Girls' Programming Network

Cryptography 2

Create a Vigenere Cipher encryptor and decryptor!

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	Testers
Renee Noble	Jess Austin
Caitlin Macleod	Julia Wong
Alex Kaczmarek	
Levie Cabuang	
Pamita Mund	
Courtney Ross	
Feona Sims	
Deanna Arora	
Alex Penna	
Manou Rosenburg	
Sheree Pudney	
Taylah Griffiths	

Part 0: Vigenere Ciphers

A vigenere cipher is like several Caesar ciphers combined.

Previously in our **Caesar cipher** we had a single rotation. For instance a rotation **key number of 3** would mean we rotate our cipher wheel like this:

For **Vigenere ciphers** we use several rotations. Instead of having the same rotation for each letter, we take turns using different rotations for different letters in the message we are encrypting.

Instead of one rotation key number we have a **keyword**, for instance our keyword might be **"gpn"**. "gpn" is a way of telling you the different rotations, 6, 15 and 13.

Keyword		
g	p	n
6	15	13

So if we want to encrypt a secret message we take turns at using the different rotations.
This is what it looks like if we encrypt the message "this is a secret message":

t	h	i	s		i	s		a		s	e	c	r	e	t		m	e	s	s	a	g	e
↓	↓	↓	↓		↓	↓		↓		↓	↓	↓	↓	↓	↓		↓	↓	↓	↓	↓	↓	↓
g	p	n	g		p	n		g		p	n	g	p	n	g		p	n	g	p	n	g	p
6	15	13	6		15	13		6		15	13	6	15	13	6		15	13	6	15	13	6	15
↓	↓	↓	↓		↓	↓		↓		↓	↓	↓	↓	↓	↓		↓	↓	↓	↓	↓	↓	↓
z	w	v	y		x	f		g		h	r	i	g	r	z		b	r	y	h	n	m	t

Use your cipher wheel to encrypt these messages with vigenere ciphers

Check out the Hint below to help and save you counting!

Task 0.1: Hide this message

Encrypt "can you keep this hidden" using the key **"code"**.
We've done some of the work for you. .

c	a	n		y	o	u		k	e	e	p		t	h	i	s		h	i	d	d	e	n
↓	↓	↓		↓	↓	↓		↓	↓	↓	↓		↓	↓	↓	↓		↓	↓	↓	↓	↓	↓
c	o	d		e	c	o		d	e	c	o		d	e	c	o		d	e	c	o	d	e
2	14	3		4	2	14		3	4	2	14		3	4	2	14		3	4	2	14	3	4
↓	↓	↓		↓	↓	↓		↓	↓	↓	↓		↓	↓	↓	↓		↓	↓	↓	↓	↓	↓
e	o	q		c	q	i		n	i	g	d		w	l	k	g		k	m	f	r	h	r

Task 0.2: Hide another

Encrypt "hide this message" using the key **"key"**.

h	i	d	e		t	h	i	s		m	e	s	s	a	g	e
↓	↓	↓	↓		↓	↓	↓	↓		↓	↓	↓	↓	↓	↓	↓
k	e	y	k		e	y	k	e		y	k	e	y	k	e	y
10	4	24	10		4	24	10	4		24	10	4	24	10	4	24
↓	↓	↓	↓		↓	↓	↓	↓		↓	↓	↓	↓	↓	↓	↓
r	m	b	o		x	f	s	w		k	o	w	q	k	k	c

Hint

Counting the wheel can take a long time. You can use this table to look up the indexes of the letters:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

You might also like to write these numbers under the letters on the inner (purple) circle of your wheel. You can use them to quickly turn your purple wheel for a particular key.

Use your cipher wheel to decrypt these vigenere encrypted messages

Just like your Caesar cipher codes, use the negative of each letter in the key to decrypt the message.

To use your cipher wheel to decrypt the first letter in the message below:

- rotate inner purple wheel -7 (ie clockwise 7) (green H should line up with purple A)
- find the letter you are decrypting on the green outer wheel (F)
- get it's matching letter on the purple inner wheel (Y) - this is the decrypted letter - write it in the first space below

Task 0.3: Uncover the message

Can you **decrypt** this message using the key **"hack"**.
We've done some of the work for you. .

f	o	w		m	y	a	e	u	l	d		v	r	l		c	q	n	l
↓	↓	↓		↓	↓	↓	↓	↓	↓	↓		↓	↓	↓		↓	↓	↓	↓
h	a	c		k	h	a	c	k	h	a		c	k	h		a	c	k	h
-7	0	-2		-10	-7	0	-2	-10	-7	0		-2	-10	-7		0	-2	-10	-7
↓	↓	↓		↓	↓	↓	↓	↓	↓	↓		↓	↓	↓		↓	↓	↓	↓
y	o	u		c	r	a	c	k	e	d		t	h	e		c	o	d	e

Task 0.4: Crack another

Can you **decrypt** this message using the key **"bug"**.

u	b	k		f	u	m	m	y		n	b	m		r	b	h	j	f	x
↓	↓	↓		↓	↓	↓	↓	↓		↓	↓	↓		↓	↓	↓	↓	↓	↓
b	u	g		b	u	g	b	u		g	b	u		g	b	u	g	b	u
-1	-20	-6		-1	-20	-6	-1	-20		-6	-1	-20		-6	-1	-20	-6	-1	-20
↓	↓	↓		↓	↓	↓	↓	↓		↓	↓	↓		↓	↓	↓	↓	↓	↓
t	h	e		e	a	g	l	e		h	a	s		l	a	n	d	e	d

☑ CHECKPOINT ☑

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

☐ You encrypted *"can you keep this hidden"* !

☐ You encrypted *"hide this message"* !

☐ You cracked the first decryption!

☐ You flew through the last decryption!

Part 1: Key questions

Task 1.1: New file, new welcome

1. Make a copy of your main.py that you created from the first workbook (rename it to something meaningful).
2. Delete all the code from main.py
3. At the top of the file use a comment to write your name!
4. Print out a welcome message to the users. Something like "Welcome to Vigenere Cipher"

Task 1.2: Ask the user for a message

Ask the user for the **message** they want to encrypt. Store it in a variable.

Task 1.3: Ask the user for a keyword

Ask the user for the **keyword** they want to use to encrypt the message. Store it in a variable.

☑ CHECKPOINT ☑

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

☐ You should have saved a copy of your old code and have a new main.py file

☐ A welcome message is printed

☐ You have asked for and stored a message

☐ You have asked for and stored a keyword

☐ Run your code.

TUTOR TIPS

The code should look like this (no bonuses):

```
# <the student's name>
print('Welcome to Vigenere Cipher')
message = input('What do you want to encrypt? ')
key = input('What is the encryption key? ')
```

★ BONUS 1.4: Customised Welcome

Waiting for the next lecture? Try adding this bonus feature!!

1. Ask the user for their name
2. Print a customised message for the user using their name. Something like this:

```
----------------------------------------
    Welcome to the Vigenere Cipher, Lyndsey
----------------------------------------
```

TUTOR TIPS

The code should look like this (with bonuses):

```python
# <the student's name>
name = input('What is your name? ')
print('Welcome to the Vigenere Cipher, ' + name)
message = input('What do you want to encrypt? ')
key = input('What is the encryption key? ')
```

Part 2: Calculating Keys!

Task 2.1: Lots of keys, lots of maths

This part can be a little tricky, so let's do it by hand first so that we know what the computer should be doing and we can check that it's doing the right thing!

First we have to take the keyword that we are using - let's use **gpn** for an example. We need to turn it into a bunch of number keys. We do this by finding how far each letter in the key is from the letter A! That's its rotation number!

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Keyword	g	p	n
Keynums	6	15	13

Now it's your turn!

Try this one:

Keyword	c	o	d	i	n	g
Keynums	2	14	3	8	13	6

And this one:

Keyword	m	y	k	e	y
Keynums	12	24	10	4	24

Task 2.2: Create the alphabet

1. Create a variable called `alphabet` to store the letters a-z.
   `abcdefghijklmnopqrstuvwxyz`

This variable should be the same as the one that you made in the caesar cipher

TUTOR TIPS

Encourage them to put their alphabet variable at the top of their code. This is the standard place to put global variables like this. It will make sure it's available when they need to use it in the code rather than it getting lost and in the wrong place in their code.

Task 2.3: Create a list for keys

We're going to need a place to store our key numbers. Let's store them in a list!

1. **Make an empty list** called `key_nums`.
   We'll put the key numbers in later!

Hint

You can make an empty list like this:

```
Fruits = []
```

The two square brackets create a list with nothing inside it. An empty list can have items added to it later!

Task 2.4: Get loopy

We're going to want to find the index of every letter in our key.

Create a `for loop` that loops through the letters in our key. Store the current letter as you loop through in a variable called `current_key_letter`.

Hint

An example of looping through letters in a name with a for loop looks like this:

```
for letter in name:
    print(letter)
```

Letter is the variable that will change to hold all the letters of a name, one at a time.

Task 2.5: The index is the rotation!

When we did it by hand we saw the rotation number is actually just the index of the key letter in the alphabet. Let's get the index for each letter.

1. Go to the line below your for loop line, we're going to add code in an indented block here.
2. Our for loop looks at the key letters one at a time. **Find the index** of the `current_key_letter`. **Store it** in a variable called `current_key_num`.

Hint

You can use this to find the position of the letter "p" in a string called word.

```
word = "gpn"
p_index = word.index("p")
```

Task 2.6: Keep the keys!

Now we need to add the `current_key_num` to our list.

1. Go to the line below where you got `current_key_num`, make sure you are still indented inside the loop.
2. Add the `current_key_num` to the list of `key_nums`.

Hint

Remember, you can use append() to add an item to a list!

```
foods = ["chocolate", "pizza", "lollies"]
foods.append("apples")
```

Task 2.7: Print the keys!

To make sure this worked, **print** out **key_nums** to confirm that the numbers are what you expect.

☑ CHECKPOINT ☑

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

☐ You have the alphabet stored

☐ You used a for loop to loop through the letters in the key

☐ You found the index of each key and stored it in a list

☐ You checked your list by printing it out

☐ Run your code

TUTOR TIPS

The code should look like this:

```python
alphabet = 'abcdefghijklmnopqrstuvwxyz'
key_nums = []

name = input('What is your name? ')
print('Welcome to the Vigenere Cipher, ' + name)
message = input('What do you want to encrypt? ')
key = input('What is the encryption key? ')

for current_key_letter in key:
    current_key_num = alphabet.index(current_key_letter)
    key_nums.append(current_key_num)
print(key_nums)
```

Part 3: Encrypt!

Remember Caesar ciphers! Vigenere ciphers are like a lot of Caesar ciphers put together. We rotate through multiple keys, instead of using just one.
So we're going to turn some of our Caesar cipher code into a function that helps us do Caesar cipher encryption for one letter for any key we give it.

Task 3.1: Write the scaffold

Now it's time to write our function. Let's put it right after our alphabet variable.
1. Name it `encrypt_letter`
2. Make it take two parameters:
    ○ The **letter** we want to encrypt
    ○ The **key** we will use to encrypt it.
3. For now, make it `return` the letter without doing anything to it.

Hint

Below is a function that adds two numbers together.
● It is called add
● It passes num1 and num2 as parameters
● It adds the two numbers together
● It returns total as the result

```
def add(num1, num2):
    total = num1 + num2
    return total
```

Task 3.2: Make it work

Now we need to make our function do something. Let's make our function do caesar cypher encryption on the letter that was put in using the key that was put in. .

In the body of your function make it:
1. Find the index of the current letter
2. Calculate the rotated index of the current letter
3. Find the encrypted letter based on the rotated index

Hint

This is the code you should bring in from the caesar cypher:
```
current_index = alphabet.index(letter)
new_index = (current_index + key_num) % 26
new_letter = alphabet[new_index]
```

Task 3.3: Return

1. Update your `return` line to return the new encrypted letter

Task 3.4: Test it!

Now that your function is running some code, you can test it!
Just below where you print `key_nums`, call your function using 'a' as the letter parameter and 1 as the key parameter. What value do you expect it to return? Does it do what you expect?

Hint

Here's how you can call the `sum` function and store the result in the `result` variable:

```
result = sum(1,2)
```

☑ CHECKPOINT ☑

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

☐ You wrote a function for encrypting letters

☐ Your function returns the encrypted letter

☐ You tested the function by calling it and looking at the result

TUTOR TIPS

Test of their function with parameters ('a',1) should return (and print) 'b' - a shifted by 1 is b

The code should look like this:

```python
# <the student's name>
alphabet = 'abcdefghijklmnopqrstuvwxyz'
key_nums = []

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

name = input('What is your name? ')
print('Welcome to the Vigenere Cipher, ' + name)
message = input('What do you want to encrypt? ')
key = input('What is the encryption key? ')

for current_key_letter in key:
    current_key_num = alphabet.index(current_key_letter)
    key_nums.append(current_key_num)

result = encrypt_letter('a', 1)
print(result)
```

Part 4: Matching messages to keys

Normally our message is longer than our key. We saw at the start that we use the same key over and over again by rotating through using the different letters.

Remember this example where the key was _gpn_:

t	h	i	s		i	s		a		s	e	c	r	e	t		m	e	s	s	a	g	e

g	p	n	g		p	n		g		p	n	g	p	n	g		p	n	g	p	n	g	p
6	15	13	6		15	13		6		15	13	6	15	13	6		15	13	6	15	13	6	15

z	w	v	y		x	f		g		h	r	i	g	r	z		b	r	y	h	n	m	t

To make the computer do this we'll need to give it a way to keep count of which key we are up to. This way we won't ask for a position in the key that doesn't exist and get an error!

How do we do that?

1. **We keep count of which letter we are up to in our message:**

Message	s	e	c	r	e	t
Count	0	1	2	3	4	5

2. **Find a way for those numbers to wrap around to match the number of letters in the key.** Since our key is GPN we only want to use numbers 0, 1, 2.

   We can use remainders to wrap around! (remember module (%) from earlier)

   These are the remainders when we divide the count for each position by 3:

   0 ÷ 3 = 0 remainder 0   3 ÷ 3 = 1 remainder 0
   1 ÷ 3 = 0 remainder 1   4 ÷ 3 = 1 remainder 1
   2 ÷ 3 = 0 remainder 2   5 ÷ 3 = 1 remainder 2

   Look all the remainders are 0, 1 or 2!

3. **Put those remainders in the table and find the matching letter from that position in the keyword:**

Message	s	e	c	r	e	t
Count	0	1	2	3	4	5
Remainder	0	1	2	0	1	2
Key letter	g	p	n	g	p	n

Task 4.1: Counting keys

Now it's your turn!

Let's use the keyword "**code**"

What number should we use to divide and find the remainder? __4__

Use this number to fill in the Remainder line
Fill in the Key Letter line using the keyword

Message	c	r	y	p	t	o	g	r	a	p	h	y
Count	0	1	2	3	4	5	6	7	8	9	10	11
Remainder	0	1	2	3	0	1	2	3	0	1	2	3
Key Letter	c	o	d	e	c	o	d	e	c	o	d	e

The way that we get the computer to figure out a remainder is by using modulo like this:
```
remainder = 5%3
```
This will make remainder = 2 since the remainder of 5 ÷ 3 is 2

Task 4.2: Start counting

We need somewhere to count how far we are through the secret keyword.
1. Go to your code after the loop
2. Remove (or comment out) the three lines we used to test previous steps (printing key_nums, calling your function using ('a', 1) and printing the returned function value)
3. Create a variable called `count` and set it to 0

Task 4.3: How long is the key

We need to know how long the key is so we'll be able to work out when to loop around.
1. Use **len** to find the length of the `key`. Store it in a variable called `key_len`.

Hint

You can find the length of something using the `len` function like this:

```
word = "hello"
word_length = len(word)
```

Task 4.4: Looping through message letters

It's time to take a look at the message one letter at a time!
1. Create a `for` loop that loops through the letters in the message. Store the current letter in a variable called `current_letter`.

Task 4.5: Getting the right key

This step is one of the hardest parts of the workbook - read it carefully a couple of times and think it through. Ask for help if you need it - that's what the tutors are here for!

We need to use `count` and `key_len` to get the correct key to use for the current letter!
1. You can use `count % key_len` to get which position you are up to in the key. Store this in a variable called `key_position`.
2. Get the key number that corresponds to the `key_position`. You'll need to look up the `key_position` in your `keys_nums` list, store it in a variable called `current_key_num`.

Hint

You can find the item in a list at a specific index like this:

```
faves = ['books', 'butterfly', 'chocolate', 'skateboard']
print(faves[2])

>>> chocolate
```

Remember list indices start at 0

TUTOR TIPS

This is probably one of the hardest parts of the workbook both logically and in terms of python knowledge.

It has a complex combination of numbers and indexes.
We are looking for these lines:

```
# finds the position in the key number list that we will need
key_position = count % key_len

# Goes and gets the key number out of the list
current_key_num = key_nums[key_position]
```

Task 4.6: Call the function

You now have the `current_letter` and the `current_key_num`. We can encrypt!
1. Go to the spot below where you get the `current_key_num`. Make sure you are still inside the loop.
2. Call your `encrypt_letter` function for the `current_letter` and `current_key_num`.
3. Store the result and print it out. Don't forget to make it print on one line!

Hint

Here's how you can call the `sum` function and store the total in the `result` variable:
```
score1 = 10
score2 = 9
result = sum(score1,score2)
```

Task 4.7: Keep counting

Finally we are finished with this letter!
1. After you have printed the encrypted letter, add a new line inside the loop.
2. Add 1 to `count` to get ready for the next position in the message.
3. Add a temporary line to print out what the `key_position` is too. Run your code. Does it loop through the numbers like we did in 4.1 when we did it by hand?
4. Comment out the temporary print line you added in the step above once you have checked that `key_position` is looping through the numbers correctly.
5. Run your code again to print your encrypted message on one line.

☑ CHECKPOINT ☑

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

☐ Your code loops through the positions in the key

☐ Your code encrypts all the letters in the message

☐ You printed out your encrypted message on one line.

☐ Run the code!

TUTOR TIPS

The code should look like this:

```python
# <the student's name>
alphabet = 'abcdefghijklmnopqrstuvwxyz'
key_nums = []

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

name = input('What is your name? ')
print('Welcome to the Vigenere Cipher, ' + name)
message = input('What do you want to encrypt? ')
key = input('What is the encryption key? ')

for current_key_letter in key:
    current_key_num = alphabet.index(current_key_letter)
    key_nums.append(current_key_num)

count = 0
key_len = len(key)

for current_letter in message:
    key_position = count % key_len
    current_key_num = key_nums[key_position]

    result = encrypt_letter(current_letter, current_key_num)
    print(result, end='')

    count += 1
    # print(key_position)
```

Part 5: Space Ace

Remember this example, notice how we don't encrypt spaces:

t	h	i	s		i	s		a		s	e	c	r	e	t		m	e	s	s	a	g	e
↓	↓	↓	↓		↓	↓		↓		↓	↓	↓	↓	↓	↓		↓	↓	↓	↓	↓	↓	↓

g	p	n	g		p	n		g		p	n	g	p	n	g		p	n	g	p	n	g	p
6	15	13	6		15	13		6		15	13	6	15	13	6		15	13	6	15	13	6	15

↓	↓	↓	↓		↓	↓		↓		↓	↓	↓	↓	↓	↓		↓	↓	↓	↓	↓	↓	↓
z	w	v	y		x	f		g		h	r	i	g	r	z		b	r	y	h	n	m	t

Task 5.1: Testing With A Space!

1. Test your code with 2 words now instead of one, what happens?
2. What about something with an exclamation mark?

Did you get an error? Good!
At the moment, your code can encrypt any letter of the alphabet start but doesn't know what to do with spaces and other funny characters!

Task 5.2: Check for funny characters

We can see that our program tries to encrypt all characters but sometimes it breaks. We only want to encrypt letters in the alphabet variable we set-up earlier.

1. Create an **if statement** as the first line inside the message for loop you just added in Task 4.4.
2. Your **if statement** should check if the `current_letter` is in the `alphabet`.
3. Indent all your code about getting the `current_key`, calling your `encrypt_letter` function and increasing `count`, inside the if statement. This code only runs if `current_letter` is in the `alphabet`.

Hint

Why put the count inside the if statement?

Look at the example above. See that we don't move along a position in the key when we do a space. So we only increase the count for letters in the alphabet.

Spaces and other non-alphabet characters don't count!

Task 5.3: Ignore the spaces

1. Create an `else` statement to handle when characters are not in the alphabet.
2. Inside the else statement, `print` the character you are on, but don't encrypt it. Make sure they print on the same line too.
3. Run your code with a message that has spaces and characters not in the alphabet to check it works correctly.

Hint

An `else` always needs to be attached to an `if` statement

```
if raining == True:
    print ('oh no!')
else:
    print ('Yay!')
```

☑ CHECKPOINT ☑

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

☐ You only encrypt letters of the alphabet

☐ Spaces and non-alphabet characters print out unencrypted

TUTOR TIPS

The code should look like this:

```python
alphabet = 'abcdefghijklmnopqrstuvwxyz'
key_nums = []

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

name = input('What is your name? ')
print('Welcome to the Vigenere Cipher, ' + name)
message = input('What do you want to encrypt? ')
key = input('What is the encryption key? ')

for current_key_letter in key:
    current_key_num = alphabet.index(current_key_letter)
    key_nums.append(current_key_num)

count = 0
key_len = len(key)

for current_letter in message:
    if current_letter in alphabet:
        key_position = count % key_len
        current_key_num = key_nums[key_position]

        result = encrypt_letter(current_letter,
current_key_num)
        print(result, end='')
        count += 1
    else:
        print(current_letter,end='')
```

Part 6: Decrypt!

So far we've only been able to encrypt, but what if we've received a secret message and need to decrypt it?

Task 6.1: To decrypt, or not to decrypt - that is the question

1. Just like in our Caesar cipher, ask the user whether they want to encrypt or decrypt - e or d
2. Store the answer in a variable called mode
3. Have a look at the words you used in Task 1.2 and 1.3 to ask the user for the message and key word they wanted to use for the **encryption**. If you used the word 'encrypt', how could you change the words so it makes more sense to the user if they entered 'd' in response to our new question we added in (1) above. Thinking about things like this makes our code more 'user friendly'.

Task 6.2: Create the decryption key

Just like our Caesar Cipher, if we rotate by the negative of the rotation we can decrypt. We just need to do that for all of our key numbers!

1. Go to the part of your code where you loop through the letters of the keyword to find the key numbers. Go to the line after where you got the `current_key_number`.
2. On the next line add an if statement that checks what mode the user entered. We want to check if we are in decrypting mode.
3. If we are in decrypting mode we want to multiply the `current_key_number` by `-1`. Overwrite the current value with this new value.
4. Make sure your `append` **is not** inside the `if` statement
5. Try using encryption mode to encrypt a message and then decrypt mode to decrypt it (using the same keyword). Do you get back to your original message?

☑ CHECKPOINT ☑

If you can tick all of these off you can go to the Extensions:

☐ Your code now has two modes, encrypt or decrypt

☐ Your code can now decrypt an encrypted message

Students may forget to use == in if statements

The code should look like this:

```python
# <the student's name>
alphabet = 'abcdefghijklmnopqrstuvwxyz'
key_nums = []

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

name = input('What is your name? ')
print('Welcome to the Vigenere Cipher, ' + name)
mode = input('Do you want to encrypt or decrypt - e or d? ')
# there are many ways they can change the messages depending on if
# e or d entered
if mode == 'e':
    message = input('What do you want to encrypt? ')
    key = input('What is the encryption key? ')
else:
    message = input('What do you want to decrypt? ')
    key = input('What is the decryption key? ')


for current_key_letter in key:
    current_key_num = alphabet.index(current_key_letter)
    if mode == 'd':
        current_key_num *= -1
    key_nums.append(current_key_num)

count = 0
key_len = len(key)

for current_letter in message:
    if current_letter in alphabet:
        key_position = count % key_len
        current_key_num = key_nums[key_position]

        result = encrypt_letter(current_letter, current_key_num)
        print(result, end='')

        count += 1
    else:
        print(current_letter,end='')
```

Extension 7: Files!

Instead of typing out a huge long secret message, let's make python read it from a file!

Task 7.1: Getting a file

1. Take a copy of your main.py code that you've written so far - that way you will still have your original code to use if you aren't using a message from a file.
2. Get an encrypted text file from the GPN page - there will also be a key word.
3. Upload the encrypted file into Replit

Task 7.2: Read the text!

1. Remove the line where you asked the user for a `message`
2. At that same location, get python to open the file
3. Read the contents of the file
4. Store that file data in a variable called `message`

Hint

Reading from python files is a two step process.

Step 1: If we had a file called "cats.txt" (which is in the same folder as your code) we could use:

```python
f = open("cats.txt", "r")
```

Step 2: But that will just open the file. To put the text inside it in a variable, we then have to use:

```python
text_inside_cats = f.read()
```

Task 7.3: Decrypting files

1. Run your program, selecting decrypt mode and entering the key word for the file.
2. Try decrypting the other text files from the GPN page - don't forget to change your code each time to read the new file name.

1. You should also be able to use your code to encrypt a file.
2. Create your own .txt file in the same place as your code.
3. Type into the file a message that you want to encrypt.
4. Run your program, selecting encrypt mode and entering a keyword.
5. Your encrypted message should print on the screen.

TUTOR TIPS

The code should look like this:

```python
# <the student's name>
alphabet = 'abcdefghijklmnopqrstuvwxyz'
key_nums = []

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

name = input('What is your name? ')
print('Welcome to the Vigenere Cipher, ' + name)
mode = input('Do you want to encrypt or decrypt - e or d? ')
if mode == 'e':
    #message = input('What do you want to encrypt? ')
    key = input('What is the encryption key? ')
else:
    #message = input('What do you want to decrypt? ')
    key = input('What is the decryption key? ')

f = open('message.txt', 'r')
message = f.read()

for current_key_letter in key:
    current_key_num = alphabet.index(current_key_letter)
    if mode == 'decrypt':
        current_key_num *= -1
    key_nums.append(current_key_num)

count = 0
key_len = len(key)

for current_letter in message:
    if current_letter in alphabet:
        key_position = count % key_len
        current_key_num = key_nums[key_position]

        result = encrypt_letter(current_letter, current_key_num)
        print(result, end='')

        count += 1
    else:
        print(current_letter,end='')
```