Codebreaker

Analyse a frequency graph to crack the code, whilst learning about lists and functions

Step 1 You will make

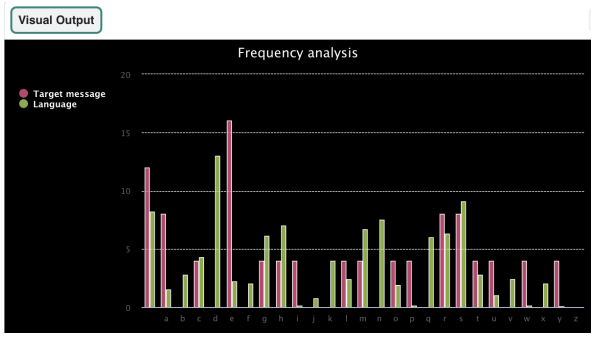
Take our survey (https://form.raspberrypi.org/f/code-editor-feedback) to help make our Code Editor better!

Analyse a graph to crack the code, whilst learning about lists and functions.

You will:

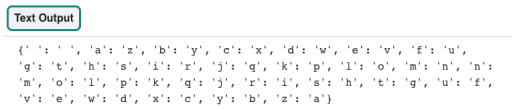
- Define a function that takes parameters and returns a value
- Use `while` and `for` loops to repeat tasks
- Create a chart to display frequency data

> The Atbash cypher is one of the world's oldest known cyphers (a cypher is a secret or disguised way of writing). Originally developed in the Hebrew language, it takes the alphabet and matches it to its reverse order to create a secret message. The name derives from the first, last, second, and second-last Hebrew letters: Aleph, Taw, Bet, and Shin.



Step 2 Encode the alphabet

To start, you will create a dictionary for your encoded letters.



Open the Codebreaker starter project (https://editor.raspberrypi.org/en/projects/codebreaker-project-starter). The Raspberry Pi code editor will open in another browser tab.

If you have a Raspberry Pi account, you can click Save to save a copy of the starter code to your library.

If you are not using the code editor in your browser, you will need to download the project files and you may need to install `pygal` before you can import it.

Installing pygal

On Windows

In the Command Prompt type the following and press the **Enter** key:

```
pip install pygal
```

Wait for the installation to complete and then continue with the project.

On a Mac

In the Terminal type the following and press the **Enter** key:

```
pip3 install pygal
```

Wait for the installation to complete and then continue with the project.

On Linux, including Raspberry Pi OS

In the Terminal type the following and press the **Enter** key:

```
pip install pygal
```

Wait for the installation to complete and then continue with the project.

Set up the alphabet list and the code dictionary

The codebreaker program starts with two data structures. The first data structure is a list of all the letters in the alphabet and the second is a `code` dictionary. To save typing time, you can create a list from a string by using the `list()` function.

Using the list function in Python

The `list()` function returns a list that you can use in your code.

Returning an empty list

If you use `list()` with no parameters in the curved brackets, it will return an empty list.

Here is an example of `list()` being used with no parameters.

```
print(list())
```

The output of this code would be:

```
[]
```

This is because the `list()` function has returned an empty list.

Converting a string into a list

You can use the `list()` function to convert a string into a list.

For example, you might have a variable that holds the vowels from the English alphabet. To convert this into a list, you would pass the `vowels` variable through the parameters (round brackets) and the function would return it as a list.

You can see this happening at line 2 in the code example below:

main.py

```
1 vowels = 'AEIOU' # The variable holds a string of vowels
2 vowel_list = list(vowels) # Create a list that holds each vowel as a separate item
3 print(vowel_list) # Display the list of vowels
```

The output of this code would be:

```
['A', 'E', 'I', 'O', 'U']
```

Find the `# Set up data structures` comment in the program, then use the `list()` function to create a list of letters from the `alphabet`. Next, initialise the `code` dictionary so that you can populate it in a later step.

The `alphabet` list contains spaces at the beginning and end to preserve the spaces in the message. Strong encryption would not do this, as it makes the message easier to decode. The spaces have been kept in for this project to make the messages easier to read.

main.py

```
5 # Set up data structures
6 alphabet = list(' abcdefghijklmnopqrstuvwxyz ')  # List from a string
7 code = {}
```

Create a new list that reverses the alphabet

You need to create a new list that holds the alphabet, but backwards. You can use the `list()` function again to help with this. You can also use the `reversed()` function to reverse an existing list.

Find the `# Create the atbash code by reversing the alphabet` comment then define a new function called `create_code`. Next, create a list that holds the reverse of the `alphabet` list.

main.py - create_code()

```
# Create the atbash code by reversing the alphabet
def create_code():
    backwards = list(reversed(alphabet))  # Reverses a list
```

Encode the alphabet

> Encoding is when you convert data from one form to another. In an atbash cypher for example, the letter 'e' would be encoded as a 'v'.

You now have two lists. One contains the alphabet written forwards, the other with the alphabet backwards. You are now going to use these two lists to populate a dictionary. The key will store the alphabet written forwards and the paired value will store the alphabet backwards.

The code dictionary is really important because you can use it to match each letter from your message using the key, with its encoded paired value.

Within your `create_code` function, populate the `code` dictionary with data from the two lists. Use a `for` loop to get the length of the `alphabet` list and populate the dictionary with the data.

`len()` is a function that you can use to find out the length of an object, such as a list. It is used here to iterate a `for` loop, as many times as there are characters in the `alphabet` list — its length.

main.py - create_code()

```
def create_code():
    backwards = list(reversed(alphabet))  # Reverses a list

    for i in range(len(alphabet)):  # Gets the length of a list
        code[alphabet[i]] = backwards[i]  # Populate the code dictionary with a letter of the alphabet and its encoded letter
```

Creating a `main()` function is useful to call all of the required functions when your program first starts.

Find the `# Start up` comment and define a `main()` function to call your `code()` function. Next, call the `main()` function in the main body of your code.

main.py - main()

```
# Start up
def main():
    create_code()

main()
```

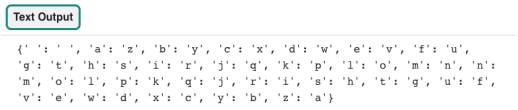
Test and debug

To test that your `code` dictionary has populated correctly, you can `print` the dictionary in full. Under your `for` loop in the `create_code` function, add a `print` function to display the contents.

main.py - create_code()

```
def create_code():
    backwards = list(reversed(alphabet))

    for i in range(len(alphabet)):  #  Gets length of a list
        code[alphabet[i]] = backwards[i]  #  Populates the code dictionary with a letter of the alphabet and its encoded letter

        print(code)
```

Test: Run your code to see if the `code` dictionary displays correctly. You should see a pattern starting with the letter `a` paired with `z` and the letter `b` being paired with `y`.



Debug: There are no error messages but your code dictionary is not displaying on the screen:

- Make sure that `print(code)` is indented correctly within the `create_code` function
- Check that you have called the `create_code()` and the `main()` function correctly

Debug: If you see a message about `code` not being defined, make sure that you have initialised the `code` dictionary.

Debug: If you see a message about an indentation error:

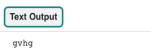
- Check that you have indented all of your code correctly
- Look back at the sample code on this page to help you check

In the next step, you will encode a message with the help of your `code` dictionary.

Save your project

Step 3 Encode a message

In this step, you will create a function that can take your text, flip it and reverse it with your atbash cypher list, and return it as an encoded message.



Comment out the print statement used for testing on line 17 by placing a hashtag at the beginning of the line:

main.py - create_code()

```
14  for i in range(len(alphabet)):  # Gets length of a list
15          code[alphabet[i]] = backwards[i]  # Populates the code dictionary with a letter of the alphabet and its
16  encoded letter
17
    # print(code)
```

Set up your atbash function

You will now add your new function that will encode some text using the atbash cypher.

Find the comment that says `# Encode/decode a piece of text – atbash is symmetrical`. Underneath the comment, define a function called `atbash`, with the parameter `text`. Parameters allow you to pass values into functions that can be used within that function.

main.py - atbash()

```
26  # Encode/decode a piece of text – atbash is symmetrical
27  def atbash(text):
```

Press **`Enter`**. You should see the next line indented.

Parameters in functions

When you call or define a function, you always add curved brackets after its name. Just like this example below:

```
def menu(): # Defines a function
  print('Hello')

menu() # Calls a function
```

Those brackets can be used to pass data into a function from another section of your code. This data can then be used by the function to carry out some tasks.

The labels inside the brackets are called parameters. A function can have multiple parameters depending on the purpose of the function.

The example function below has two parameters, which are `name` and `player_id`.

```
def menu(name, player_id):
  print(f'Hello {name}, your player ID is {player_id}')
```

Another part of your code might ask for a player's name or generate a player ID. These can then be passed into the `menu()` function to be used to display a welcome message.

Values that are passed into a function are called arguments.

In the example code below you can see the `menu()` function being defined. You can also see the player's name and ID being passed into the function as arguments.

```
1  def menu(name, player_id):
2
3    print(f'Hello {name}, your player ID is {player_id}') # The function uses the values
4
5  username = 'Hayden'
6  id = 3215
7
8  menu(username, id) # 'Hayden' and '3215' are passed as arguments into the function
```

Convert text to lower case

First your function needs to convert the `text` to lower case. A new variable called `output` then needs to be created to hold the encoded message.

Beneath the line of code where you have defined the `atbash()` function, type:

main.py - atbash()

```
26 # Encode/decode a piece of text – atbash is symmetrical
27 def atbash(text):
28     text = text.lower()  # Converts text to lower case
29     output = ''
```

Encode your text

The next part of your code will encode the `text` that has been passed into the function. A `for` loop is used to go through each letter in the `text` and convert it to an encoded letter using the `code` dictionary. Finally, it will return the encoded message.

Leave a blank line under the last code you entered (make sure you keep the indent), then type:

main.py - atbash()

```
26 # Encode/decode a piece of text – atbash is symmetrical
27 def atbash(text):
28     text = text.lower()  # Converts text to lower case
29     output = ''
30
31     for letter in text:
32         if letter in code:
33             output += code[letter]  # Populates output with the encoded/decoded message using the dictionary
34
35     return output  # Return the encoded/decoded message
```

Test and debug

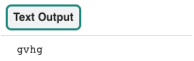
Now that you have a function that will encode text, you need to run it to make sure it works. Find your `main()` function and add in a function call to run the `atbash()` function.

The 'Test' string is passed into the function so that it can be encoded.

main.py - main()

```
# Start up
def main():
    create_code()
    print(atbash('Test'))
```

Test: Run your code to see if the test message displays correctly. You should see the console output `gvhg`.



Debug: If you see a message about an indentation error:

- Check that you have indented all of your code correctly
- Look back at the sample code on this page to help you check

Comment out your `print(atbash('Test'))` line of code now that you have finished testing.

main.py - main()

```
# Start up
def main():
    create_code()
    # print(atbash('Test'))
```

In the next step you will encode a message with the help of your `code` dictionary.

Save your project

Step 4 Create a menu

Now you are going to create a menu system for your user to make choices about what they would like to do.

Find the comment in your code that says `# Create a text-based menu system` and begin by defining a function called `menu()`:

main.py - menu()

```
# Create a text-based menu system
def menu():
```

Your menu needs a loop that continually asks the user what they would like to do until they have entered a valid choice. To get this started, you will create a variable called `choice` and set it to `''`. This will allow the while loop to run its first loop.

Create a new variable called `choice` and set the value to `''`:

main.py - menu()

```
# Create a text-based menu system
def menu():
    choice = ''  # Start with a wrong answer for choice.
```

Use a `while` loop to get user input

Now that you have set `choice` to a wrong answer, you want to create a loop that will only break if an `input` that matches a right answer is given. You want a while loop that runs as long as your answer DOES NOT match one you have defined.

You can use a while loop to run a piece of code while a condition is True. In this instance, as long as the user does not choose `c` or `f`, the loop will continue to run. Enter the code that will set the conditions for a while loop and prompt the user for input:

main.py - menu()

```
def menu():
    choice = ''  # Start with a wrong answer for choice

    while choice != 'c' and choice != 'f':  # Keep asking the user for the right answer
        choice = input('Please enter c to encode/decode text, or f to perform frequency analysis: ')
```

Once the user has given a correct answer, the loop will end. Next create an `if` statement that will run your `atbash` function if the user enters `c`.

You will decide what happens when a user enters `f` in a later step.

Underneath the last line (making sure you still have an indent!), type:

main.py - menu()

```
def menu():
    choice = ''  # Start with a wrong answer for choice

    while choice != 'c' and choice != 'f':  # Keep asking the user for the right answer
        choice = input('Please enter c to encode/decode text, or f to perform frequency analysis: ')

    if choice == 'c':
        print('Running your message through the cypher…')
        message = 'my secret message'
        code = atbash(message)
        print(code)
```

Change the string that says `'my secret message'` to anything you like. This string is the message that will be encoded and decoded.

main.py - menu()

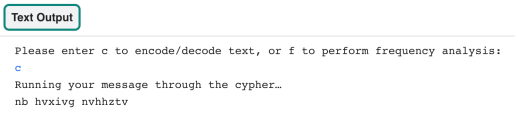
```
def menu():
    choice = ''  # Start with a wrong answer for choice.

    while choice != 'c' and choice != 'f':  # Keep asking the user for the right answer
        choice = input('Please enter c to encode/decode text, or f to perform frequency analysis: ')

    if choice == 'c':
        print('Running your message through the cypher…')
        message = 'my secret message'
        code = atbash(message)
        print(code)
```

At the end of your `main()` function, type `menu()` to call the `menu` function when the program runs:

main.py - main()

```
# Start up
def main():
    create_code()
    # print(atbash('Test'))
    menu()
```

Test: Run your code. Type `c` and press **`Enter`** to encode your message string!



Debug: If you see a message about an indentation error:

- Check that you have indented all of your code correctly
- Look back at the sample code on this page to help you check

Debug: If you see the error message `c is not defined` when you run your code, check that you have used apostrophes (‘’) around your c in the condition `choice != 'c'`.

Debug: If nothing happens when you press `c`, check that you have correctly spelled `message`.

In the next step you will use your `atbash()` function to encode the contents of a text file.

Save your project

Step 5 Encode text from a file

It's time to encode a message from a text file.

> Loading the text from a file is more efficient than typing or pasting a large string into a program. There is less opportunity to 'break' your code when changing a single target file name, than when copy and pasting large blocks of text each time.

Find the `# Fetch and return text from a file` comment then define a `get_text()` function. This function has one parameter called `filename`. Use the `filename` to open the file and read it into the `text` variable, then return the `text` variable.

main.py - get_text()

```
# Fetch and return text from a file
def get_text(filename):
    with open(filename) as f:
        text = f.read().replace('\n','')  # Need to strip the newline characters

    return text
```

The `menu()` function needs to encode a secret message from a text file. Replace your secret message with the `get_text()` function call. Enter the name of the file `input.txt` as an argument.

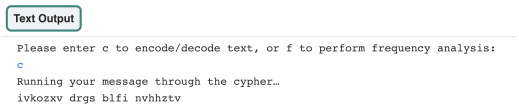
main.py - menu()

```
if choice == 'c':
    print('Running your message through the cypher…')
    message = get_text('input.txt')  # Take input from a file
    code = atbash(message)
    print(code)
```

You can now add your own secret message to the `input.txt` file.

Find the `input.txt` file in your code editor to access the contents of the text file. Delete the `replace with your message` text and enter your own secret message.

Test: Run your code to see if it displays your encoded message after entering the letter 'c' when prompted.



Debug: Your encoded message doesn't look exactly like the message in the screenshot:

- This is normal. This is the encoded message for the text `replace with your message`. Your message will be different.

Debug: You see an error message that says `TypeError: get_text() takes exactly 1 arguments`:

- Check that you have entered `input.txt` inside the round brackets on line 57

Debug: You see an `Indentation error` message:

- Check that you have correctly indented all of your new code. Revisit the tasks above to check.

Decode the message

The atbash cypher encodes a message using the reverse letters of the alphabet. This means that exactly the same code can be used to decode the message. You can test this by taking your encoded message, copying and pasting it into your `input.txt` file and running the code again.

Run your code so that it displays your encoded message. Select the encoded message and copy it. Go back to `input.txt` and delete your message. Next, paste your new message into the empty file.

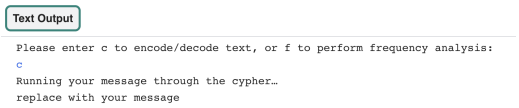
Remember that your code converts any text to lower case, so you will see your message in lower-case letters.

Copying and pasting

You can copy text and paste a copy in another place.

1. Select the text you want to copy by dragging your mouse over it while holding down the left button.
2. Copy the text by using your browser's menu — choose Edit > Copy. You can also use a keyboard shortcut — **`Ctrl`**+**`C`** on Windows or Linux systems, or **`Cmd`**+**`C`** on a Mac.
3. Move your text cursor (the flashing line that shows where you are typing) to where you want to place a copy of the text.
4. Paste the text by using your browser's menu — choose Edit > Paste. You can also use a keyboard shortcut — **`Ctrl`**+**`V`** on Windows or Linux systems, or **`Cmd`**+**`V`** on a Mac.

Test: Run your code again and press 'c' when prompted. It will display the decoded version of your original message.



Debug: It still displays the encoded message:

- Make sure that you have copy and pasted the encoded message into `input.txt`

In the next step you will write the code to analyse the frequency of letters in your text file.

Save your project

Step 6 Create a frequency analyser

In this step, you will code a frequency analyser function to work out how often each letter of the alphabet appears in your text.

```
Please enter c to encode/decode text, or f to perform frequency analysis:
f
Analysing message…
{' ': 12.0, 'a': 8.0, 'b': 0.0, 'c': 4.0, 'd': 0.0, 'e': 16.0, 'f': 0.0, 'g': 4.0,
'h': 4.0, 'i': 4.0, 'j': 0.0, 'k': 0.0, 'l': 4.0, 'm': 4.0, 'n': 0.0, 'o': 4.0, 'p':
4.0, 'q': 0.0, 'r': 8.0, 's': 8.0, 't': 4.0, 'u': 4.0, 'v': 0.0, 'w': 4.0, 'x': 0.0,
'y': 4.0, 'z': 0.0}
```

> Frequency analysis measures how often something appears so you can look for patterns in that data. It is possible to decode monoalphabetic cyphers (if you know the language the message is in) by looking at how often each letter appears and matching it to the most commonly used letters (http://letterfrequency.org/letter-frequency-by-language/) in that language. This will be explained in further detail later.

You now need to create a function that will take your text and convert it all to one case (to avoid confusion), count the number of times each letter in the message appears, then convert that number into a percentage of the whole so you can compare it to the frequency of letters in English.

Beneath the comment that reads `# Calculate the frequency of all letters in a piece of text`, define a function called `frequency`, and have the first thing it does be to convert your message to lower case and make it a list:

main.py - frequency()

```
18 # Calculate the frequency of all letters in a piece of text
19 def frequency(text):
20     text = list(text.lower())  # Convert the message to lower case and make it a list
```

Create a dictionary called `freq` and for every `letter` in the list `alphabet` assign a value of `0`. Make sure you keep the indentation and type:

main.py - frequency()

```
18 # Calculate the frequency of all letters in a piece of text
19 def frequency(text):
20     text = list(text.lower())  # Convert the message to lower case and make it a list
21
22     freq = {}  # Create a dictionary of every letter, with a count of 0
23     for letter in alphabet:
24         freq[letter] = 0
```

The next thing you need your function to do is to count the all letters in the message. Create a variable called `total_letters` and assign the length of the text to that variable.

Make sure you keep the indentation as shown in this code.

main.py - frequency()

```
# Calculate the frequency of all letters in a piece of text
def frequency(text):
    text = list(text.lower())  # Convert the message to lower case and make it a list

    freq = {}  # Create a dictionary of every letter, with a count of 0
    for letter in alphabet:
        freq[letter] = 0

    total_letters = len(text)  # Count the letters in the message
```

Once you know how long the message is, you can begin counting the letters in it to determine how often they appear.

Create a `for` loop to count every time each letter appears in the text. Leave a blank line at the end of your script, make sure you keep the indentation, and add:

main.py - frequency()

```
# Calculate the frequency of all letters in a piece of text
def frequency(text):
    text = list(text.lower())  # Convert the message to lower case and make it a list

    freq = {}  # Create a dictionary of every letter, with a count of 0
    for letter in alphabet:
        freq[letter] = 0

    total_letters = len(text)  # Count the letters in the message

    for letter in text:
        if letter in freq:
            freq[letter] += 1
```

Count the letters

main.py

```
for letter in text:
    if letter in freq:
        freq[letter] += 1
```

This section of code looks at each of the letters in your message `text`, and if the letter appears in your frequency list, it adds `1` to that letter's value. The more times a letter appears, the higher that value will be. Once you know how often each letter appears, you can then convert from this count to a percentage of the whole message (since you know its length). Any characters that are not in the dictionary — such as punctuation — will be ignored, and won't appear in the message.

Create a `loop` that converts the number of times the letters appear into a percentage of the whole message.

main.py - frequency()

```
# Calculate the frequency of all letters in a piece of text
def frequency(text):
    text = list(text.lower())  # Convert the message to lower case and make it a list

    freq = {}  # Create a dictionary of every letter, with a count of 0
    for letter in alphabet:
        freq[letter] = 0

    total_letters = len(text)  # Count the letters in the message

    for letter in text:
        if letter in freq:
            freq[letter] += 1

    for letter in freq:
        freq[letter] = freq[letter] / total_letters * 100  # Convert from counts to percentages
```

Return the frequency dictionary so it can be used elsewhere in your code. Leave a blank line and type:

main.py - frequency()

```
# Calculate the frequency of all letters in a piece of text
def frequency(text):
    text = list(text.lower())  # Convert the message to lower case and make it a list

    freq = {}  # Create a dictionary of every letter, with a count of 0
    for letter in alphabet:
        freq[letter] = 0

    total_letters = len(text)  # Count the letters in the message

    for letter in text:
        if letter in freq:
            freq[letter] += 1

    for letter in freq:
        freq[letter] = freq[letter] / total_letters * 100  # Convert from counts to percentages

    return freq
```

Extend the menu to include 'f'

Now that you have a function that can calculate the frequency of letters in your message, you need to link it to your user menu. Right now, the user can only choose the letter 'c' to encode or decode a message. If they type the letter 'f', nothing happens. You are now going to add the option 'f' to analyse the letter frequency of your message by calling your new function.

Underneath your first `if` statement asking the user to select 'c', you need to add an `elif` statement so the user can choose the option to analyse and print the letter frequency by pressing 'f'.

Leave a blank line after the `if` statement and, on line 72, type:

main.py - menu()

```
63  while choice != 'c' and choice != 'f':  # Keep asking the user for the right answer
64      choice = input('Please enter c to encode/decode text, or f to perform frequency analysis: ')
65
66  if choice == 'c':
67      print('Running your message through the cypher…')
68      message = get_text('input.txt')  # Take input from a file
69      code = atbash(message)
70      print(code)
71
72  elif choice == 'f':
73      print('Analysing message…')
74      message = get_text('input.txt')
75      message_freq = frequency(message)
76      print(message_freq)
```

Save and run your code. Choose 'f' at the prompt and you should see a readout of the letter frequency of your message in the console. The values you see from your message will be different from the values shown here:

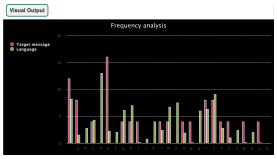
```
Please enter c to encode/decode text, or f to perform frequency analysis:
f
Analysing message…
{' ': 12.0, 'a': 8.0, 'b': 0.0, 'c': 4.0, 'd': 0.0, 'e': 16.0, 'f': 0.0, 'g': 4.0,
'h': 4.0, 'i': 4.0, 'j': 0.0, 'k': 0.0, 'l': 4.0, 'm': 4.0, 'n': 0.0, 'o': 4.0, 'p':
4.0, 'q': 0.0, 'r': 8.0, 's': 8.0, 't': 4.0, 'u': 4.0, 'v': 0.0, 'w': 4.0, 'x': 0.0,
'y': 4.0, 'z': 0.0}
```

In the next step, you will display the frequency analysis data in a cool looking chart!

Save your project

Step 7 Analyse the frequency

Use a bar chart to analyse the frequency of letters in an encoded message.



> In all languages, each letter in its alphabet has a 'personality' or set of traits when used in that language. One of the most obvious traits a letter has in any language is how often it appears. Frequency analysis is the method of breaking codes by looking at how often letters are used in the language of the code, and comparing that to how often encoded letters show up in a message. In English, the letter e is the most commonly used letter (it shows up 12.8% of the time), followed by t (at 9.1%). The least often used letter is z.

The `print(message_freq)` line of code is no longer needed. Add a `#` to the beginning of it so that Python ignores it when the code is run.

main.py - menu()

```
72  elif choice == 'f':
73      print('Analysing message…')
74      message = get_text('input.txt')  # Take input from the same file. We have a 'longer.txt' or similar containing
75  cyphertext we know to perform reasonably well for frequency analysis
76      message_freq = frequency(message)  # Get the frequency of the letters in the message, as %
        # print(message_freq)
```

Make the frequency chart function

Find the `# Make frequency chart` comment and create a new function called `make_chart()`. This function needs two parameters called `text` and `language`. The frequency chart will be a bar chart with the title `Frequency analysis` and with x-axis labels using the keys from the `freq` dictionary.

The `freq` dictionary values will be passed into the function when it is called later in the code, via the `text` parameter.

main.py - make_chart()

```
36  # Make frequency chart
37  def make_chart(text, language):
38      chart = Bar(width=800, height=400, title='Frequency analysis', x_labels = list(text.keys()))
```

Label the chart with the frequency of letters in the encoded message and the known letter frequency of the language the message is in. This data has been passed into the function via the `text` and `language` parameters.

main.py - make_chart()

```
# Make frequency chart
def make_chart(text, language):
    chart = Bar(width=800, height=400, title='Frequency analysis', x_labels = list(text.keys()))
    chart.add('Target message', list(text.values()))  # Label the frequency data for the encoded message
    chart.add('Language', list(language.values()))  # Label the frequency data for the language
```

Render the chart so that it will display when the function is called.

main.py - make_chart()

```
# Make frequency chart
def make_chart(text, language):
    chart = Bar(width=800, height=400, title='Frequency analysis', x_labels = list(text.keys()))
    chart.add('Target message', list(text.values()))  # Label the frequency data for the encoded message
    chart.add('Language', list(language.values()))  # Label the frequency data for the language

    chart.render()
```

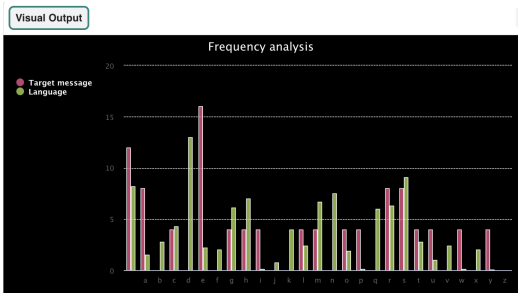
Call the frequency chart function

Find your `elif` in the `menu()` function. Add a line of code that will import the `english` frequency dictionary from the `frequency.py` file. Add another line of code that will call the `make_chart` function to draw the chart.

main.py - menu()

```
elif choice == 'f':
    print('Analysing message…')
    message = get_text('input.txt')  # Take input from the same file. We have a 'longer.txt' or similar containing
cyphertext we know to perform reasonably well for frequency analysis
    message_freq = frequency(message)  # Get the frequency of the letters in the message, as %
    # print(message_freq)
    lang_freq = english  # Import the English frequency dictionary
    make_chart(message_freq, lang_freq)  # Call the function to make a chart
```

Test: Run your code to display the frequency analysis bar chart.



Debug: Your chart doesn't look exactly the same as the one displayed in the image above:

- This is normal. Your chart will display the frequency data for the secret message that you have entered in `input.txt`.

Debug: You see the following error message `NameError: name 'lang_freq' is not defined`:

- Check that you added the line of code `lang_freq = english` before the `make_chart()` function call.

Debug: You see an `Indentation error` message:

- Check that you have correctly indented all of your new code. Revisit the tasks above to check.

Analyse the frequency chart

The chart that has been produced shows the frequency of letters in the English language, labelled as Language. You can see that the letter e is the most frequently used letter in the English language because it has the highest bar for all of the language values.

The frequency chart also lists the frequency of letters in your encoded message, labelled as Target message. This includes the spaces in your message, which can be seen in the last bar on the right. To work out what encoding has been used for this message, you can compare the bars showing the English language with the bars on the encoded message. The highest bar (ignoring the spaces) in the encoded message text will most likely be an e. The second highest letter will most likely be a t as this is the next most popular letter.

Codebreakers can use the frequency of letters to work out the type of encoding that has been used on the message. They can use trial and error to predict what a letter might represent using the chart as a guide.

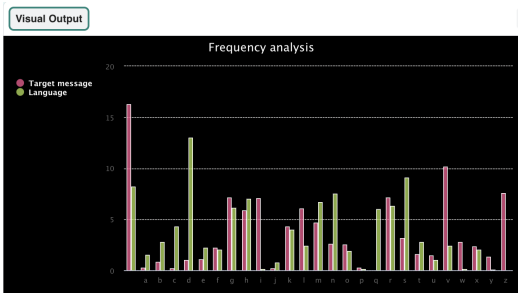
Your secret message is quite small, which makes it tricky to analyse using a frequency chart. Change your code so that it analyses the message in `longer.txt` instead.

Change `input.txt` to `longer.txt`.

main.py - menu()

```
75  elif choice == 'f':
76      print('Analysing message…')
77      message = get_text('longer.txt')
78      message_freq = frequency(message)  # Get the frequency of the letters in the message, as %
```

Analyse the frequency chart by looking at the Language values and the Target message values. Notice how the highest bar for Language is e and the highest bar for Target message is v. This is because with the Atbash cypher, the letter e is encoded with the letter v.



Save your project

Upgrade your project

In this step, discover ways to upgrade your project!

Get inspiration

Have a look at these example projects to spark some ideas of your own!

This project has an extra menu option to enter and encode your own short text:

Custom short message encoder:

You can find the Custom short message encoder project here (https://editor.raspberrypi.org/en/projects/short-message-encoder)

This project can complete frequency analysis in English, French, or Spanish:

Three language atbash encoder:

You can find the Three language atbash encoder project here (https://editor.raspberrypi.org/en/projects/three-language-encoder)

Change the starting letter of the alphabet to create a different cypher. You can do this by changing the original `alphabet` list.

Add another language to your frequency analysis. You will need to investigate the frequency of letters used and place them into a dictionary in the `frequency.py` program.

Add another menu item to allow users to change the language.

If possible, swap your cyphertext with a partner and see if they can crack your code using their frequency analyser.

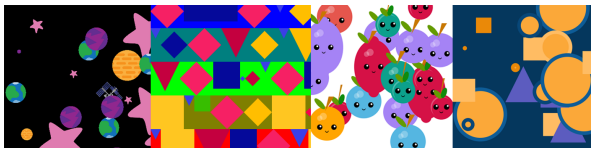
Completed project

You can view the completed project here (https://editor.raspberrypi.org/en/projects/codebreaker-project-example).

Save your project

What next?

If you are following the More Python (https://projects.raspberrypi.org/en/pathways/more-python) pathway, you can move on to the Encoded art (https://projects.raspberrypi.org/en/projects/encoded-art/0) project. In this project, you will make amazing artwork using data!



If you want to have more fun exploring Python, then you could try out any of these projects (https://projects.raspberrypi.org/en/projects?software%5B%5D=python&curriculum%5B%5D=%202).
