, contain at least 1 special
character from the following: "!", "@", "#", "*" contain at
least 1 uppercase character, contain at least 1 lowercase
character, and contain at least 1 number

Please create a password: Bonk12!
Password too weak! Strong passwords must be greater than or
equal to 8 characters in length, contain at least 1 special
character from the following: "!", "@", "#", "*" contain at
```

```
least 1 uppercase character, contain at least 1 lowercase
character, and contain at least 1 number

Please create a password: Bonk12!!
Thank you! Your password is considered strong.
```