Advanced Python Exercises

version 2020-08

Licence

1) Chapter 1 – writing better code

Comprehensions

Exercise 1.1

Use the Thonny interactive window to:

i) We want to write out the first 100 cube numbers. Write a **one-line list comprehension** to do this.

ii) We have a set of the names:
`characters = {'wotan', 'fricka', 'loge', 'siegfried'}`
Capitalise the first letter of each name with a list comprehension.

iii) Repeat ii), but generate a set as output instead of a list.

iv) Write a list comprehension to filter `range(50)` for odd numbers.

v) Write a list comprehension to filter `range(50)` for odd numbers and then print the square of these numbers.

vi) The list below contains scores as a fraction of 1. Convert these to percentages formatted to 1 decimal place using a list comprehension and an f-string.
`pc = [0.2, 0.1, 1, 0.95]`

Building and using more complex data structures

Exercise 1.2

A chessboard comprises 64 squares, with 8 columns (A-H) and 8 rows (1-8). Using nested iterations, create a list of lists in which each inner list should correspond to a chessboard row. (Hint: iterate over a `range` to generate the numbers).

Now you have done this, iterate over the data structure to print out the chessboard, as shown below.

```
a8 b8 c8 d8 e8 f8 g8 h8
a7 b7 c7 d7 e7 f7 g7 h7
a6 b6 c6 d6 e6 f6 g6 h6
a5 b5 c5 d5 e5 f5 g5 h5
a4 b4 c4 d4 e4 f4 g4 h4
a3 b3 c3 d3 e3 f3 g3 h3
a2 b2 c2 d2 e2 f2 g2 h2
a1 b1 c1 d1 e1 f1 g1 h1
```

Call this script `chess_lists.py`.

Scoping

Exercise 1.3

Look at the code below to calculate the circumference of a circle. For some reason it isn't working. Go ahead and fix it. (Note: the formula for calculating the circumference of a circle is $2\pi r$, where r is the radius of a the cirle. Let $\pi = 3.142$.)

```
def circumference_calc(radius):
    circumference = 2 * 3.142 * radius

circumference_calc(5)

print(f"The circumference of a circle of radius 5cm is:
{circumference:.2f}cm")
```

Exercise 1.4

Similarly, try to fix the code below.

```
def circumference_calc(radius):
    circumference = 2 * 3.142 * radius
    return circumference

radius = 90

circumference_calc()

print(f"The circumference of a circle of radius 90cm is:
{circumference_calc():.2f}cm")
```

Object oriented programming

Exercise 1.5

Look at the code below that defines the class `Individual`. Copy this code into Thonny and make a script called `individuals.py`. Now add code to the script to interact with the `Individual` class to perform the following tasks (you may not understand everything in the Class, but that doesn't matter for this exercise).

i) Create instances of the class `Individual`. Name the instances `individual1`, `individual2` and `individual3`. During object instantiation, set the `character_name` of these three new objects to "Buster", "Tobias" and "Lucille" respectively.

ii) What happens if you pass these three newly created objects to the `print` function?

iii) Check whether Buster is happy. Get him to speak.

iv) Call Buster's `switch_mood` method. Is he still happy? What does he say now?

```
class Individual:

    Counter = 0

    @classmethod
    def AddOne(self):
        self.Counter += 1

    def __init__(self, character_name):
        self.character_name = character_name
        self.happy = True
        self.AddOne()
        self.id = self.Counter

    def __str__(self):
        return f'Instance: {self.id} {self.character_name}'

    def get_character_name(self):
        return self.character_name

    def is_happy(self):
        return self.happy

    def switch_mood(self):
        self.happy = not self.happy

    def speak(self):
        if(self.happy):
            return f'Hello, I am {self.character_name}'
        else:
            return 'Go away!'
```

Generators

Exercise 1.6

Make a generator that returns the phrase: "10 green bottles hanging on the wall!". On the next iteration, there will be 9 bottles – and so on until no bottles remain.

When you have got this working, perform one more iteration on the generator. What happens?

Exercise 1.7

Write a generator that upon every iteration returns a letter (in order) from the word 'Sisyphus'. On reaching the end of the word, the generator should return to the start, and so on, in perpetuity. Retrieve the first 50 letters from this generator and name the script `sisyphus.py`.

Exercise 1.8
Modify the generator created in the previous exercise to take an argument which corresponds to the word that should be repeated, one letter at a time. Iterate over this generator as many times you wish and call this new script `sisyphus2.py`.

Exercise 1.9*
Write a generator to produce The Fibonacci Sequence of numbers. In this series, the next number is found by adding up the two numbers before it:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

Print the first 100 Fibonacci Sequence numbers. Name the script `fibonacci.py`.

Exercise 1.10*
Write a generator to produce prime numbers (prime numbers are only exactly divisible by themselves or 1 i.e. 2, 3, 5, 7, 11, …). Print out all the prime numbers less than 1,000. Name the script `primes.py`.

Error Handling

Exercise 1.11
Modify the code below so that it now raises an exception if the data file 'colours.txt' is not found (an `IOError`). The script should then exit gracefully, giving an explanation to the user. Change the path to the script to something incorrect, so that the file is never found and the script always raises an exception. Name this new script `colours_errorcheck.py`.

```
my_file = 'colours.txt'

unique_colours_count = dict()

#Reading file
with open(my_file, 'r') as file:
    for total, line in enumerate(file):
        colour = line.strip()
        unique_colours_count[colour] = unique_colours_count.get(colour, 0) +
1

#Printing out results
print(f'Total colours: {total + 1}')    #enumerate starts at 0, so add 1

for colour, count in unique_colours_count.items():
    print(f'{colour}: {count}')
```

Exercise 1.12
Write a script that asks the user for two numbers and then adds these numbers together. Incorporate error handling into the script in case the user does not provide valid numerical values. Name the script `adder.py`.

Exercise 1.13*
Write a script called `capital_cities.py`. The script should read the file `captials_countries.txt` and write these into a dictionary, with the country name as the key and

the capital city as the dictionary value. The script should then prompt the user to input a country name. After entering a country name, the program displays to the screen the capital of that country. This process is in a loop, so will continue unless the user enters "quit", and then the program will exit.

Importantly, make sure this script incorporates error handling to deal with the `IOError` if the file `captials_countries.txt` is not opened, in which case the program will quit. If the country entered by the user is not found in the dictionary, this will also generate and error (when entering the sought dictionary key between square brackets – i.e. don't use the dictionary `get()` method). Raise an exception (you should be able to work out the type of error generated using the traceback message) to this and let the user know the country entered was not found. Then, continue in the loop.

(Note: make this country recognition process case insensitive e.g. France is treated the same as FRANCE.)

Chapter 2 – modules and packages

The documentation found at https://docs.python.org/3/index.html should help you with these exercises.

Exercise 2.1 – the math module

The `math` module assists with mathematical calculations. You can read up on the math module at: https://docs.python.org/3/library/math.html

Use the Thonny interactive mode to:

i) Get the decimal value of Pi.
ii) Get the decimal value of e.
iii) Calculate 21! (this in maths means factorial i.e. 21 x 20 x 19 x … x 1).
iv) Determine the ceiling value of 12.0000001.

Exercise 2.2 - the sys module

The `sys` module gives information relating to your Python implementation. In the interactive window, use the sys module to:

i) Find out the names of the command line arguments passed when running the script.
ii) Determine the currently loaded Python modules.
iii) Find the directories where Python looks for modules to import.

Exercise 2.3 – the time module

The `time` module is frequently used to add a defined delay to the running of a script. To illustrate this point, write a Python script called `delay.py` that counts from 1 to 10, with a delay of 1 second between each increment.

Exercise 2.4 – the datetime module

The `datetime` module allows you to obtain and manipulate date and time information. Create a script (named `age_in_days_calculator.py`) that asks the user for their date of birth. The script then returns their age in days. When writing the script, remember to handle errors generated by the user. (Hint: the code `datetime.datetime.now().date()` may help you with datetime-to-date conversion.)

Exercise 2.5 – the subprocess module

This module enables the user to run command line process from a script. To illustrate this point, write a script called `subprocess_delay.py` that runs the `delay.py` script. Notice anything different about how the results are displayed to the screen, as compared to running the `delay.py` script directly?

Exercise 2.6 – the glob module

The `glob` module is used to search a filesystem to identify files with names matching a pre-defined pattern. Write a python script named `globby.py` that searches the provided folder `sample_files` to:

i) To print to the screen the name of the files `*.conf`

ii) Print to the screen the number of files ending in `.csv`

iii) To print to the screen the contents of the `*.txt files`. Make use of the `subprocess` module for printing to the screen with either the command `cat` (on Linux/Unix/MacOS) or `type` (Windows). (These are operating system commands that you would normally run on the command line, usually via either the Bash shell, Mac terminal or the MS DOS prompt.)

Exercise 2.7 – the os module

The `os` module allows you to interact with your operating system via Python. Create a script named `os_exercise.py` and do the following:

i) Import the `os` module and use it to print to the screen your current working directory.

ii) Make a directory `called os_test_directory` using the `mkdir()` method. Check it has been created using your regular file navigation program.

iii) What do you think `os.sep()` returns? If you're not sure, then take a look at the Python documentation for clarification.

iv) The `os.environ` is a mappable object representing the environment variable (a dynamic-named value that can affect the way running processes behave on a computer). Iterate over it as you would a dictionary, and print the elements to the screen.

Exercise 2.8 – the String module

This module gives the user access to common character sets. To become familiar with the module, try the following exercise.

Remember from school that simple form of secret code to hide writing, in which letters were transformed into numbers depending on their position in the alphabet (e.g. cab = 3, 1, 2)? Well, use the character set in the `string` module to encode the sentence "It was a bright cold day in April, and the clocks were striking thirteen." (The punctuation can be ignored). The script should be called `enigma.py`.

(Hint: there are several ways to do this. It may be helpful to know that the `get()` method of a dictionary returns a `None` value, E rather than generating an error.)

Exercise 2.9 – the csv module

Let's try some importing/exporting data and performing simple statistics in a new script, that you should name `footie.py`.

i) Import the English Football Premier League Year End Table named 'Premier_League_Final_Table_1999.txt' using the csv module.

ii) Write out this same table to a CSV file using the `csv` module (essentially, we have now converted the file from a tab-separated file to a comma-separated file). Name this output file "Premier_League_Final_Table_1999.csv" (notice the changed file extension).

Exercise 2.10 - the textwrap module

The `textwrap` module was designed for the formatting of strings. Use the module in a script called `width_fomatter.py` that takes a phrase from the user and prints this phrase to the screen, but the width of each line should be fixed at 10 characters.

Exercise 2.11 – the Random module

The module `random` is used to generate pseudo-random numbers and we shall use it to create a Python script to generate 8-character passwords:

i) Import `randint` from `random` and then create a **generator** that creates an infinite stream of random characters (the `string` module may also help you here).
ii) Print to the screen an 8-character password.
iii) Save the script name as `password_generator.py`.

Exercise 2.12* – the csv module and statistics module

Use the `footie.py` script from a previous exercise.

i) Adjust the team names so that they are in capital letters when printed out to this csv file.
ii) Calculate the mean average points total (final column) and the associated standard deviation using the `statistics` module. Print these values to the screen.
Call this new script `footie2.py`.

Exercise 2.13* - the textwrap module

Modify the previous script `width_fomatter.py` to do the following:

i) Create a Python function that takes a FASTA sequence (`str`) and a width value (`int`) and then adjusts the nucleotide component (i.e. not the header) of the FASTA sequence so the width is wrapped to the number of specified characters. The function should return the formatted sequence. Name the function `format_fasta_width`.
ii) Make sure the function has documentation and performs the necessary assertions of the arguments passed to `format_fasta_width`.
iii) Incorporate the `format_fasta_width` function in a script named `fasta_formatter.py`. This script should read in the file "sample.fasta" and edit the width of the FASTA sequences to 40 characters and then write the formatted reads to a new output file.

Exercise 2.14 – the argparse module

The `argparse` module intelligently handles arguments passed to a Python script. We shall use this module to build on a previous script `width_fomatter.py`. Copy this file, creating a new script named: `width_fomatter2.py`. Now:

1) Use the `argparse` module to enable the filename(s) of the text files to be processed to now be passed via the command line.
2) Add a parameter `--width` to which should be passed the desired width of the formatted FASTA file. If this is not specified, it should default to 50.
3) There are always potential problems when allowing the user to specify files to process. Make sure the script handles potential errors with `try`/`exception` handling.

Exercise 2.15 - external modules

There is a multiplicity of libraries and modules external to core Python that may help with your work. We shall look briefly at two of these.

i) Matplotlib can be used for creating graphs and figures. Have a look at the website at: https://matplotlib.org/ and think how this may potentially help you with your work.

ii) Biopython is used for bioinformatics and computational biology. Have a look at the website at: https://biopython.org/.
iii) Install Biopython within Thonny on your machine (Tools > Manage Packages)
iv) Write a script named `biopython_trial.py` that makes use of Biopython for translating the following nucleotide sequence into amino acids.

```
atgaaaaacacggttaaactttcgcttattggttttgtcatgctcacagttcttctattaggagaaacggtgatt
gctcagaaaagaaaaccatgttattcacaagagccagacaaaacatgtgaagtcaatcgctgcaaagccaactgt
gtcaaaaaacataagaaaatactggcttttacctcgtgcattaaagaaaataacggaaatatgtattgccgatgt
caatatccttgccctccctaa
```

Chapter 3 regular expressions

Exercise 3.1 – introduction to creating regexes

Create a script called `hamlet_regexes.py` and try the following lookups on the line from the play Hamlet:

```
To be, or not to be, that is the question
```

i) Look for the pattern `To be` in the line from Hamlet. Print the object returned by performing the `match()` method on the pattern object.
ii) Repeat question i), only this time look up the string `question`.
iii) Try ii) again, but perform the `search()` method on the pattern object. Did you get a different result this time? Why?
iv) Count how many times the letter 'o' occurs in the phrase using the `findall()` method on the pattern object.
v) Repeat the previous question, but use the `finditer()` method on the pattern object.
vi) Print out the positions of all the occurrences of the letter 'o'. (Hint, use `finditer()` method once again).

Exercise 3.2 – character classes

Create a script called `doolittle.py` and make regexes to perform the tasks listed below on the phrase:

```
The rain in Spain stays mainly in the plain.
```

i) Print the capital letters
ii) Print the non-letter characters
iii) Print the words beginning with a capital letter
iv) Print the words containing `ain`

Exercise 3.3 – more regexes and metacharacters

Create a script called `operas.py` and make regexes to evaluate the following list. (Hint: a loop may help here.)

```
Rienzi
Nixon in China
Tosca
Il barbiere di Siviglia
```

```
Rigoletto
Le nozze di Figaro
Don Giovanni
Madama Butterfly
```

i) Identify operas containing a whitespace character.
ii) Identify operas ending with the letter `e`.
iii) Identify operas **not** ending with a vowel.
iv) Identify operas containing the letter `T`.
v) Repeat iv, but add the compilation flag `IGNORECASE`. How does that affect the results?
vi) Identify operas containing the phrase `di` followed by any three characters. Also, capture those three characters and print them to the screen.

Exercise 3.4 – even more regexes and metacharacters

Once again, perform the tasks below, saving the script to a file named `misc_regex.py`.

i) You have created a regex for a well-known song lyric, but unfortunately it is greedily matching too many words. Change the regex so it only matches to the first `doo` in the sequence. **This will require only one character change.**

```
lyric = 'Baby shark, doo doo doo doo doo doo'
pattern = 'Baby.+doo'
```

ii) The restriction enzyme BstXI has a recognition site CCANNNNNNTGG (where N represents any nucleotide). Will this enzyme cut this sequence: CTGTCCATGCAAATGGTGCTAGGGTCGATCGTG? Make a regex to find out.

iii) Write a raw string to match \\\\4 in 123\\\\456. Check it works using a regex.

iv) Using a regex, split the string below into a list. For some reason the delimiter to use is a changing number of consecutive non-alphanumeric characters:

```
John---Paul===George:::::Ringo
```

v) Pop culture is not a fan of good punctuation. Remove the full stops from the name `will.i.am` using a regex.