Girls' Programming Network

Cryptography

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

Renee Noble
Caitlin Macleod
Alex Kaczmarek
Levie Cabuang
Pamita Mund
Courtney Ross
Feona Sims
Deanna Arora
Alex McCulloch

Testers

Jess Austin
Julia Wong

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
		

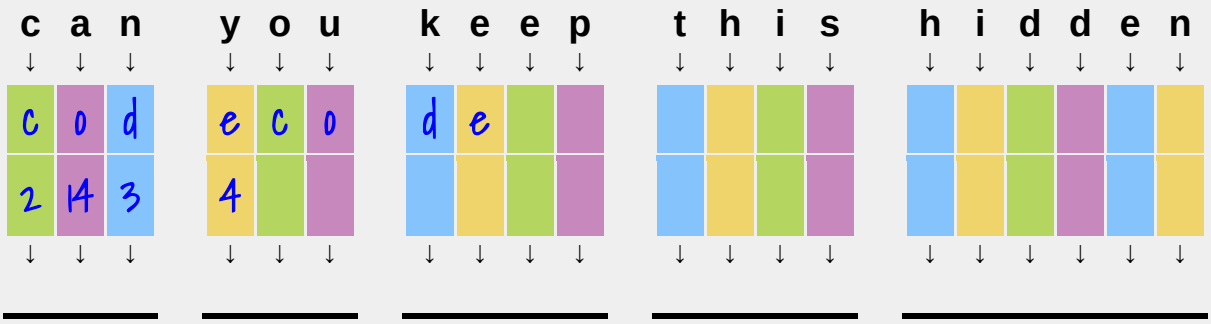
So if we want to encrypt a secret message we take turns at using the different rotations.
This is what it looks like if we encrypt the message "this is a secret message":

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

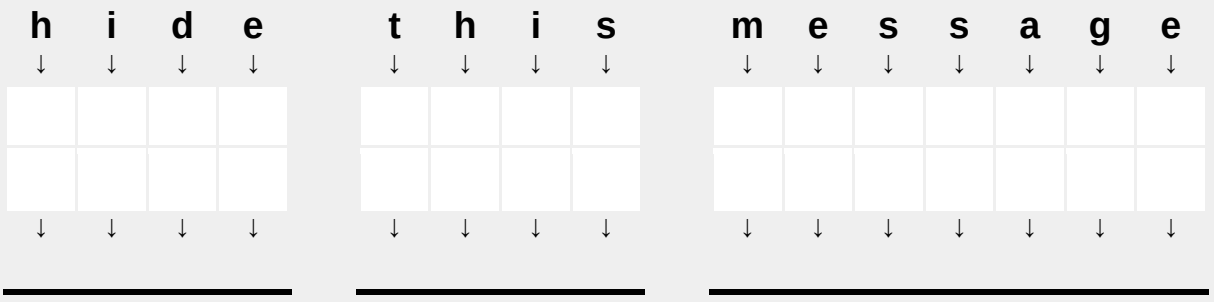
Task 0.1: Hide this message

Encrypt “can you keep this hidden” using the key **“code”**.
We’ve done some of the work for you. .



Task 0.2: Hide another

Encrypt “hide this message” using the key **“key”**.

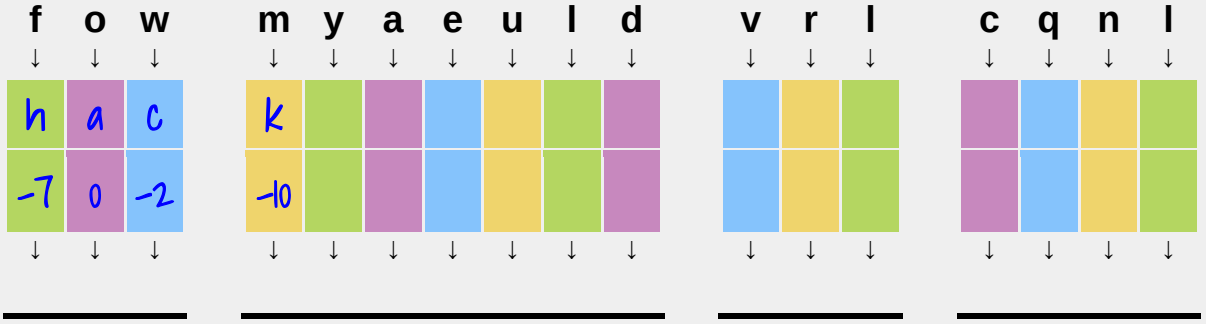


Use your cipher wheel to decrypt these vigenere encrypted messages

Just like Caesar ciphers, use the negative of each letter in the key to decrypt the message

Task 0.3: Uncover the message

Can you **decrypt** this message using the key **“hack”**.
We’ve done some of the work for you. .



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
↓	↓	↓		↓	↓	↓	↓	↓		↓	↓	↓		↓	↓	↓	↓	↓	↓
↓	↓	↓		↓	↓	↓	↓	↓		↓	↓	↓		↓	↓	↓	↓	↓	↓

☑ 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. Create a new python file 'vigenere_cipher.py'
2. 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 a file called vigenere_cipher.py
- ☐ A welcome message is printed
- ☐ You have asked for and stored a message
- ☐ You have asked for and stored a keyword
- ☐ Run your code.

★ 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
----------------------------------------
```

Part 2: Calculating Keys!

Lists

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 this by finding how far each letters in the key is from the letter A! That's it's 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						

And this one:

Keyword	**m**	**y**	**k**	**e**	**y**
Keynums					

Task 2.2: Create the alphabet

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

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

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.
You might want to store the current letter as you loop through in something 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`.

Task 2.6: Keep the keys!

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

1. Got 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

Functio

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 use 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 write 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 and print the result. 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 ☑

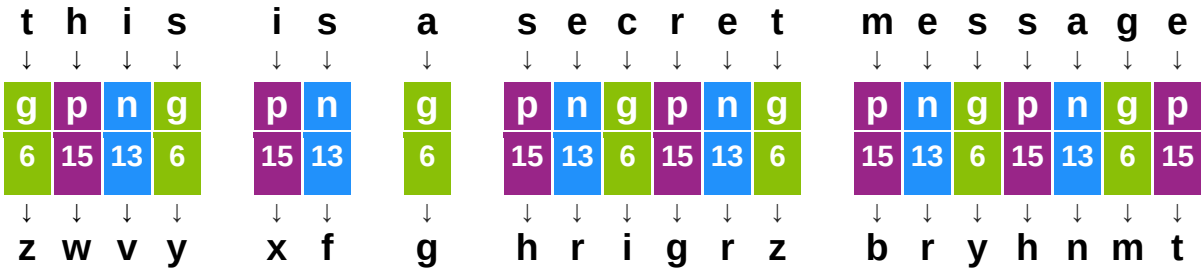
If you can tick all of these off you can go to Part 4:

- ☐ 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

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*:



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

   Look all the remainders are 0, 1 or 2!

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

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 key word "**code**"

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

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												
Key letter												

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 message.

1. Go to your code after the loop where we added things to `key_nums`.
2. 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

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`.

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.
1. Call your `encrypt_letter` function for the `current_letter` and `current_key_num`.
2. 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 `total_score` 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 `current_key_num` is too. Run your code. Does it loop through the numbers like we did in 4.1 when we did it by hand?

☑ 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!

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 on 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 that you told it at the 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 encrypted all characters and but sometimes it breaks. We only want to encrypt letters of the alphabet variable we set-up earlier.

1. Create an if statement inside your message for loop. It should be on the line directly after where you start using the for loop.
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.

Hint

Why put the count inside the if statement?

See in the example above, 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 stay on the same line too.

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

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

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 key 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

☑ 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

Extension 7: Files!

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. Get a text file either from bit.ly/gpn-crypto2 or write you own
2. Make sure you save it in the same place as your code.

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:

```
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:

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

Task 7.3: Decrypting files

1. Try you file extension on one of the secret message files from the website using the key provided.