Python

Module 2

Your First Games

Bits and Bots
study group

Content

This module

For this module, make sure you:

- Make sure you understand everything from Module 1 (a short recap is provided below). We will continue with the Python basics that we've learned.
- Read this module. While working through it, you will also use the following reading material:
    - Chapter 2 of *Automate the Boring Stuff.* This chapter will cover some of the same concepts (Booleans, comparison operators, importing modules, etc.) and has practice questions.
    - Read and work through chapter 3 of *Invent Your Own Computer Games With Python.* Be able to understand the concepts covered in this chapter.

Further learning:

- If you want to do more reading and practising on the basics, check out the Python Tutorial (w3schools.com). Here you can check out multiple assignments and try out the basics for yourself.
- Chapter 3 of Michael Dawsons, Python Programming for the Absolute Beginner.

Recap of Module 1

In the first module, you've installed Python, and have learnt more about the Python and GitHub basics. Every Python module from now on starts with a short recap. Nevertheless, it is wise to create your own notes while studying.

An ***expression*** is the most basic kind of programming instruction in Python. An expression is a combination of operators and operands that is interpreted to produce some other value, for example: `2 + 2 = 4`. The operator + (addition) evaluates to the single value 4. It is also possible to use more than one operator in an expression. For example: `10 + 3 * 4`. Going without precedence it could have given two different outputs 22 or 52. But now looking at operator precedence, it must yield 22.

A ***data type*** is an important concept in programming. It is a category for values. Examples are integer, float and string. An integer is a round number, like 22. A float is a number such as 2.57. A string is a text value, for example: Hello, World!

Variables are containers for storing data values. A variable is created the moment you assign a value to it. This means that the moment you type `x = 36` in your code, you have created a variable (named `x`) with the value `36` (which is the data type integer).

A ***function*** is a block of code that will only run once it's called. You can pass data into a function and it will return data as a result. You can create functions yourself, or make use of existing functions, like `print()`, to display something on your screen, or `input()`, in order to let the user type something that can be used as input.

GitHub is a platform that allows anyone to create, store, manage, and share content. The study group Bits and Bots makes use of GitHub as well. Our repository is used to share learning materials and games that have been created by members. Every repository has a

READme file that is written in ***Markdown***. This is an easy-to-read and easy-to-write language for formatting plain text. In GitHub the changes you make are automatically tracked. You can edit, create a branch and merge changes in the main repository. By default a repository has a ***main branch***. By creating your own branch, you can have different versions of a repository at one time. Saved changes are called ***commits***, accompanied by a commit message that explains the changes that were made. The work on different branches will not show up on the main branch, until you ***merge*** it. This can be done with a ***pull request***. We have provided a guide on GitHub, based on the expert session. It can be found here.

Read, Create, and Review

It is time to create your first games in this module! To review what you have done and learned, it is important to write down your process. So, for example, if an error occurs, make sure to take a screenshot so you can discuss it in the group on Discord or during a monthly meeting. Also write down what you have done to fix the problem, step by step.

When learning something new, repetition is important. Try to remember to review what you have done and see if you can recognise and understand which variables, data types, statements, and functions you have used. Are you able to explain what every line of code does, in your own words?

Please remember that Rome wasn't built in a day! It is better to divide the work and exercises that are given in this module. Every part contains a black box with instruction what to do exactly. This might help you to structure the workload.

Last but not least, you can discuss your games with other members of the study group in the Discord channels. For example: What errors did occur? And how did you solve them?

Getting started: Creating a Flowchart

A flowchart will help you to understand the logic of the code you're creating better. Often, there are more ways than one to get to the end of a program. Have a look at the flowchart below. As you can see, there are more ways to decide what to eat tonight in this example. These branching points, where you have multiple choices to go to the next step. The other steps are represented with rectangles.

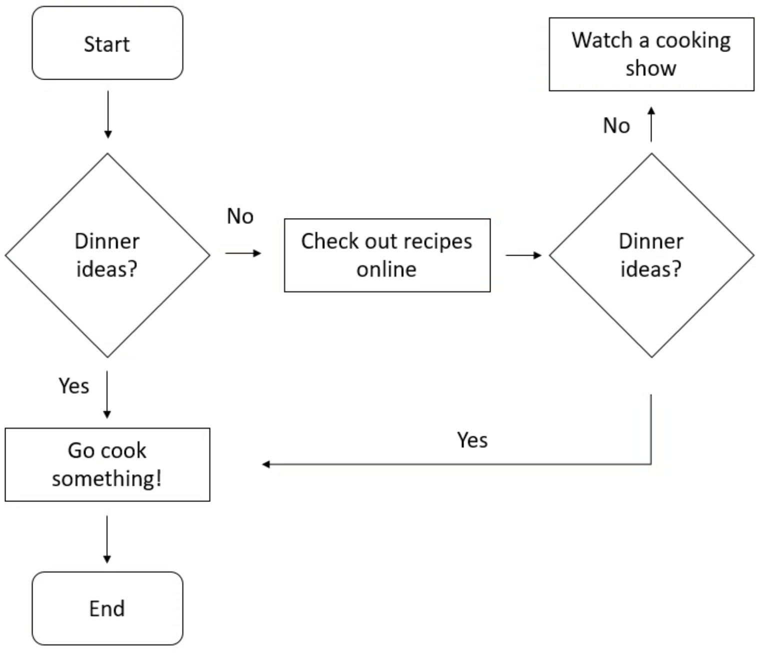


Figure 1: Flowchart dinner ideas

If you look closely, you'll see that this flowchart is not finished. If this was a program, and you end up watching a cooking show, you would not be able to stop and go to the end. When coding in Python, you have to understand how to stop the program and how to represent these yes and no options. Furthermore, it is possible to make your code more complex. You can add extra conditions. For example `if` you want to eat healthy, eat broccoli, `else` order pizza.

In order to learn how to express conditional choices in code, read and work through chapter 2 of *Automate the Boring Stuff* until "A Short Program: Guess the Number".

Create a Flowchart for Guess the Number

Let's start with your first game: Guess the Number! Both books of Al Sweigart cover this game, but for this module we will follow 'Invent your own computer games with Python'. In chapter 3 you will learn more about functions, the ***Boolean*** data type and about ***flow control statements***. Also, we will import the ***random module*** in the code. By importing this module, the program can call the `randint() function`. This function will come up with a random number for the player to guess.

Before reading the chapter and writing the code for this game, look at the output below. This is what will appear on your screen after you run your code and play the Guess the Number game. The player's input is marked in bold.

```
Hello! What is your name?
Albert
Well, Albert, I am thinking of a number between 1 and 20.
Take a guess.
10
Your guess is too high.
Take a guess.
2
Your guess is too low.
Take a guess.
4
Good job, Albert! You guessed my number in 3 guesses!
```

Create a flowchart for the Guess the Number game, like the one we provided in figure 1. It might help to answer the question: What is it exactly that you will tell your computer to do?

Recreate Guess the Number

Now that you have made a flowchart, it is time to recreate the game. When working through the chapter, remember: Don't copy the code from the book, but type every line down by yourself in IDLE or Mu and save your file. This may seem a bit dull, but it helps you understand the structure of the code better, because you have to type in every character yourself. After you did so, run it and see what happens. Then closely read the chapter with your program at hand.

Both IDLE and Mu will give you some hints when writing your code. They show different colouring for functions and statements and provide hints on closing your functions and strings. They also allow you to press tab to insert four spaces or provide the spaces automatically. You can try both IDLE and Mu out and see what suits you best.

Read and work through chapter 3 of Invent Your Own Computer Games, "Guess the Number".

Create a flowchart for Rock, Paper, Scissors

With similar principles that you have learned and used in the Guess the Number game, you can create another game: Rock, Paper, Scissors. It is recommended to (re)read chapter 2 from 'Automate the Boring Stuff' and recreate the game, but you can also skip to: "A Short Program: Rock, Paper, Scissors" (near the end of Chapter 2).

Below you can see the output of Rock, Paper, Scissors. The lines in bold are the input that the player gives to the computer, to tell their move. Start with creating a flowchart for the game.

```
ROCK, PAPER, SCISSORS
0 Wins, 0 Losses, 0 Ties
Enter your move: (r)ock (p)aper (s)cissors or (q)uit
p
PAPER versus...
PAPER
It is a tie!
0 Wins, 1 Losses, 1 Ties
Enter your move: (r)ock (p)aper (s)cissors or (q)uit
s
SCISSORS versus...
PAPER
You win!
1 Wins, 1 Losses, 1 Ties
Enter your move: (r)ock (p)aper (s)cissors or (q)uit
q
```

Create a flowchart for Rock, Paper, Scissors, like you did for Guess the Number.

Recreate Rock, Paper, Scissors

After you've created the flowchart, it is time to recreate the game. Don't forget to review what you've done, as explained at the beginning of this module.

Identify which variables, data types, statements and functions you have used. Try to explain what errors occurred (if any) and how to solve them. Explain briefly in your own words what every line of code does. This helps you to understand the program better, and how to debug it. We will cover debugging the code in more detail in the next module.

Programming values and principles

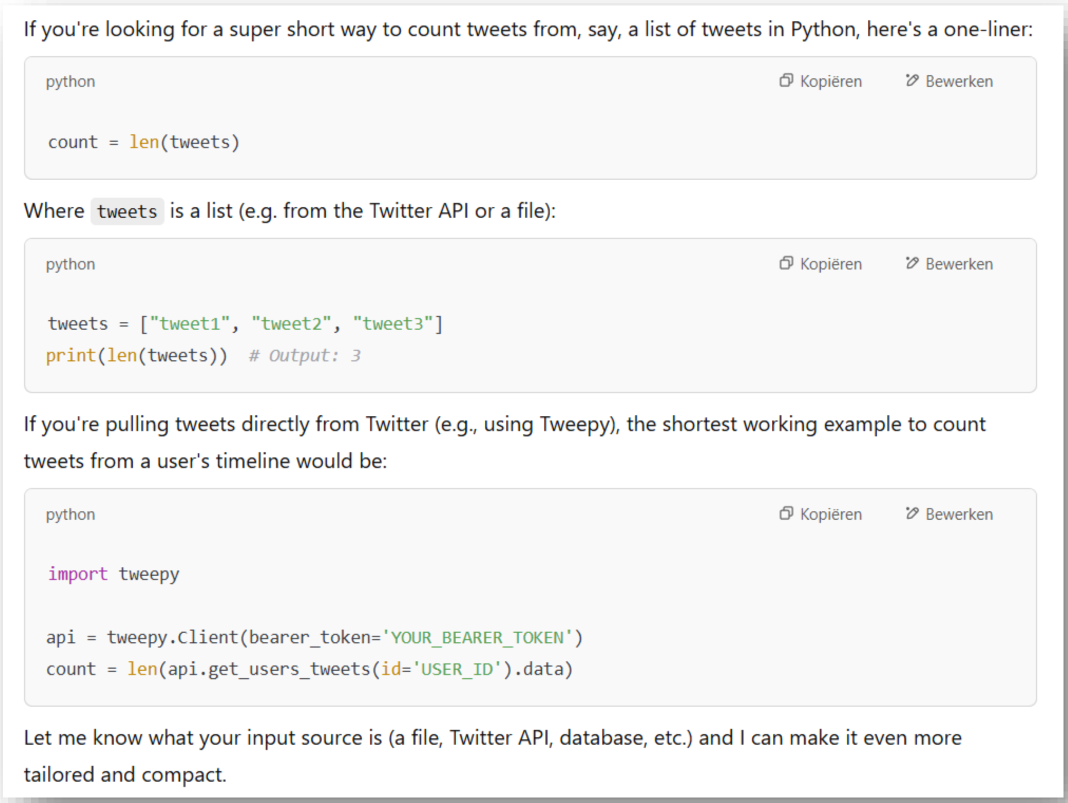
The expert session in May is about "Programming values and principles" and will be given by Daniel Steinmeier. This part of the module should be done after that expert session. We will also make sure to provide you all the slides of the session afterwards to aid you in the assignment.

Abstract expert session
In programming there is a certain notion that some ways of writing code are more elegant than others. We might have an intuitive understanding of the notion of elegance when it comes to natural language. But what is elegance in coding? As we will see this can be the complete opposite of elegance in natural language. Take synonyms for example. Synonyms provide variation and make text less boring. But one of the principles of Python coding is: "There should be one – and preferably only one – obvious way to do it."

What is this way? In the case of HTML similar principles apply. Why is it 'wrong' to use tables for lay-out? In this session we will look at important coding principles and the values behind them for both Python and the Web. As a sidestep we will also touch upon the question of how generative AI should and should not be used when aspiring to writing quality code.

Exercise programming values and principles

Prompt: "Show me the most short python code to count numbers of tweets"



If you're looking for a super short way to count tweets from, say, a list of tweets in Python, here's a one-liner:

```python
count = len(tweets)
```

Where `tweets` is a list (e.g. from the Twitter API or a file):

```python
tweets = ["tweet1", "tweet2", "tweet3"]
print(len(tweets))  # Output: 3
```

If you're pulling tweets directly from Twitter (e.g., using Tweepy), the shortest working example to count tweets from a user's timeline would be:

```python
import tweepy

api = tweepy.Client(bearer_token='YOUR_BEARER_TOKEN')
count = len(api.get_users_tweets(id='USER_ID').data)
```

Let me know what your input source is (a file, Twitter API, database, etc.) and I can make it even more tailored and compact.

Exercise 1: Checking validity of a generated answer

1. Which of these three would you choose as a start for counting the tweets in your personal archive? Why?
2. Look at the third option. Find the Tweepy documentation online and answer the following questions.
    a. What does the function Client.get_users_tweets() do?
    b. Are there any limitations mentioned when using this function?
    c. Are there any alternative functions for counting tweets?
    d. Without testing, do you think this function will work as expected? Explain your answer.

Exercise 2: Refactoring your code

Download the file 'calculate.py', that has been shared on Discord with Python Module 2, and save it in a local directory. This script counts the number of occurrences of values in a list. The list is based on a dataset offered as open access by the KB. You can find this dataset here: Metadata: Overview Sources

Open the xlsx page and copy-paste the column with languages in an empty text file. Name the file 'books-languages.txt' and save it in the same directory as the calculate.py script.

Run the script, if all goes well you should see as output a list with the number of unique languages in the file and a dictionary with languages and number of occurrences.

This script is pretty ugly! Make it more pretty in 4 steps. Make use of the slides of the expert session, if you need help.

1. Deduplicate the code, make sure the same code is not repeated, especially the code for reading the file.
2. Make it pythonic! What is the pythonic way of handling files? Are there other things that can be improved?
3. Create a function for opening the file and extracting the values. Make sure to return something sensible.
4. Make printing prettier and place comments.

As a bonus or when you have progressed your Python skills there are a number of improvements that can still be made so be sure to come back to this script and improve it even more. Things that might make it even better: making the filename an argument that can be passed on the command line (with sys.argv), packing the printing statements in a function as well, using more columns in the excel-file by splitting on tab instead of assuming the values to be a whole line, reading the xlsx as-is with the pandas-library, writing the output to a file or creating a simple user interface for it with Tkinter.

Make sure to write down your findings for both exercises. The answers will be provided in the next module.