Lab-03

Table of Contents

Overview

I'd encourage you **not to use libraries** (*import* etc) for the exercises in this lab. For these exercises; just *code it yourself*.

The idea is to get you practicing, using:

- prompts to receive user-inputs, then;
- single variables to store user input, and then;
- string output to do something to the data.

…then:

- recording the same data into aggregate data structures, known as *lists* [] then;
- storing these lists, which might contain lists, onto other lists.
- accessing rows and columns of lists of lists etc.

… not by taking a Python package off the shelf and coding it in a few lines. We will enter this level of abstraction (using packages) later in the course, and I'll show you some powerful packages you can use to make doing this kind of thing easier, but for now its ok to climb up the hill a bit and feel the gradient of what it is to learn to code. Using a package is all well-and-good, but programming, anyway, is always a mixture of climbing up the hill (working on your own code) and sliding down the hill (using packages).

Finally, after you tidy your scripts up, you should thinking about the idea of 'persistence' of program data states outside of RAM (Random Access Memory, which is relatively expensive, and yet it cannot store variable states without power being fed into the computer). Alternatively, 'outside of RAM' can be thought of as 'disk' memory, a place where we can store program states via the notion of files (or databases, if we like).

***Tips*:** You could probably get through all the exercises by using (for the earlier exercises) simple variables of different types, type conversions, which we have covered, the *input()* method in Python3, and the simple *print()* method, etc. Then (for the latter exercises), all of the above, in addition to the use of lists of data (and you might also want to know that in Python3 its possible to store single variables in lists, along with lists stored on the same list).

Exercise 1: simple variables storing user-input associated with schema columns

Consider this data *'schema'*:

Forename	Surname	Height	Weight	Age

Table 1: A data 'schema', which is a word used to refer to the arrangement of data.

Write a script *data_prompter_1.py* that prompts the user to input some data according to this schema, and then outputs the following to the console:

Dear **Richard Holden**, you are **49**, **5** feet **11** inches, **61.5** kg and you were born in ***1973***. Notice the bold date ***1973;*** this is *calculated* from the data and you could do this from subtracting the age from the current year which (I think) is *2022*.

… where the **bold** values are derived from variables that hold the values associated with the respective user-input prompts.

string	string	int, int	float	int
Forename	Surname	Height	Weight	Age
Richard	Holden	5, 11	12.0	49

Table 2: poorly designed schema with included data type association and one row of data.

By the way, the data corresponds to the schema, which generally should associate with data types, if things were a little more 'proper' [1].

[1] Another *by the way*: the design of this schema is absolutely *terrible*, but this **doesn't** concern us too much here. And its deliberate…you will learn how to properly design data schema (schemata?) in the **Database Development** module. No point getting too 'anal' here, either; we are focusing on programming some fairly simple scripts. Furthermore, the real world is, unfortunately, brimming with poorly formatted data. If data is the '*new oil*', then it seems most of it is fairly difficult to drill into (from a programming perspective) because of poor data design. In fact, while it is terrible, it's actually much better than the format of a lot of data kept here https://www.data.gov.uk/, for example, which is the UK Government's data website. If you look at some data there you might (depending on your political point of view) conclude that making a mess of things is a

Exercise 2: aggregate variables (lists) storing 'rows' of data input

Write a script *data_prompter_2.py*. This time, include altered script code from Exercise 1a in order to prompt the user to input rows of data, each time in a while loop, until there are *N* rows of data, like this:

string	string	int, int	float	int
Forename	Surname	Height	Weight	Age
Richard	Holden	5, 11	12.0	49
Steve	Davis	6, 1	11.0	65

Table 3: poorly designed schema with included data type association and two rows of data.

N = 2 is fine. Sometimes laziness is very sensible. Well, at least don't make N too big otherwise that's you inputting quite a bit of data each time you run the program. When data has been input, store the following kind of output in a list:

Dear **Richard Holden**, you are **49**, **5** feet **11** inches, **61.5** kg and you were born in ***1973***. Notice the bold date ***1973;*** this is *calculated* from the data and you could do this from subtracting the age from the current year which (I think) is *2022*.

...and then only display the contents of the list after the while loop has ended, at the end of the program.

Exercise 3: 'accessing' list data with simple variables

Prompt the user to:

- enter a zero-indexed row item – i.e., in a [0, N-1] range. When they have done this, print the associated data row to the console.
- enter a zero-indexed column item – i.e. in the range [0, 4] (there are only 5 columns, as is determined by the schema). When they have done this, print the associated data column to the console, for every row in your list-of-lists.

Exercise 4

Again, you should have some 'header' comments at the top of your files.py:

transferable skill! Anyway, regardless of if it is messy or not, getting your hands on some data that **interests you** is a good way to start thinking of your own programming projects, so I'd encourage you to think about what interests you and search for some available data.

```
# SCRIPT NAME: <>
# DESCRIPTION:
# USAGE:
# USAGE EXAMPLE:
```

Exercise 5

Take a look at the following link:

https://softwareengineering.stackexchange.com/questions/365762/what-is-data-persistence-in-the-context-of-software-engineering

… at some other point during this programming course we will look at how to store data (values/states of variables) onto disk, so it can be *loaded* into RAM in a less tedious way than relying on user-inputs, to represent the same state the program was in when the said data was stored (a.k.a. being 'written' to disk) at a previous point in time, when the program was running.

~~~~
~~~~