Python

Module 3

Functions, Loops, and Debugging

Bits and Bots
study group

Inhoud

This module

For this module, make sure you:

- Read module 3 in the guide. Make sure you understand the concepts that will be used in this module and that you know what is expected of you.
  - If needed, read the summaries at the end of chapter 2 of *Automate the Boring Stuff*[1] and chapter 3 of *Invent Your Own Computer Games With Python*[2].
  - If you haven't done so already, it can be wise to make notes while working through the modules. This can be your own summary.
- Read chapter 5 of *Invent Your Own Computer Games With Python.*[3]
- Read chapter 6 of *Invent Your Own Computer Games With Python*[4] and chapter 11 of *Automate the Boring Stuff*[5] on debugging code.

Further learning:

- If you want to do more reading and practising on the basics, check out: Python Tutorial (w3schools.com). Here you can check out multiple assignments and try out the basics for yourself.
- You can reread Chapter 3 of Michael Dawsons, *Python Programming for the Absolute Beginner*, about while loops and else/elif clauses, if necessary.
- Want to try out some commands that you can use in the command prompt on Windows? Check out the: Command Challenge! (cmdchallenge.com)

Recap of Module 2

It is advised to create a flow chart for every script or game you create, especially when you are just getting started with Python. This way, you make it obvious what you want your program to do, like skip (`else`, `if` or `elif` statements) or repeat instructions (`while` and `for` loops). With the help of the Boolean values `True` or `False`, the computer knows how to handle parts of your code.

In the first two modules, we have seen various built-in functions like `print()` and `input()`. Python also has a standard library that contains a set of modules that you can import in your code. For example the `random` module that has been used to create the first two games.

[1] Automate the Boring Stuff with Python
[2] Chapter 3 – Guess the Number (inventwithpython.com)
[3] Chapter 5 - Dragon Realm (inventwithpython.com)
[4] Chapter 6 - Using the Debugger (inventwithpython.com)
[5] Automate the Boring Stuff with Python

Answers Exercise Programming Values and Principles

In module 2, you have made an exercise concerning programming values and principles. In this module we will provide you with the answers of the exercise.

Exercise 1 - Checking the validity of an AI-generated answer

Question 1
Which of these three would you choose as a start for counting the tweets in your personal archive? Why?

Question 1 – answer

- The first option is not very helpful, it only shows you how to count the length of a variable but no pointers on how to get the Twitter data into the variable.
- The second option generates a bogus list and is just as unhelpful.
- The third option is the most interesting but requires real data like the authentication-token. Even though it is the most promising solution it requires a lot of context to work, like you need to know where to get the token and the user_id, so it is not a complete working solution.

Question 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.

Question 2 – answer
About Tweepy:

a) The function gets a list of tweets from a single user.
b) It only fetches the most recent 10 tweets, for the rest you need pagination.
c) There is a function Client.get_all_tweets_count but this requires a special account from Twitter.
d) No, because of the fact that it is paginated.

Exercise 2 – Refactoring your code

Locate the file 'calculate.py'. 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 it here: Metadata: Overview Sources

Open the xlsx on this page and copy-paste the column with languages in an empty textfile. Name the file 'books-languages.txt' and put 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:

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.

Model answers

Step 1

```
fh = open('books-languages.txt') #warning: ugly coding for educational purposes
content = fh.readlines()
langdict = {}
for lang in content:
    lang = lang.strip()
    if lang in langdict:
        langdict[lang] += 1
    else:
        langdict[lang] = 1
print(langdict.keys())
print(repr(langdict))
fh.close()

#improve this code
#first step: deduplication of code
#second step: make it pythonic
#third step: create functions
#fourth step: make printing more user-friendly and create comments
```

Step 2

```
with open('books-languages.txt') as fh: #warning: ugly coding for educational purposes
    langdict = {}
    for lang in fh:
        lang = lang.strip()
        if not lang in langdict:
            langdict[lang] = 0
        langdict[lang] += 1
    print(langdict.keys())
    print(repr(langdict))

#improve this code
#first step: deduplication of code
#second step: make it pythonic
#third step: create functions
#fourth step: make printing more user-friendly and create comments
```

Step 3

```
def count_items(textfile):
    with open(textfile) as fh: #warning: ugly coding for educational purposes
        langdict = {}
        for lang in fh:
            lang = lang.strip()
            if not lang in langdict:
                langdict[lang] = 0
            langdict[lang] += 1
    return langdict

f = 'books-languages.txt'
d = count_items(f)
print(d.keys())
print(repr(d))

#improve this code
#first step: deduplication of code
#second step: make it pythonic
#third step: create functions
#fourth step: make printing more user-friendly and create comments
```

Step 4

```
#Warning: too many comments for educational purposes
def count_items(textfile): #function for counting items in a textfile, assuming each item is on a new line
    with open(textfile) as fh:
        langdict = {}
        for lang in fh:
            lang = lang.strip() #stripping newline to get the clean value
            if not lang in langdict: #when a new item is encountered create a dictionary entry for it
                langdict[lang] = 0 #set counter to zero
            langdict[lang] += 1
    return langdict #return the whole dictionary consisting of item as key and count as value

f = 'books-languages.txt' #variable for local file holding the values to be counted
d = count_items(f) #call function and store dictionary in d
print('Number of unique items: ' + ', '.join(d.keys())) #keys are the unique entries, join creates a comma separated list
for i in d: #iterates over the keys
    s = '{:<20}' #creating a fixed length for the entries so numbers will align
    print(s.format(i) + '\t' + str(d[i])) #print entries and count on a separate line
```

Dragon Realm

In part 1 of this module, you will create another game: Dragon Realm (chapter 5). In this game, the player is in a land full of dragons. The dragons all live in caves with their large piles of collected treasure. Some dragons are friendly and share their treasure. Other dragons are hungry and eat anyone who enters their cave. The player approaches two caves, one with a friendly dragon and the other with a hungry dragon, but doesn't know which dragon is in which cave. The player must choose between the two.

To make this game, you will have to import libraries, create your own functions and use `else/if` and `while` loops. A new function (the `sleep()` function) is introduced, and you will work with the `and`, `or`, and `not` Boolean operators again.

Create a flowchart for the Dragon Realm game, based on the output below.

Below you see what Dragon Realm should look like when you run the code. The player's input is in bold.

```
You are in a land full of dragons. In front of you, you see two caves.
In one cave, the dragon is friendly and will share his treasure with you.
The other dragon is greedy and hungry, and will eat you on sight.
Which cave will you go into? (1 or 2)
1
You approach the cave...
It is dark and spooky...
A large dragon jumps out in front of you! He opens his jaws and...
Gobbles you down in one bite!
Do you want to play again (yes or no)
no
```

After making you flowchart, work through chapter 5.

Write down: Did your flowchart look the same as the one provided in the chapter? What were the differences? Try to figure out what would and wouldn't work in the game if you have missed a step.

Practice More With Loops

Now that you've seen loops in action, it makes it more clear what they can do. In programming, loops are very important to add an extra logical layer in your code. For games, this means that you can add choices. In general, the use of loops can help you count things, or execute a set of statements.

If you want to practice more with loops, try out these exercises or do this quiz.

Debugging Code

Part 2 of this module revolves around debugging. As explained in the introduction, you have to be very precise in telling the computer what to do. When you make a tiny mistake or a typo, like forgetting to end a string with a quote, the computer will give you an error message. These errors are called bugs. In Python the computer will indicate where the program stopped running. In this module, you will learn more about bugs and how to fix them. This might not be a very fun part of the module, but it can be very helpful to understand Python better. A detailed explanation can be found in Al Sweigarts 'Invent Your Own Computer Games with Python', chapter 6 and chapter 11 in 'Automate the Boring Stuff', of the same author.

Chapter 6 covers the types of bugs that can occur and how to use a debugger. This is a program that lets you go through your code one line at a time in the same order that Python executes each instruction. There's a bit of overlap with chapter 11, but the first half of chapter 11 explains how you can get more detailed information about the bugs and how to use the logging module that enables you to display log messages on your screen as your program runs. It helps you understand what's happening in your program and in what order.

Try and fix the code! Now it's time to apply what you've just learned in other pieces of code. Here you can find a code with Syntax Errors. Can you fix them? If you're up for it, try to fix the Logic Errors as well. For the exercise with the syntax errors, make notes during the progress. Before running and altering the code, have a look at it. Can you already identify some of the bugs?

The Command-Line Interface

The expert session in module 3 is focused on the command-line interface. While the expert session will show you the commend-line interface in Windows (the command prompt), we will also make sure to provide you with the commands for the other operating systems (Mac/Linux). In this last part of the module, you're going to apply what you've learned from the expert session yourself. Do not forget to use the command-line interface guide for this exercise. We will share this guide with you after the monthly meeting (option 2).

Exercise Command-Line Interface

For this exercise, you are going to try out some commands and bundle them in a batch script. Follow the steps below. Make sure to only use the command-line interface for steps 2-6.

1. Open the command prompt
2. Navigate your way to a folder
3. Show the content of that folder
4. Make a new folder (in the folder you are at)
5. Duplicate a file and put the duplicate in your newly created folder
6. Calculate a checksum[6] (SHA512 or MD5) of the file in your new folder
7. Create a batch script that includes steps 4-6.
8. Execute that batch script.

You've learnt how to use the interactive shell and save and run your Python programs. It is also possible to run a python file directly from the command-line interface. Try it out! Open the command-line interface and execute a Python file.

Extra exercise
Make a list of the most useful commands for you and try to combine them in a batch script.

[6] Checksums are values that are generated from transmitted data before and after transmission. They are a sort of fingerprint for your file. If the checksum stays the same, you know the file is the same. If anything has changed, the checksum too will have changed. This can indicate bit rot or an error that happened during migration.