Girls' Programming Network

Cryptography

Create a Caesar and Vigenere Cipher Cracker!

This project was created by GPN Australia for GPN sites all around Australia!

This workbook and related materials were created by tutors at:

Sydney, Canberra and Perth

Girls' Programming Network

If you see any of the following tutors don't forget to thank them!!

Writers

Renee Noble
Courtney Ross
Branda Zhong

Testers

Alex McCulloch

Part 0: Setting up

Task 0.1: Making a python file

Open the start menu, and type 'IDLE'. Select IDLE 3.6.

1. Go to the file menu and select 'New File'. This opens a new window.
2. Go to the file menu, select 'Save As'
3. Go to the Desktop and save the file as 'caesar_cracker.py'

Task 0.2: You've got a blank space, so write your name!

At the top of the file use a comment to write your name!
Any line starting with # is a comment.

```
# This is a comment
```

☑ CHECKPOINT ☑

If you can tick all of these off you can go to Part 1:

- ☐ You should have a file called caesar_cracker.py
- ☐ Your file has your name at the top in a comment
- ☐ Run your file with F5 key and it does nothing!!
- ☐ You understand what you're going to build

Files

Part 1: The English Dictionary

Task 1.1: Set the Stage!

Our caesar cracker needs to be able to detect when it has found some english words. So it can do this, we need to tell our program what english words are!

We're going to use a set to do this. This guarantees that each word is only listed once, and so python can search through it really quickly.

1. Create a variable at the top of your file called dictionary. Use it to store an empty set

Hint

You can create a set using the following code:

```
set()
```

Hint

Global variables are variables that can be accessed by all the code in your program. They need to be located at the top of your program.

Task 1.2: Reading Words

Now we need to define every word in the English language!

Luckily for you, we've got a file called **`dictionary.txt`** you can download from girlsprogramming.network/crypto-files3 that you can use to read in all the words!

1. Download the file
2. Write the code that will open up the file and go through it line by line.

Hint

You can open files for reading with the following code:

```
with open('x.txt') as f:
      for line in f:
            #do something with data
```

Task 1.3: Defining Words

1. As we go through the words in the file, `add` them to the `dictionary`!
2. Don't forget to remove all whitespace from the end of each line that is being read in.

Hint

`.strip()` removes white spaces. The output will be `Hello`

```
string = ' Hello '
string = string.strip()
print(string)
```

Hint

Use `add` to add things to sets.

```
my_set = set()
my_set.add("1")
```

☑ CHECKPOINT ☑

If you can tick all of these off you can go to Part 2:

☐ You have a global variable of type set called dictionary

☐ You have read in every word from dictionary.txt and added it to the dictionary set

☐ You have printed out the set to check that your code works

Part 2: Counting Words

Task 2.1: Creating functions

To determine if the caesar cracker has found the right key, we need to test the decrypted text to see how many English words it has.

1. Create a function called **`count_english`**, and give it a parameter called **`text`**.

Hint

The following function is called subtract, and has two parameters, num1 and num2.

```
def subtract(num1, num2):
      # this function subtracts num2 from num1
```

Task 2.2: Splitting Words

1. In the **`count_english`** function, create a **`list`** of **`words`** by splitting the **`text`** on spaces.

Hint

`.split()` separates a string of characters into bits at a specific character or group of characters.

```
string = 'blue red green'
print(string.split())

The output will be ['blue', 'red', 'green']
```

Task 2.3: One, Two, Three!

Your program needs to track how many English words were found in the text so we can check to see if the best key was found.

1. Create a variable called **`counter`**, and set it to **`0`**.

Task 2.4: Is it a word?

Check to see how many English words there are in the text.

1. Create a for loop to go over every word in the text
2. For every word, check to see if it's in the dictionary
3. If the word is in the dictionary, increase the counter by 1.

Task 2.5: Returning the count

1. Now that the program has counted how many English words there are in the text, we need to return the total after we've finished looping through all the words.

Hint

To return a value to the main code we can use the word `return` like this:

```
def subtract(num1, num2):
      return num1 - num2
```

Task 2.6: Testing Time!

1. Check to make sure your function works by calling it using different texts. Don't forget to test it using some fake words too!

Hint

You can call the subtract function used in a previous hint like this:

```
total = subtract(1, 2)
```

☑ CHECKPOINT ☑

If you can tick all of these off you can go to Part 3:

- ☐ You have a function called count_english
- ☐ Your function counts how many english words are found in a text
- ☐ Your function returns the number of english words found
- ☐ You have tested your function!

Part 3: Time to crack!

Task 3.1:

1. Create a new function called `caesar_decrypt`. It should take `message` and `key` as parameters.

Task 3.2:

Remember all that code you wrote in the first workbook for decrypting and encrypting caesar ciphers?

1. Add all that code to your function, but change it so it's always decrypting.
2. Set the mode so it always decrypts or change your code so it always decrypts.
3. Instead of printing straight away we want to build up a new encrypted word and return it. Create a variable called `new_message`, set it to be an empty string.
4. Loop through every character in the `message` and:
    - Check to see `if` it's a letter in the `alphabet`
    - Decrypt the character if required
    - Add the character to the `new_message`

Task 3.3: Return the message!

1. `return` the decrypted message.

Hint

Take a look at the hint from Task 2.5 if you get stuck!

Task 3.4: Test your function

1. Make sure your function works by giving it encrypted messages to decrypt!

☑ CHECKPOINT ☑

If you can tick all of these off you can go to Part 4:

- ☐ You have a function called caesar_decrypt
- ☐ Your function decrypts the message
- ☐ Your function returns the new message
- ☐ You have tested your function!

Part 4: Which is the best key?

Task 4.1: I don't know, so let's try all the keys!

If we are given a random secret message then we don't know what key we need! To figure it out we want to work out which key makes the most English words when we use it to decrypt the message. To do this, we're going to try all the keys from 0 to 25 to see which key returns the most words.

1. Create a new function called **`find_best_key`**.
2. It should take one argument, which is the message you want to decrypt.

Task 4.2: Keeping track of the best key

We're going to need two variables inside our function for tracking the best results:

1. Create **`best_key`**, which stores the best key that has been found so far
2. Create **`best_count`**, which stores the corresponding count of words found.
3. Set both of them to be **`0`**

Task 4.3: Using a for loop to test all the keys

Let's use a `for` loop to test all the keys. The keys that are tested should be 0-25.

1. Create a `for` loop to loop through the range 0-25

For each key:

2. Call your decrypter function on your message with the current key.
3. Store the decrypted **`message`** in a variable, e.g. **`decrypted_text`**
4. Use the **`count_english`** function on **`decrypted_text`** to count how many english words are returned, and store this count in another variable.
5. Check `if` this count is higher than your previous other best count. If it is, store it in **`best_count`** and update the **`best_key`** to be the key that you are up to in your loop.

Hint

To test all the keys from 0 - 25, you'll need Python's **`range()`** function. The function will test all the numbers from zero up to but not including whatever number you put in the brackets.
You can use it like this:

```
# the for loop will test the numbers 0, 1 and 2 but NOT 3
for i in range(3):
      print(i)
```

Task 4.4: Return the best key!

1. Once we have finished testing all the keys, we should `return best_key`

Task 4.5: Test your function!

In order to test your function, you definitely don't need to test it for each key.

1. Use your Caesar Cipher from earlier today to create a couple of test cases (try a couple of keys, maybe 1 and 5)
2. Call your `find_best_key` on your encrypted message and print out the result
3. See if your program returns the right key!

☑ CHECKPOINT ☑

If you can tick all of these off you can go to Part 5:

- ☐ You have a function called find_best_key
- ☐ Your function checks each key for number of words returned
- ☐ Your function returns the key with the highest number of english words

Part 5: Putting it all together

Task 5.1: The main game!

Now we have written all our other functions, we need to write a function that brings them all together.

1. Create a `main` function which takes no arguments.

Task 5.2: Get the secret message!

Inside your main function we need to get the message we are going to decrypt.

1. Download one of the secret messages from the website and store it in the same location as your python file.
2. Open the file.
3. Read the message, it will only be 1 line long.
4. Strip any whitespace off the end of the message.

Hint

You can read a whole message in one go using the .read() method. This is great if you only want to read one line.

This might look something like this:

```
with open("my_file.txt") as f:
    contents = f.read()
```

Task 5.3: Finding the best key

Now it's time to use our functions from before!

1. Call the `find_best_key` function you wrote in part 4, put in the secret message you got from the file.
2. Store the result, the best key, in a variable.

Task 5.4: decrypting the message

Now we know the best key we can use our `caesar_decrypt` function to return the decrypted message

1. Call caesar_decrypt on your secret message using the best key you returned in the previous step.
2. Print out the decrypted message!

Task 5.5: Calling your main function

Finally, add a line at the bottom of your file that will call your main function.

Hint

Make sure this is not inside any other functions. This is where your program start running from!

☑ CHECKPOINT ☑

If you can tick all of these off you can go to Extension Part 6:

- ☐ You have a function called main
- ☐ You read in the secret message from a file
- ☐ Your main function calls your get_best_key function
- ☐ Your main function calls caesar_decrypt on the secret message using the best key.
- ☐ You call your main function at the bottom of your code
- ☐ You ran your code and it printed a message that made sense!

Part 6: Extension: Vigenere Cracker

Now we've done the caesar cipher cracker, let's change ours to be a Vigenere Cipher Cracker.

We've found a file called keys.txt, and believe that it contains the keys used to encode some secret documents using vigenere ciphers.

Task 6.1: Copycat

This program is going to be a lot like our Caesar Cipher Cracker. So make a new file and copy your Caesar Cipher Cracker code into it. Name the new file vigenere_cracker.py

Task 6.2: Download the secrets

1. Download a secret message you want to decrypt
2. Download the keys.txt file, the file we believe contains all the secret keys used to encrypt the secret messages.

Task 6.3: Keeper of the keys

Let's get all the keys out of the keys.txt file so we can use them all in our program

1. Open the keys.txt file.
2. Loop through all keys and store them in a list

You should add this code in the same place that you loop through and make your dictionary set! It's also a global variable and it's nice to keep them together.

Task 6.4: Which key is it?

Instead of looping through all of the keys in the range from 0-25, we're going to try all of the secret keys from the keys.txt file.

1. Change your code, so where you used to loop through the numbers 0-25, now loop through all the keys in your list of keys.
2. This time instead of a caesar_decrypt function we need a vigenere_decrypt function. Rename the function and make sure to go through your code and rename it everywhere!
3. Like you did when you created your caesar decrypt, copy code from your vigenere encryption file to create a vigenere decrypt function that returns the decrypted message.
   Make sure you copy across your helper functions from your vigenere file too.