CS521 - Homework 7 - Alexander Bean

A. Caesar Cipher Problem

I defined two functions as requested: encrypt_caesar() and decrypt_caesar(). These functions both use the same code, with the exception of adding VS subtracting the secret_key from the Unicode value of the letter. This program did specify creating the two functions, but I believe there is a more optimal way to do this without repeating code.

The functions operate with a for loop with nested if statements. For each letter in the text entered as a argument, if it is an upper or lower case character, it enters a nested loop. If not, is it a whitespace or non-letter text and is skipped over.

In the nested loop, if the value is upper, unicode values for uppercase A and Z are assigned for future calculations. The same thing is done for lowercase, but with lowercase a and z.

I defined a variable for the range of characters to be used by the formula, which is the unicode value of z subtracted by the unicode value of a. This will produce 25, and adding 1 ensures the entire alphabet is processed.

```
# Question 1

def encrypt_caesar(plaintext, secret_key):
    ''' This function encrypts a text by shifting the unicode letters up by a certain key number

    Parameters:
    plaintext (str): Text to be encrypted
    secret_key(int): Number of unicode letters to shift the text over by

    Returns:
    ciphertext (str): Text encrypted with characters shifted up based on unicode value and secret_key.
    '''
    ciphertext = ""

    for i in plaintext:
        if i.isupper() or i.islower():
            if i.isupper():
                first_unicode = ord("A")
                last_unicode = ord("Z")
            else:
                first_unicode = ord("a")
                last_unicode = ord("z")

            char_range = last_unicode - first_unicode + 1

            i = ((ord(i) + secret_key - first_unicode) % char_range) + first_unicode
            i = chr(i)

        ciphertext += i

    return ciphertext

def decrypt_caesar(ciphertext, secret_key):
    ''' This function decrypts a text by shifting the unicode letters down a certain key number

    Parameters:
    ciphertext (str): Text to be decrypted
    secret_key(int): Number of unicode letters to shift the text over by

    Returns:
    plaintext (str): Text decrypted with characters shifted down based on unicode value and secret_key.
    '''
    plaintext = ""

    for i in ciphertext:
        if i.isupper() or i.islower():
            if i.isupper():
                first_unicode = ord("A")
                last_unicode = ord("Z")
            else:
                first_unicode = ord("a")
                last_unicode = ord("z")

            char_range = last_unicode - first_unicode + 1

            i = ((ord(i) - secret_key - first_unicode) % char_range) + first_unicode
            i = chr(i)

        plaintext += i

    return plaintext
```

The next step is using the unicode value of each character and, depending on the encryption/decryption, adding or subtracting the secret key value from it, as well as the value of the first unicode. Then, the

modulus is used to ensure that the unicode values stay within the range of uppercase or lowercase letters and do not change cases. With that calculated, the first unicode value is added to the total, and the encrypted/decrypted unicode value is determined. The unicode value is converted into text value with the chr() function.

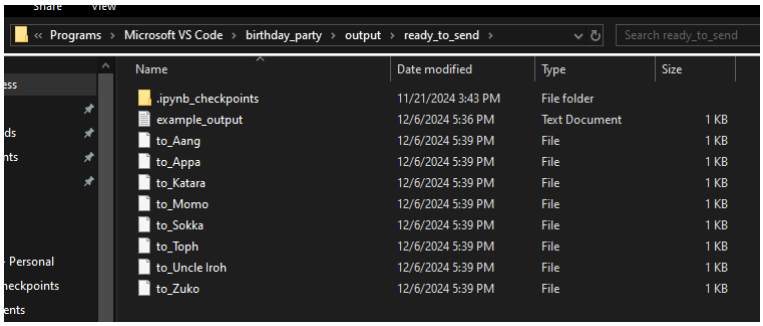
Outside of all loops, the newly determined character is appended to the new text string. With this code, all of the specified test cases pass and correctly encrypt and decrypt the sample messages.

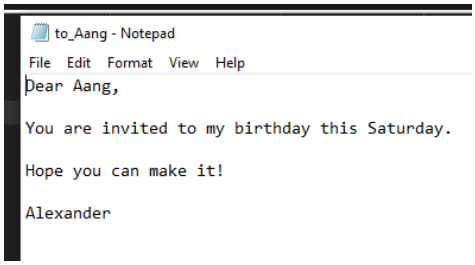
```
# Test Cases

text_str = [("Hello, World!", 3), ("Welcome to Python Programming!", 6),
            ("This is CS 521", 15), ("Happy Halloween!", 5)]
test_num = 0

for plaintext, secret_key in text_str:
    test_num += 1
    print("\nTest", test_num)
    print("Unencrypted:\t", plaintext)

    ciphertext = encrypt_caesar(plaintext, secret_key)
    print("Encrypted:\t", ciphertext)

    decrypted_plaintext = decrypt_caesar(ciphertext, secret_key)
    print("Decrypted:\t", decrypted_plaintext)
```

```
Test 1
Unencrypted:     Hello, World!
Encrypted:       Khoor, Zruog!
Decrypted:       Hello, World!

Test 2
Unencrypted:     Welcome to Python Programming!
Encrypted:       Ckriusk zu Veznut Vxumxgssotm!
Decrypted:       Welcome to Python Programming!

Test 3
Unencrypted:     This is CS 521
Encrypted:       Iwxh xh RH 521
Decrypted:       This is CS 521

Test 4
Unencrypted:     Happy Halloween!
Encrypted:       Mfuud Mfqqtbjjs!
Decrypted:       Happy Halloween!
```

B. The Birthday Party Problem

```
# Question 2

from os import getcwd
path = getcwd()

with open(path+"\\birthday_party\\input\\letters\\starting_letter.txt") as file_obj:
    template = file_obj.read().replace("[insert your name here]", "Alexander")

with open(path+"\\birthday_party\\input\\names\\names_of_invitees.txt") as file_obj:
    names = file_obj.readlines()

for name in names:
    to_invitee = template.replace("[invitee's name]", name.strip())
    with open(path + "\\birthday_party\\output\\ready_to_send\\to_"+name.strip(), mode="w") as file_obj:
        file_obj.write(to_invitee)
```





This problem has been included as a .zip file, as requested in the instructions.

In this problem, I was very careful when identifying the file path to reduce the risk of issues when testing on other devices. The getcwd() command identifies the current working directory and I saved it to a variable for easy reference.

I first opened the starting_letter template file and extracted it as a template. Within the same line, I used the replace() method to fill my name into the [insert your name here] spot.

Next, I opened the file with the names of invitees and used the readlines() function to iterate through the file and append them to a list called names. With that, I had all the names of the people to attend my birthday party.

In the next time, I used a for loop to iterate through each person in the names list. I assigned a variable to the starting_letter example with the letter template and used the replace() method again to replace the [invitee's name] spot with the name of the invitee. I also used the strip() method to remove the trailing newline character.

In the final step, I opened the file in write mode using the file path, and appended the stripped name to the title to ensure the letter name was “To [invitee]”. Once opened, I wrote the to_invitee variable to the file, which included the dynamically populated invitee name, and signed off with my name.

In the screenshots included, each person in the invitee list has their invite addressed directly to them. When opening, the letter is also addressed to them in the recipient line. My name is entered in the signature line.

C. The any() and all() functions

1 - 4.

C. The any() and all() functions

```
# 1. Given a list of integers, check if all elements are non-negative.

num_list = [3, 2, 1, 0]                    # All are non-negative
print(all([num >= 0 for num in num_list]))  # Returns True

num_list = [3, 2, 1, 0, -1]                # There is a negative
print(all([num >= 0 for num in num_list]))  # Returns False
```

```
True
False
```

```
# 2. Given a list of strings, check if any string in the list is a palindrome (reads the same forwards and backwards).

str_list = ["example", "test"]                              # No palindrome
print(any([string == string[::-1] for string in str_list])) # Returns False

str_list = ["example", "test", "kayak"]                     # 'kayak' is a palindrome
print(any([string == string[::-1] for string in str_list])) # Returns True
```

```
False
True
```

```
# 3. Given a list of numbers, determine if all numbers are even.

num_list = [2, 4, 6]                          # All nums are even
print(all([num % 2 == 0 for num in num_list]))  # Returns True

num_list = [2, 4, 6, 7]                       # There is an odd num
print(all([num % 2 == 0 for num in num_list]))  # Returns False
```

```
True
False
```

```
# 4. Given a list of numbers and a test number, determine if the test number is in the list.

num_list = [4, 6, 4, 2, 3, 6, 6, 5, 0, 3, 0, 10, 1, 1, 8, 7, 5, 10, 0, 6]

test_num = 5                                    # Num in list
print(any([num == test_num for num in num_list]))   # Returns True

test_num = 25                                   # Num not in list
print(any([num == test_num for num in num_list]))   # Returns False
```

```
True
False
```

In these problems, I listed two scenarios to demonstrate how the any() or all() function worked to create the intended effect. If any of the values return True within the lists, the any() function will return True. It will return False only if ALL the values return False. If any values return False, the all() function will return False. It will return True only if ALL the values return True. The code is well-documented and should not require further explanation.

5.

```
# 5. Given a list of integers, check if all numbers are non-negative and at least one is a multiple of 10

def PositiveWithAMultOfTen(nums):
    ''' This function calculates if all numbers listed are positive with at least one multiple of 10

    Parameters:
    nums (list): List of integers

    Returns:
    True/False (bool): Condition based on the any() and all() statements
    '''
    if all([num >= 0 for num in num_list]) == True:
        if any([num % 10 == 0 for num in num_list]) == True:
            print("All numbers are positive and a multiple of 10 is present.")
            return True
        else:
            print("All positive, but no multiple of 10 is present.")
            return False
    else:
        print("A negative number is present in the list.")
        return False

num_list = [3, 2, 1, -1]            # There is a negative
print(PositiveWithAMultOfTen(num_list), "\n")    # Returns False

num_list = [3, 2, 1]                # All positive, but no mult of 10
print(PositiveWithAMultOfTen(num_list), "\n")    # Returns False

num_list = [3, 2, 1, 10]            # All positive, mult of 10 present
print(PositiveWithAMultOfTen(num_list))    # Returns True
```

```
A negative number is present in the list.
False

All positive, but no multiple of 10 is present.
False

All numbers are positive and a multiple of 10 is present.
True
```

This nested for-loop takes a list of numbers and determines if all numbers are positive and if at least one of them is a multiple of 10. The outer loop checks whether all of the values are positive. If false, it means there is a negative number in the list. If true, it enters the next loop.

The next loop checks if any of the numbers are a multiple of 10. If false, a multiple of 10 is not present. If true, at least one is. To achieve this, I used the modulus to cleanly identify multiples of 10. Any multiple of 10 would have no remaining value, aka zero. I found in my research that calculating this way does not work with negative numbers–however, since any lists that make it into the nested loop have been verified to only contain positives, this is a non-issue.

Like with the previous questions, I listed well-documented test cases to demonstrate the functionality of the code and any()/all() functions.

6.

```python
# 6. Given a list of dictionaries where each dictionary represents a person with 'name' and 'age' keys, check if:
# - All persons are above 18, and
# - at least one is above 30.

def Over18AtLeastOneAbove30(persons):
    ''' This function calculates if all people listed are above 18 with at least one over 30

    Parameters:
    persons (list): List of nested dictionaries with names and ages

    Returns:
    True/False (bool): Condition based on the any() and all() statements
    '''
    if all([person['age'] >= 18 for person in persons]) == True:
        if any([person['age'] >= 30 for person in persons]) == True:
            print("All persons are above 18 and at least one is above 30.")
            return True
        else:
            print("All persons are above 18, but no one is above 30.")
            return False
    else:
        print("Not all persons in the list are above 18.")
        return False

persons = [{'name': 'Alice', 'age': 10}, {'name': 'Bob', 'age': 25}]    # One is not above 18
print(Over18AtLeastOneAbove30(persons), "\n")                           # Returns False

persons = [{'name': 'Alice', 'age': 25}, {'name': 'Bob', 'age': 25}]    # All above 18, no one above 30
print(Over18AtLeastOneAbove30(persons), "\n")                               # Returns False

persons = [{'name': 'Alice', 'age': 25}, {'name': 'Bob', 'age': 35}]    # All above 18 and one above 30
print(Over18AtLeastOneAbove30(persons))                                 # Returns True
```

```
Not all persons in the list are above 18.
False

All persons are above 18, but no one is above 30.
False

All persons are above 18 and at least one is above 30.
True
```

This nested for-loop takes a list of nested dictionaries with names and ages. It determines if all ages are 18 and above and if at least one age is 30 and above. The outer loop checks whether all of the age values are 18 and above. If false, it means there is an age below 18 in the list. If true, it enters the next loop.

The next loop checks if any of the ages are 30 or greater. The calculation for this is done in the same way as the previous step, but swapping 18 for 30. If false, an age above 30 is not present. If true, at least one is. This program is simpler than Question 5, and should require no further explanation.

Like with the previous questions, I listed well-documented test cases to demonstrate the functionality of the code and any()/all() functions.

D. Read/Write Data

```
from random import randint
from os import getcwd

directory = getcwd()

file_name = input(print("Please enter the name of the file: "))
print(f"The file will be named: {file_name}.txt")

file_path = directory + "\\" + file_name + ".txt"
#print("\n", file_path)

with open(file_path, mode="w") as file_obj:
    for i in range(1, 101):
        file_obj.write(str(randint(1, 50)) + "\t")

with open(file_path) as file_obj:
    print("\n", file_obj.read())
```

```
Please enter the name of the file:
The file will be named: random_integer.txt

 47	19	43	36	1	44	2	43	1	45	32	19	46	6	42	33
42	4	41	43	38	1	34	11	38	46	41	6	31	50	17	50
45	48	9	26	12	25	50	25	30	48	30	21	39	46	5	33
6	19	5	19	18	3	16	47	43	17	44	6	9	46	22	37
15	7	9	28	10	30	34	40	40	49	2	12	44	31	45	1
2	6	8	20	34	46	22	12	35	24	45	48	18	16	13	37
30	47	3	3
```

This program was simpler to write than Question B. I was very careful when identifying the file path to reduce the risk of issues when testing on other devices. The getcwd() command, as used in Question B, identifies the current working directory and I saved it to a variable for easy reference.

I used the input from the user to create the file with the requested name. I then opened the file in Write mode to populate it with 100 random integers between 1 and 50. To do this, I created a for loop to iterate over a range of 100, and had the program write a random integer determined by the randint() function. Then, to format it as requested, I added a tab break \t at the end.

To confirm, I opened the file in read mode and printed the contents. Additionally, the file is located in the specified directory and displays the same output as printed in the console.

E. Remove Text

```
# Create testing file

from os import getcwd
directory = getcwd()

file_name = "my_text"
file_path = directory + "\\" + file_name + ".txt"

with open(file_path, mode="w") as file_obj:
    file_obj.write("Good morning! How are you doing today?\nIt's a beautiful morning. Hope your day is going well.")

# Remove text

file_name = str(input(print("Please enter the name of the file: ")))
str_to_remove = str(input(print("Please enter the string to be removed from your file: ")))

with open(file_path) as file_obj:
    file_text_before = file_obj.read()

file_text_after = file_text_before.replace(str_to_remove, "")

print("\nThe contents of your file are:\n" + file_text_before)
print("\nThe contents, after modification are:\n" + file_text_after)

with open(file_path, mode="a") as file_obj:
    file_obj.write("\n" + file_text_after)
```

```
Please enter the name of the file:
Please enter the string to be removed from your file:

The contents of your file are:
Good morning! How are you doing today?
It's a beautiful morning. Hope your day is going well.

The contents, after modification are:
Good ! How are you doing today?
It's a beautiful . Hope your day is going well.
```

I defined variables for the file name and path for simplicity. I first initialize the file by opening a file in write mode and writing a sample text to it.

Then, it prompts a user to enter the name of the file and the text to remove. I then open the file to parse the text and assign it to a variable. Then, I use the variable and the replace() method to remove the specified text and assign it to a new variable. I did this because I encountered issues when assigning variables and changing the text in one “with open()” block. This way, I was able to effectively capture the text in the file and make the necessary changes.

At the very end, the previous and new contents are returned to the user. Then, the new string after removing the text is written to the file, effectively replacing the original text.

F. Count Words

```
# Reading from file

from os import getcwd
directory = getcwd()

file_name = "gettysburg"
file_path = directory + "\\" + file_name + ".txt"

with open(file_path) as file_obj:
    text = file_obj.read()

# Counting words

print(text)

text = text.split()
count = 0

for word in text:
    if any([letter.isupper() or letter.islower() for letter in word]):
        count += 1

print("---\n\nThere are", count, "words in this text.")
```

```
Four score and seven years ago our fathers brought forth on this continent, a new nation,
conceived in Liberty, and dedicated to the proposition that all men are created equal.

Now we are engaged in a great civil war, testing whether that nation, or any nation so
conceived and dedicated, can long endure. We are met on a great battle-field of that war.
We have come to dedicate a portion of that field, as a final resting place for those who here gave their
lives that that nation might live. It is altogether fitting and proper that we should do this.

But, in a larger sense, we can not dedicate -- we can not consecrate -- we can not hallow -- this ground.
The brave men, living and dead, who struggled here, have consecrated it, far above our poor power to add or detract.
The world will little note, nor long remember what we say here, but it can never forget what they did here.
It is for us the living, rather, to be dedicated here to the unfinished work which they who fought here have thus
far so nobly advanced. It is rather for us to be here dedicated to the great task remaining before us -- that from these
honored dead we take increased devotion to that cause for which they gave the last full measure of devotion -- that
we here highly resolve that these dead shall not have died in vain -- that this nation, under God, shall have a new
birth of freedom -- and that government of the people, by the people, for the people, shall not perish from the earth.

---

There are 270 words in this text.
```

My first step was to read the file, so I used the same commands as in the other questions to get the directory and read the contents of the file. I assign the content to the text variable for easy manipulation.

I used the text.split() method to split the text based on white space. This way, I was able to produce a list of individual words split on the spaces, and could then eliminate any text that doesn't classify as a word. I also created the count variable to support the following loop to count all the words in the text.

The next step was to make a for loop to iterate through every "word" in the split text object. Within this, I had another for loop with list comprehension. It evaluated every single letter in the word with an any() function to determine if any uppercase or lowercase letters were present. If so, it counted as a word and the count variable was increased. If not, then the counter did not increase.

I did notice, however, that the word count was not quite correct. The actual Gettysburg address is 271/272 words (depending on the spacing/hyphenation of 'battlefield'). This text has it as 'battle-field', so this address is 271 words. However, my code produced 270. I looked into this, and found that the provided text file is missing a "so" in the line "...or any nation so conceived and [so] dedicated…". So, my program is calculating correctly, as it seems the source file is missing a word.