Week 12: Caesar cipher and file handling

This is a **bonus worksheet**, intended for anyone who enjoyed learning Python this semester. We will see how to implement a cipher, and how to read and write text from and to a file. Don't worry, you don't need to look at this sheet if you don't want to!

1 Introducing the Caesar cipher

Suppose Alice wishes to send a message to Bob, but doesn't want anyone else to read it. Then she will 'encrypt' the message in some way so that it will be unreadable if a third person Eve — the eavesdropper — intercepts the message and such that Bob will be able to reconstruct the message when he receives it. The *Caesar cipher* is a simple and very old way to encrypt a message. One way of describing it is as follows. The message Alice wishes to encrypt is called the *plaintext*. Let's suppose it is

The cat sat on the mat.

Alice fixes a number, called the *shift*. Each letter in the plaintext is moved along in the alphabet by the shift. So if the shift is 2 then 'T' becomes 'V', 'H' becomes 'J', 'E' becomes 'G' and so on. Thus the message above becomes

Vjg ecv ucv qp vjg ocv.

Note that if we go off one end of the alphabet then we come back on the other end. So with a shift of 2, 'Y' becomes 'A' and 'Z' becomes 'B'. To make the message harder to *decrypt*, Alice removes all punctuation and makes all the letters upper case.

VJGECVUCVQPVJGOCV

Alice can now send this message — called the *ciphertext* — to Bob, knowing that Eve will not be able to read the message if she intercepts it. Provided that Bob knows the shift that Alice used, he can decrypt by shifting the letters in the opposite direction by the shift. So he would obtain

THECATSATONTHEMAT

and he can hopefully put the spaces and punctuation back in by hand to recover the plaintext

The cat sat on the mat.

Exercise 12.1. Decrypt the following message which was encrypted with a shift of 3. It is the first half of the opening sentence to a famous book. What is the second half of the sentence? (As is common, I have collected the letters in groups of five, to make them more readable.)

LWZDV DEULJ KWFRO GGDBL QDSUL O

Hint: you might find it easier if you write out the alphabet in order first.

2 Implementing the Caesar cipher in Python

We wish to implement the Caesar cipher in Python. The key point is going to be knowing how to shift letters along in the alphabet. Computers in general, and Python in particular, have a standard way, called Unicode, of associating every character to a number. So, for example, 'A' corresponds to 65, 'X' corresponds to 88, 'a' corresponds to 97, 'x' corresponds to 120 and '!' corresponds to 33. Python calls the associated number the *order* of the character and you use the command `ord()` to find it out.

```
for character in "ABCabc123?!@":
    print("The order of \"{0}\" is {1}"
          .format(character, ord(character)))
```

To find the character associated to a given order you use `chr()`. Here you can see the characters from order 65 to 125.

```
for order in range(65, 126):
    print(order, chr(order), sep="\t")
```

(Change the range to `range(15065, 15126)` and see what you get.)

We now start to see how we might implement the Caesar cipher: we can add the shift to the order of each character in the plaintext. However there is a complication in the cycling round from one end to the other. The order of 'Z' is 90, adding 2 to that gives 92, which has associated character '\' and not 'B' as we want.

The trick is to do arithmetic modulo the number of letters in the alphabet, 26, i.e. we use modular arithmetic. So we define the position of a letter to be its position in the alphabet, so 'A' gives 0 and 'Z' gives 25. This is calculated from the order by `position = order - ord("A")`. We can add our shift to this and take the remainder on dividing by 26, so 'Z' gives 25, add 2, gives 27, divide by 26, gives remainder 1 which corresponds to 'B'. In Python we will want to use the following.

```
shifted_position = (position + shift) % 26
```

Putting this together we can construct the following function.

```
1  def caesar_shift(character, shift):
2      """
3      Apply a Caesar shift by 'shift' letters to 'character',
4      if 'character' is not a single letter then return the
5      empty string ""
6      """
7      if len(character) != 1:
8          return ""
9      character = character.upper()
10     if character >= "A" and character <= "Z":
11         position = ord(character) - ord("A")
12         new_position = (position + shift) % 26
13         return chr(new_position + ord("A"))
14     else:
15         return ""
```

Note on Line 10, that we can compare strings, this line ensures that the character is an uppercase letter; we could also have written `ord(character) >= ord("A")`, and similarly for 'Z'.

Type the above function in to the editor then test that it works by adding the following lines underneath to form a program.

```
18 print("A with a shift of 2 gives", caesar_shift("A", 2),
19       "- should be C")
20 print("Z with a shift of 2 gives", caesar_shift("Z", 2),
21       "- should be B")
22 print("! with a shift of 2 gives", caesar_shift("!", 2),
23       "- should be empty")
```

You can add more tests with, say, lower case letters and, say, negative shifts (doing this more formally leads to 'test-driven development') however, we will leave it at that for now.

Comment out the above six testing lines — e.g. by selecting them all and using the comment command in the Edit menu — and add the following lines at the end to allow you to enter some text.

```
25 text = input("Text to shift: ")
26 shift = int(input("Shift: "))
27
28 for character in text:
29     print(caesar_shift(character, shift), end="")
```

Exercise 12.2. (i) Test this code by applying a shift of 2 to the text 'The cat sat on the mat.' then check that the output is correctly decrypted with a shift of −2 (you can copy-and-paste the output).
(ii) Redo Exercise 12.1 using this program and check you get the same answer.

3 Reading text from a file

In general, when you are sent a message to decrypt, it would be better if you did not have to type it in or copy-and-paste it. Python is able to read text from a file, so if someone sends you an encrypted file you can just tell Python to look in the file. Change lines 25 onwards to the following. We will analyze what is happening below.

```
25 infile_name = input("Name of file to shift: ")
26 shift = int(input("Shift: "))
27
28 with open(infile_name, mode="r") as infile:
29     text = infile.read()
30     for character in text:
31         print(caesar_shift(character, shift), end="")
```

Line 28 opens the file whose name is contained in the variable `infile_name` for 'r'eading and creates a Python **file object** called `infile` which is the name we will use to refer to it. This file will be readable in the indented code block that follows the **`with`** command. Line 29 reads the whole file and puts it into the string called `text`. The rest of the code is just as if we had entered the text via an `input()` command as in the previous version of the program.

Exercise 12.3. On the course webpage find the file `humpty.txt` and save it in the same directory as your python file. You can open the text in a text editor if you like and verify that it looks like nonsense. Now run the program: when prompted for the file name enter `humpty.txt` and enter −2 for the shift. This gives output that you should be able to read. (If you copy-and-paste the text you can add spaces in the correct places.) Can you identify from which piece of literature written by a mathematician it comes from?

Exercise 12.4. The file `puzzle.txt` at the course homepage contains an encryption, using the Caesar cipher, of the words of a Nobel Prize winner. However, you do not know what the shift is. Decode the message and identify the Nobel Prize winner! You might find it useful to save a different version of the program under a different name and use a **`for`** loop to go through and print out all 25 possible decryptions.

The last exercise demonstrates, hopefully, how, since the invention of the computer, the Caesar cipher and other simple methods of encryption have generally become redundant as they are reasonably easy to decrypt if you intercept a message. Much more sophisticated methods of encryption are employed today. You will have met RSA in MAS114 Numbers and Groups.

Using the command `file_object.read()` is very unsubtle, loading all of the file at one go, this can also be very inefficient. There are two alternatives to this. One is to read the file one line at a time using the `file_object.readline()` command, but we won't use this here. The other is to use `file_object.read(n)` where `n` is an integer. This reads *the next* `n` characters in the file. When Python reaches the end of the file this function just returns the empty string `""`. Thus an alternative to the above piece of code from Line 28 onwards is the following. The effect should be the same. You do not have to type this in.

```
with open(infile_name, mode="r") as infile:
    character = infile.read(1)
    while character != "":
        print(caesar_shift(character, shift), end="")
        character = infile.read(1)
```

4 Writing text to a file

As well as reading input from a file, Python can also write output to a file. Instead of using the `print` command to write to the screen you use the command `file_object.write()` in a code block, having opened the `file_object` for writing using `with open(file_name, mode="w") as file_object:`

Exercise 12.5. What will the following program do precisely?

```
with open("characters.txt", mode="w") as out_file:
    out_file.write("Some characters:\n\n")
    for order in range(33, 128):
        out_file.write(chr(order))
        if order % 16 == 0:
            out_file.write("\n")
```

Run the program then examine the file `characters.txt` which will have been created. What is the `if` command doing? Change the `16` to a `2` and check again to make sure you know what it does.

We can open a file to read and a file to write at the same time. We can put the `open()` commands in the same `with` lines if we separate them by a comma. Save your Caesar cipher program as `caesar_in_out.py` and modify the final lines to be the following.

```
25 infile_name = input("Name of file to shift: ")
26 outfile_name = input("Name of output file: ")
27 shift = int(input("Shift: "))
28
29 with open(infile_name, mode="r") as infile,\
30      open(outfile_name, mode="w") as outfile:
31     text = infile.read()
32     for character in text:
33         outfile.write(caesar_shift(character, shift))
```

Run this version of the Caesar cipher program using `humpty.txt` as the input file and call the output file something like `humpty_out.txt`. Open the output file and check the program has worked.

Being able to save the output of your program in a file can be very useful.

5 Where next?

Next semester you will be doing programming with R, which is one of the standard languages for computing with statistics. If you wish to go deeper into programmin, I suggest the first thing you learn is object oriented programming in Python. If you wish to learn another programming language, I suggest that you look into C++, Java, Julia, and Haskell, then decide which one you want to learn. Another possibility is that you learn SageMath or SymPy, which are computer algebra systems — like Mathematica — based on Python, and useful for higher mathematics. In Level 2, you have the chance to take MAS212 Scientific Computing and Simulation, which looks further at numerical computations with Python.