Girls' Programming Network

Cryptography 3

Create a Caesar and Vigenère Cipher Cracker!

TUTORS ONLY

What do you do when you want to know what a secret message says but you don't have the key? We'll be writing a program that tries to crack the code by looking for English words in the decrypted text.

Sets & 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 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

Solution

```
dictionary = set()
```

TUTOR TIPS

What are sets?
Sets are like a list, but can't have duplicates.
Sets are also like just having the key part of a dictionary.

When we look something up in a set we don't have to look through the whole set like a list to find something. It's like checking if something is in a dictionary's set of keys. This makes it faster than using a list to store our collection of words, because we will need to be able to search for individual words quickly.

If you didn't care about speed you could use a list.

TUTOR TIPS

Global variables
Global variables in python. We typically put them at the top of the file. (The actual convention is all uppercase, but we haven't explained that here).

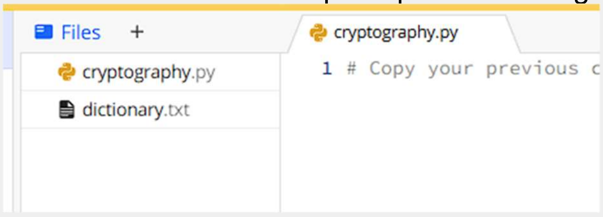
If there is not a variable with that name within the scope of the function then python will use something from the global scope.

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 the GPN website that you can use to read in all the words!

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



```
# Copy your previous c
```

Solution

```
with open("dictionary.txt") as f:
    for line in f:
```

TUTOR TIPS

Opening files with `with`:
Students don't have to use with to open the file. They can just assign `open('x.txt')` to a variable and read from that.

However `with` is preferable because it will automatically close the file when the program finishes or errors out.

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.

Solution

```
with open("dictionary.txt") as f:
    for line in f:
        dictionary.add(line.strip())
```

Full code Lesson 1

The code should look like this (no bonuses):

```
dictionary = set()
with open("dictionary.txt") as f:
    for line in f:
        dictionary.add(line.strip())
```

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`.

Solution

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

```
def count_english(text):
```

Task 2.2: Splitting Words

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

Solution

```
words = text.split()
```

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`.

Solution

```
counter = 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 list
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.

Solution

```
for word in words:
       if word in dictionary:
           counter = counter + 1
```

Task 2.5: Returning the count

1. Now that the program has counted how many English words there are in the text, return the total found

Hint

```
for word in words:
       if word in dictionary:
           counter = counter + 1
         return counter
```

TUTOR TIPS

Make sure students test their individual functions as they go.

Here you could get them to add lines like these to see if it counts only real english words:

```
print(count_english("i like pie"))  >> prints 3
print(count_english("i dont like glarble"))  >> prints 3 (if used "I" and "don't" would print 1
```

☑ 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!

Full code Lesson 2

They should have a function that looks like this:

```
def count_english(text):
   words = text.split()
   counter = 0
   for word in words:
       if word in dictionary:
           counter = counter + 1
   return counter
```

Part 3: Time to crack!

Task 3.1:

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

Solution

```
def caesar_decrypt(message, key):
```

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 decrypted 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`

TUTOR TIPS

They should have a function that looks like this:

Previously we needed to encrypt and decrypt. But now we only need to decrypt.

Students can either use their same code from before and fix to always be in decrypt mode

```
def caesar_decrypt(message, key):
    mode = "d"
    if mode == "d":
        key = -1 * key
    new_message = ""
    for current_letter in message:
        …
```

Or they can remove the unnecessary if statement to clean up their code

```
def caesar_decrypt(message, key):
    key = key * -1
    new_message = ""
    for current_letter in message:
        …
```

Task 3.3: Return the message!

1. `return` the decrypted message.

Task 3.4: Test your function

1. Make sure your function works by giving it an encrypted message to decrypt!
2. Use your Caesar Cipher from earlier today to create a couple of encrypted message test cases (try a couple of keys, maybe 1 and 5)

☑ 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!

TUTOR TIPS

They should have a function that looks like this:

```
# At the top of the file
alphabet = "abcdefghijklmnopqrstuvwxyz"

...

# Anywhere in the file
def caesar_decrypt(message, key):
   key = key * -1
   new_message = ""
   for current_letter in message:
       if current_letter in alphabet:
           current_index = alphabet.index(current_letter)
           new_index = (current_index + key) % 26
           new_letter = alphabet[new_index]
           new_message = new_message + new_letter
       else:
           new_message = new_message + current_letter
   return new_message
```

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 encrypted message, then we don't know what key we need! 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.

Solution

```
def find_best_key(message):
```

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`

Solution

```
def find_best_key(message):
    best_key = 0
    best_count = 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.
For each key:

1. Call your decrypter function on your message with the current key.
2. Store the decrypted `message` in a variable, e.g. `decrypted_text`
3. Use the `count_english` function on `decrypted_text` to count how many english words are returned, and store this count in another variable.
4. 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

```
def find_best_key(message):
    best_key = 0
    best_count = 0
    for key in range(26):
```

TUTOR TIPS

Looking for this logic to track the best key:

```
decrypted_text = caesar_decrypt(message, key)
count = count_english(decrypted_text)
if count > best_count:
    best_key = key
    best_count = count
```

Task 4.4: Return the best key!

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

TUTOR TIPS

In theory we could have kept track of the best decrypted message too, but we've separated that out just for the flow of the workbook. But students can make changes to make this into a function that returns the best decryption, rather than just the 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 encrypted message test cases - don't forget to make a note of the key you used
2. Call your `find_best_key` on your encrypted message and print out the result
3. See if your program returns the right key!

TUTOR TIPS

Get students to use their caesar ciphers to create their own test cases.

For instance they could do this:

```
find_best_key("khoor zruog")
```

Hopefully it will return a key of 3. (It translates to hello world)

☑ 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

Full code Lesson 4

They should have a function that looks like this:

```
def find_best_key(message):
    best_key = 0
    best_count = 0
    for key in range(26):
        decrypted_text = caesar_decrypt(message, key)
        count = count_english(decrypted_text)
        if count > best_count:
            best_key = key
            best_count = count
    return best_key
```

Part 5: Putting it all together

Task 5.1: The main game!

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

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

Solution

```
def main():
```

Task 5.2: Get the secret message!

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

1. Chose an encrypted file to read. Read the message
2. Strip any whitespace off the end of the message.

Solution

```
def main():
    with open("message.txt") as f:
        message = f.read().strip()
```

Task 5.3: Finding the best key

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

Solution

```
key = find_best_key(message)
```

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!

Solution

```
decrypted_text = caesar_decrypt(message, key)
print(decrypted_text)
```

Task 5.5: Calling your main function

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

TUTOR TIPS

The line they are most likely to forget is this one:

```
main()
```

Which will mean their code doesn't run anything, because we just have function definitions.

☑ 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!

TUTOR TIPS

They should have a function that looks like this:

```
def main():
    with open("message.txt") as f:
        message = f.read().strip()
    key = find_best_key(message)
    decrypted_text = caesar_decrypt(message, key)
    print(decrypted_text)

main()
```

Complete Code looks like:

TUTOR TIPS

The code should look like this:

```
alphabet = "abcdefghijklmnopqrstuvwxyz"

dictionary = set()
with open("dictionary.txt") as f:
    for line in f:
        dictionary.add(line.strip())

def count_english(text):
    words = text.split()
    counter = 0
    for word in words:
        if word in dictionary:
            counter = counter + 1
    return counter


def find_best_key(message):
    best_key = 0
    best_count = 0
    for key in range(26):
        decrypted_text = caesar_decrypt(message, key)
        count = count_english(decrypted_text)
        if count > best_count:
            best_key = key
            best_count = count
    return best_key


def caesar_decrypt(message, key):
    key = key * -1
    new_message = ""
    for current_letter in message:
        if current_letter in alphabet:
            current_index = alphabet.index(current_letter)
            new_index = (current_index + key) % 26
            new_letter = alphabet[new_index]
            new_message = new_message + new_letter
        else:
            new_message = new_message + current_letter
    return new_message

def main():
    with open("message.txt") as f:
        message = f.read().strip()
    key = find_best_key(message)
    decrypted_text = caesar_decrypt(message, key)
    print(decrypted_text)

main()
```

Part 6: Extension: Vigenère Cracker

Solution

```python
alphabet = "abcdefghijklmnopqrstuvwxyz"

keywords = []
with open("keys.txt") as f:
    for line in f:
        keywords.append(line.strip())

dictionary = set()
with open("dictionary.txt") as f:
    for line in f:
        dictionary.add(line.strip())

def count_english(text):
    words = text.split()
    counter = 0
    for word in words:
        if word in dictionary:
            counter = counter + 1
    return counter

def find_best_key(message):
    best_key = 0
    best_count = 0
    for key in keywords:
        print(key)
        decrypted_text = vigenere_decrypt(message, key)
        count = count_english(decrypted_text)
        if count > best_count:
            best_key = key
            best_count = count
            print(decrypted_text)
    return best_key

def caesar_letter(letter, key_num):
    current_index = alphabet.index(letter)
    new_index = (current_index + key_num) % 26
    new_letter = alphabet[new_index]
    return new_letter

def vigenere_decrypt(message, key):
    key_nums = []
    for letter in key:
        key_num = alphabet.index(letter)
        key_num = -1 * key_num
        key_nums.append(key_num)

    new_message = ""
    count = 0
    for current_letter in message:
        if current_letter in alphabet:
            key_num = key_nums[count % len(key_nums)]
            new_letter = caesar_letter(current_letter, key_num)
            new_message = new_message + new_letter
            count = count + 1
        else:
            new_message = new_message + current_letter
    return new_message

def main():
    with open("message_vigenere.txt") as f:
        message = f.read().strip()

    key = find_best_key(message)
    decrypted_text = vigenere_decrypt(message, key)
    print("best", key)
    print(decrypted_text)

main()
```