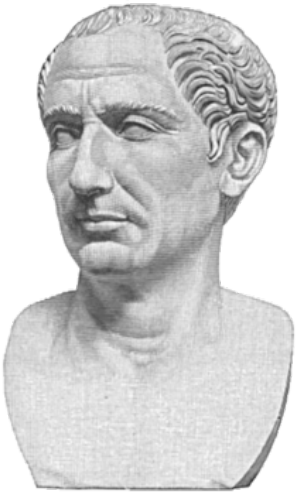
In this assignment you will define your own functions (Chapter 6), you will use the `open()` function to read (sec. 7.2) and write (sec. (sec. 7.3) to text files. Read more about `chr()` and `ord()` in section 2.7, **`for`**-loops in section 3.5, and `list`s in section 2.9.

AE1205 Assignment 3: Caesar cipher

This large assignment consists of a number of smaller assignments. With the results of these parts you will make your first decipher program, i.e. your first step to join the secret service!

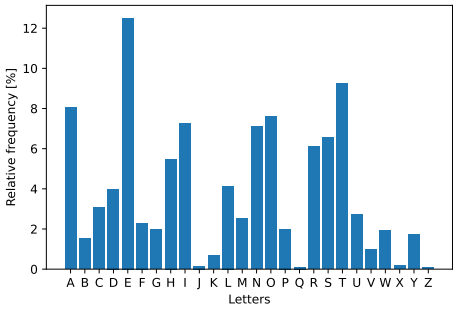


Bust of Julius Caesar, after whom the cipher was named.

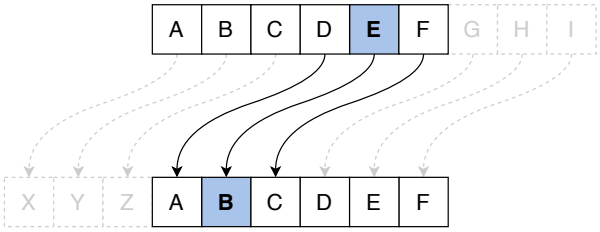
Device with concentric, rotating disks that can be used to encrypt and decrypt code.

Introduction and first steps

Download the files **`secret.zip`** and **`ch-freq-en.txt`** from BrightSpace. As you can see the first zip-file contains a set of secret, encoded messages. They have been encoded with the Caesar cipher, which is a cyclical shift in the alphabet. We happen to know that the original texts were in English and that English has a distribution of alphabetic characters as given in **`ch-freq-en.txt`**, also shown in the figure on the right side.



We do not know what the secret messages are. Each file uses a different unknown alphabetical shift. This shift is cyclical: so for instance “Z” shifted +2 will become a “B”, see the illustration below. In a number of steps you will decode these files. Because we know that the original texts are all in English, we can assume that the letter frequency in the original texts will closely match the reference values for the English language. This feature can be used to find the best fitting alphabetical shift to decode the text.



Example of decoding a text encoded with a Caesar shift of +3.

Characters are stored in computer memory as integers. For this, each character has a so-called ASCII code (a table with the first 128 ASCII codes can be found here). In Python, you can convert between a character and its numerical ASCII code by using the built-in `chr()` and `ord()` functions (see section 2.7). For example, printing the ASCII numbers of each character in the string `'Hello'` is done as follows:

```
string = 'Hello'
for ch in string:
    print('Character', ch, 'is item', ord(ch), 'in the ASCII table')
```

which will print:

```
Character H is item 72 in the ASCII table
Character e is item 101 in the ASCII table
Character l is item 108 in the ASCII table
Character l is item 108 in the ASCII table
Character o is item 111 in the ASCII table
```

Similarly, the other way around you can do:

```
print('The knights who say', chr(78) + chr(105))
```

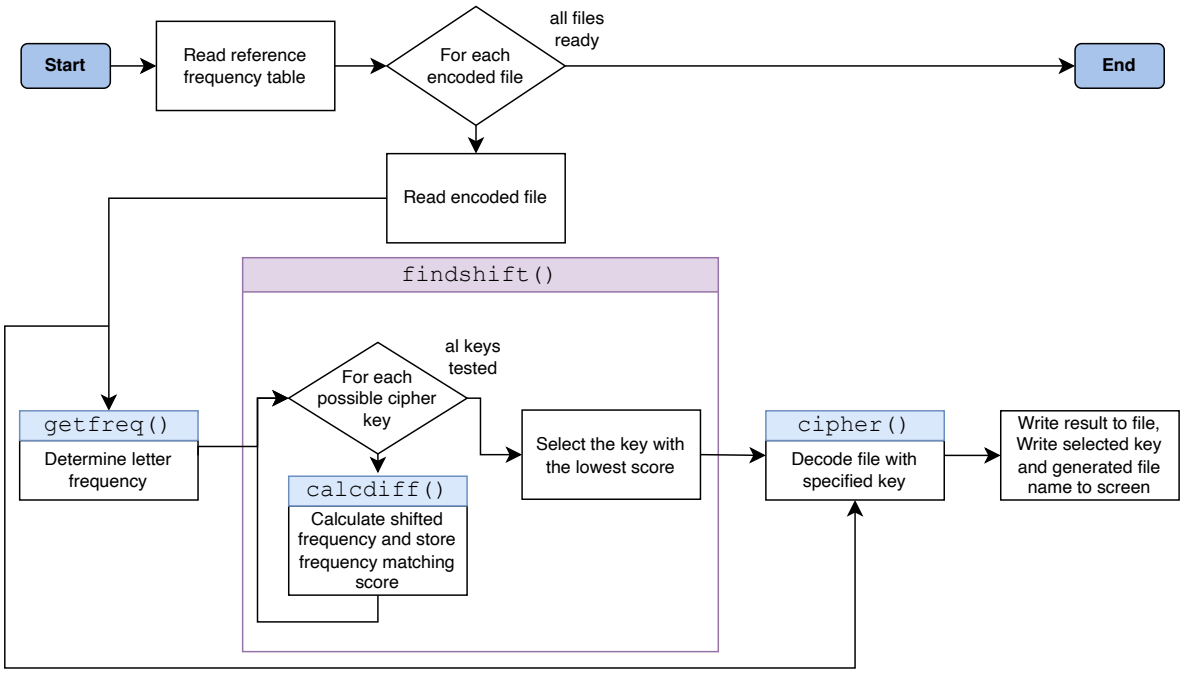
which will print:

```
The knights who say Ni
```

You will use both these functions extensively in this assignment.

Logic of the program

To successfully decode the encoded secret texts provided in the zip file, you'll need to perform a number of steps: read encoded files, determine their letter frequencies, and compare these letter frequencies for different cipher shifts (1-25) to the reference frequencies you have of the English language. With the most likely cipher your program should then decipher the text and store the result in a new file. Below is a diagram that visualises a possible implementation structure of these steps.



In this structure, your program would contain the following elements:

- A block of code to load the English letter frequencies from file, and store them in a list called `freqs_en`.
- A block of code that loops over the secret files, reads them into a string, finds the appropriate deciphering key, deciphers the file, and stores the results in a new file. Within this loop your code will call:
  - A function `getfreq()` to determine the letter frequencies for a given piece of text.
  - A function `findshift()` that, for some secret encoded text, finds the most likely Caesar cipher shift that can be used to decode the secret text.
  - A function `calcdiff()` to calculate the summed absolute frequency difference between two letter frequency tables. This function is called within `findshift()`.
  - A function `cipher()` that can be used to encode or decode a string of text, using a cipher shift passed to the function.

Loading the English letter frequency table

When you have a look at the file `ch-freq-en.txt`, you will see that it contains two columns of 26 rows. For each row, the first column contains the letter, and the second column its corresponding letter frequency for the English language. In your program, create a piece of code that reads this file, and for each line, stores the letter frequency as a (floating-point) number in a list called `freqs_en`. Note that the rows in the file are not ordered alphabetically! In can therefore be useful to start out with a list containing pre-defined default values:

```
# Start with a list of zero frequencies for each letter in the alphabet
freqs_en = 26 * [0]
```

Once you've loaded the 26 values into the `freqs_en` list, don't forget to normalise the values, so that the sum of the values in `freqs_en` adds up to 1.0!

The `getfreq()` function

To be able to compare letter frequencies of encrypted texts, shifted with candidate cipher keys, you need a function that is able to determine the letter frequencies of a given piece of text. To this end, create a function called `getfreq()`, which looks as follows:

```
def getfreq(text):
    ''' Determine letter frequencies of passed text.

        Arguments:
        - text: The text to determine the frequencies for

        Returns:
        - freqs: List with 26 letter frequencies. sum(freqs) == 1.0
    '''
    # Your code to determine the letter frequencies here
    ...

    # Return the final frequency table
    return freqs
```

This function should count all letters in the input string `text`, and use those counts to determine the *normalised* letter frequencies. Similar to the first step, where you loaded the English letter frequencies from file, in this function it also makes sense to start out with a list containing 26 zeros as a starting point. Also, don't forget to consider both lowercase and uppercase letters!

To test this function, use the following example:

Test example

```
>>>getfreq(”Come see the violence inherent in the system!! Help Help
 ↪  I'm being oppressed!!”)[:5]
[0.0, 0.01639344262295082, 0.03278688524590164,
 0.01639344262295082, 0.2459016393442623]
```

The `calcdiff()` function

The most likely deciphering key can be found by determining the key for which the corresponding letter frequency table corresponds the best to the reference table containing the letter frequencies of the English language. This is the case when the *sum of absolute differences* between the two tables is smallest. For tables A and B, this difference is given by:

$$\text{difference} = \sum_{i=0}^{i=25} \|A[i] - B[i]\|$$

Implement this equation in a function `calcdiff()`. The function should look like this:

```
def calcdiff(table1, table2):
    ''' Calculate summed absolute frequency difference between tables

        Arguments:
        - table1: The first table
        - table2: The second table

        Returns:
        - difference: The calculated summed absolute difference.
    '''
    # Your summed absolute difference calculation
    ...
    return difference
```

You can test your `getfreq()` function with the following example:

Test example

```
>>>freqs = getfreq(”Come see the violence inherent in the system!! Help
 ↪  Help I'm being oppressed!!”)
>>>calcdiff(freqs, freqs_en)
0.5453745374537453
```

The `findshift()` function

For a given piece of text encrypted with a Caesar cipher, the correct deciphering key can be found by systematically testing all cipher shifts between one and twenty-five, and selecting the cipher shift that yields the letter frequency that is closest to the reference letter frequency corresponding to the English language.

As described above, the `findshift()` function should therefore, for each possible cipher shift key (1-25):

1. Determine the shifted letter frequencies
2. Calculate the summed absolute frequency difference with the English frequencies using the previously created `calcdiff()` function.
3. Store the summed difference

Finally, the function should select the cipher shift key corresponding to the smallest summed difference as the correct deciphering key, and return it.

There are two ways to perform the first step of determining the shifted letter frequencies. The first is to generate the shifted text for each of the potential cipher keys using the `cipher()` function, and determine the corresponding letter frequencies using the `getfreq()` function. However, if you consider that when you shift all letters by a fixed amount, the resulting letter frequencies will actually be the same values, with only their position in the letter frequency table shifted! You can therefore also create the shifted letter frequency table directly from the original letter frequency table (which you can create by processing the original encrypted '`secret`' text: `getfreq(secret)`).

In the end, your function should look as follows:

```
def findshift(secret):
    ''' For some encoded secret text, find the most likely caesar shift.

        Arguments:
        - secret: The encoded text

        Returns:
        - optshift: The most likely cipher key
    '''
    # Your code which iterates over all possible cipher shifts (1-25):
    ...
    return optshift
```

You can test your function using the example `testdata.txt` secret file:

Test example

```
with open('testdata.txt') as fin:
    print(findshift(fin.read()))
```

If your `findshift()` function works correctly this should print a cipher key of 21.

The `cipher()` function

The process to encode or decode a piece of text with a Caesar cipher is the same whether you want to encode or decode; only the direction of the shift is different: use a negative shift to encode a piece of text, and a positive shift to decode it.

For your program, create a function that implements this encoding/decoding cipher. The function should take two arguments: the original or encrypted text, and the cipher shift key with which the text should be shifted. This function should look as follows:

```
def cipher(text, shift=0):
    ''' Function to encode (negative shift) or decode (positive shift)
        the provided text.

        Arguments:
        - text: A string containing the text to encode or decode
        - shift: The cipher key that indicates the offset to shift
                    each character with

        Returns:
        - newtext: A string containing the encoded/decoded text
    '''
    # Your encoding/decoding code here
    ...
```

```
    return newtext
```

When complete your function should take each character from input string `text`, shift the character by `shift` positions if it's a letter, or leave it unchanged if it's not, and add the result to an output string, which is returned by the function upon completion.

The following are some tips for the implementation of your `cipher()` function: As described above, each ASCII character can be interpreted as a numerical value using the `ord()` function. You can therefore use this function to get the position in the alphabet of a given letter, by subtracting the ASCII position of the letter 'A':

```
idx = ord(ch.upper()) - ord('A')
```

Because characters are internally represented as numbers, you can also compare them as if they are numbers. So you can test if a character is a letter by doing the following:

```
if 'A' <= ch.upper() <= 'Z':
    # ch is indeed a letter
```

Using the Caesar cipher means you are shifting characters *cyclically* in the alphabet. This means that for **positive** shifts, you have to take special care with letters at the end of the alphabet (which will shift beyond letter 26), and for **negative** shifts you need special attention for letters at the start of the alphabet. The modulo operator (`%`), which returns the *remainder* of an integer division, is very handy for this. Remember that 28%26 = 2 and 32%26 = 6, 4%26=4. If we add the cyclically shifted alphabetical index to the ASCII code of "A"(=65) for an upper case character or to the asci code of "a"(=97) for a lower case character, we can use the `chr()` function to get the Caesar-shifted character. Note that when a text has been encoded with a shift of +5, it can be decoded with a shift of both +21 or -5 due to the cyclical principle.

In your cipher code, don't forget to distinguish between uppercase and lowercase characters! If you want, you can test your `cipher()` function in the interactive Python console using the following example:

Test example

```
>>>cipher('Hello World!', 5)
'Mjqqt Btwqi!'
>>>cipher('Your mother was a hamster', -7)
'Rhnk fhmaxk ptl t atflmxk'
```

Iterating over the secret files and decoding them

With all functions defined, the final piece of code in your file should loop over all the files you want to decrypt, determine the correct cipher key, and store the decrypted text to a new file. To this end, end your program with a block of code containing the following steps:

- Loop over all input '`secret`' files, and
  - Read the entire file contents into a string '`secret`'
  - Determine the correct deciphering key using the `findshift()` function
  - Decipher the encrypted text using the found key
  - Store the deciphered text in a new file
  - Print the deciphering key, the original filename, and the new filename

You know you assignment works correctly if all the generated files contain recognisable English text.