CS-UY 1114: Lab 7

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. 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: *Et Tu, Programmator?*

Write a function called `decode_caesar` that will accept `encoded_message`, a string containing a message encoded using a **Caesar Cipher**, and `key`, which is the shift that will be used to decode the message.

The Caesar cipher works as follows. Let's say we have the following snippet of the lyrics to Electric Light Orchestra's 1985 hit, **"Calling America"**:

```
Talk is cheap on satellite
But all I get is static information
I'm still here
Re-dial on automatic
```

If we wanted to encode this using the Caeser cipher, with a key of (for example) 3. We would shift each letter's alphabetical index by 3 steps (i.e. `"a"` has an alphabetical index of 1, so if we shift it by 3, we'd get 4, or `"d"`, "b" has an alphabetical index of 2, etc.). These are the lyrics encoded:

```
Wdon lv fkhds rq vdwhoolwh
Exw doo L jhw lv vwdwlf lqirupdwlrq
L'p vwloo khuh uh-gldo rq dxwrpdwlf
```

Your job is to **decode** encrypted messages like this, and return the decoded, original message.

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

```
def main():
    """
    Just some sample behavior.
    """
    decryption_key = 3
    line = input("Enter the encoded line: ")
    decryption = decode_caesar(line,decryption_key)
    print(decryption)

main()
```

My recommendation is to define a **separate function that decrypts a single letter**. Then, you can create a new, decoded string letter by letter.

Note the following:

- Your program should be case-sensitive. That is, if our decoding key is 3, `"D"` shifts to `"A"` and `"d"` shifts to `"a"`.
- You may assume that you will always know the value of the decryption key.

- Your program should only shift characters that belong to the English dictionary. As such, you may assume the encoded message will be in English. (Caesar would have likely disapproved of this)
- Naturally, any modules that decode the Caesar cipher (of which I am sure there is at least one), are forbidden in this problem. Everything must be done manually, letter by letter, by you.

HINT: The usefulness of `ord` and `chr` is likely obvious, but `%` (the modulus operator) will also come in very handy. To understand why, consider the instance of shifting `"x"`, `"y"`, or `"z"` when the decryption key is 3. As there is no letter after `"z"`, we must "rotate" back to `"a"`. A look at the Caeser cipher's wiki page (linked above) might help make this clear as well.

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 decoded 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 your 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 currenetly 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 paramters for the first function. You can use the same message as the first paramter, 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: *Reading With Numbers*

Often times when a word is mispelled, 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 5: *Budgeting*

For this problem we will be writing 3 functions as follows.

```
def calculate_income(hourly_rate):
    """
    Calculate weekly income based on hourly rate and hours worked

    :param hourly_rate: the hourly pay rate used to calculate income
    :return: float of total_pay
    """
```

`calculate_income()` will calculate the weekly income of the user. We will assume the user works a standard 5 day work week and will input their whole number hours for each day. Furthermore, any hours worked over 40 hours is considered overtime and will be paid at 1.5 times the standard hourly rate.

The output should appear as follows and the function shoulr return the total pay.

```
Enter the number of hours worked today: 5
Enter the number of hours worked today: 4
Enter the number of hours worked today: 12
Enter the number of hours worked today: 10
Enter the number of hours worked today: 10
```

```
def calculate_expenses():
    """
    Calculate weekly expenses based on user input of expenses

    :return: float of total expenses
    """
```

`calculate_expenses()` function will calculate the total weekly expenses of the user. The user will enter floats of all money they spent in the week until they enter `Q` to quit.

Allow the user to enter all expenses as shown and return the total money they spent.

```
Enter how much money you spent or Q if done: 23
Enter how much money you spent or Q if done: 22
Enter how much money you spent or Q if done: 100
Enter how much money you spent or Q if done: Q
```

```
def budget_outcome(income, expenses):
    """
    Determine whether the user had a gain or loss over the week.

    :param income: total amount of weekly income
    :param expenses: total weekly expenses
    :return: the difference between income and expenses
    """
```

The `budget_outcome()` function should *return the difference between income and expenses* and print a message based on whether the user lost money or gained money over the week as shown.

```
Well done you had a gain of 477.5
```

```
You had a loss of -125.0
```

This `main` function is a nice way to test your code using all three functions.

```
def main():
    income = calculate_income(15)
    expenses = calculate_expenses()
    budget_outcome(income, expenses)
```

Problem 6: *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.
```