Coding in Action Lab I

Instructions

The following problems are carefully laid out, we suggest you to solve them in the given order.
The point value of each problem is indicated next to its title. The maximum total score is 21 points.
Got a question? You can ask your peers, search the Web, use `man`, etc.
Collaboration is permitted and encouraged, but plagiarism is not. The latter includes using AI tools, such as ChatGPT. Should you get caught copying or using an AI tool, you will fail the exam.
The Teacher reserves the right to question you, should he have doubts about the authenticity of your code.

Submission

Turn in your assignment in your Git repository, following the submission procedures for all your exercises. Make sure you have the appropriate permissions on your files and directories. Only the work inside your repository will be evaluated.
Do not hesitate to double check the names of your files to ensure they are correct. Do not leave any additional file in your directory other than the one specified in the box.

Assessment

Your work will be checked and graded by a program called Moulinette. Moulinette is very meticulous and strict in its evaluation of your work. It is entirely automated and there is no way to negotiate with it. So if you want to avoid bad surprises, be as thorough as possible. Moulinette is not very open-minded.
If your program contains a syntax error, the score for that problem will be 0. To obtain the maximum score you have to do exactly as requested: define all functions as indicated and precisely reproduce the examples, for the specified test cases. Beware that the examples could very well call for details that are not explicitly mentioned in the problem statement, so examine them thoroughly.

Watch out! This document could potentially change up before submission.

Problem 00: `run_script` (1 point)

The file IT_products.csv, to be found in the data folder, reports the prices of IT products sold by an IT retail store. The Python script read_and_print.py must be passed a file path, and it reads the data contained in that file as a Python list and prints the list on screen.

Assignment:

> Write a bash script run_script.sh to run read_and_print.py from the Command Line Interface (CLI), assuming that read_and_print.py and data are in the same folder.

Turn-in directory:	ex00/
Files to turn in:	run_script.sh

Example:

```
42~ > ./run_script.sh
['Lightning Charging Cable,14.95', '34in Ultrawide Monitor,379.99',
'Bose SoundSport Headphones,99.99', 'Flatscreen TV,300.0', '27in FHD Monitor,149.99',
'iPhone,700.0', 'ThinkPad Laptop,999.99', 'Macbook Pro Laptop,1700.0',
'Google Phone,600.0', 'Wired Headphones,11.99', '20in Monitor,109.99',
'LG Dryer,600.0', 'AAA Batteries (4-pack),2.99', 'USB-C Charging Cable,11.95',
'AA Batteries (4-pack),3.84', 'Apple Airpods Headphones,150.0', 'Vareebadd Phone,400.0',
'LG Washing Machine,600.0', '27in 4K Gaming Monitor,389.99']
42~ >
```

Problem 01: `get_lines` (2 points)

We now want to process the data contained in that file. To this end, we need to define a few functions. Let us start with a simple one.

Assignment:

> Taking inspiration from read_and_print.py, define a Python function named `get_lines` that takes a file name as argument and returns the content of that file as a list, formatted as in read_and_print.py: the ith element of the list is the ith line of the file, stripped of any trailing "new line" characters.
> Write a program that asks the user to insert the file name and prints the output of `get_lines` to screen.

Turn-in directory:	ex01/
Files to turn in:	get_lines.py

Examples:

```
42~ > python3 get_lines.py
Insert the file name: data/IT_products.csv
['Lightning Charging Cable,14.95', '34in Ultrawide Monitor,379.99',
'Bose SoundSport Headphones,99.99', 'Flatscreen TV,300.0', '27in FHD Monitor,149.99',
'iPhone,700.0', 'ThinkPad Laptop,999.99', 'Macbook Pro Laptop,1700.0',
'Google Phone,600.0', 'Wired Headphones,11.99', '20in Monitor,109.99',
'LG Dryer,600.0', 'AAA Batteries (4-pack),2.99', 'USB-C Charging Cable,11.95',
'AA Batteries (4-pack),3.84', 'Apple Airpods Headphones,150.0', 'Vareebadd Phone,400.0',
'LG Washing Machine,600.0', '27in 4K Gaming Monitor,389.99']
42~ >
```

```
42~ > python3 get_lines.py
Insert the file name: data/non_existing_file.csv
Traceback (most recent call last):
.
.
.
FileNotFoundError: No such file or directory: 'data/non_existing_file.csv'42~ >
```

Problem 02: `get_prices` (3 points)

Each line of IT_products.csv contains two values: the name of the product and its price, separated by a comma. To use the data, we need to split the line and separate the two values.

Assignment:

> Define a function named `get_prices` that takes as argument the name of the file containing products with their prices, and returns the products and the prices in two different lists. The two lists must be consistent, meaning that the ith price must be the price of the ith product. While the product names should be strings, the prices should be numbers.
> Write a program that: (i) asks the user to insert the file name, terminating if the file does not exists; (ii) calls `get_prices` on the file; (iii) prints a "slice" composed of the first 3 elements of the two obtained lists; (iv) prints all products with their prices, one per line.

Turn-in directory:	ex02/
Files to turn in:	get_prices.py

Examples:

```
42~ > python3 get_prices.py
Insert the file name: data/IT_products.csv
['Lightning Charging Cable', '34in Ultrawide Monitor', 'Bose SoundSport Headphones']
[14.95, 379.99, 99.99]
Lightning Charging Cable 14.95
34in Ultrawide Monitor 379.99
Bose SoundSport Headphones 99.99
Flatscreen TV 300.0
27in FHD Monitor 149.99
iPhone 700.0
ThinkPad Laptop 999.99
Macbook Pro Laptop 1700.0
Google Phone 600.0
Wired Headphones 11.99
20in Monitor 109.99
LG Dryer 600.0
AAA Batteries (4-pack) 2.99
USB-C Charging Cable 11.95
AA Batteries (4-pack) 3.84
Apple Airpods Headphones 150.0
Vareebadd Phone 400.0
LG Washing Machine 600.0
27in 4K Gaming Monitor 389.99
42~ >
```

```
42~ > python3 get_prices.py
Insert the file name: data/non_existing_file.csv
Sorry, the specified file does not exist
42~ >
```

Problem 03: `filter_products` (3 points)

We are now ready to start processing the data. We begin with a simple and frequently needed task: filtering the items based on whether they do or do not satisfy some condition.

Assignment:

> Write a program that asks the user to insert the file name, calls `get_prices` on the file, and creates the following two files in the data folder:
>
> - cheap_products.csv, containing all products costing less than 100 dollars, in the same format of the original file.
> - expensive_products.csv, containing all products costing at least 100 dollars, in the same format of the original file.

Turn-in directory:	ex03/
Files to turn in:	filter_products.py

Example:

```
42~ > python3 filter_products.py
Insert the file name: data/IT_products.csv
Files created!
42~ > cat data/cheap_products.csv
Lightning Charging Cable,14.95
Bose SoundSport Headphones,99.99
Wired Headphones,11.99
AAA Batteries (4-pack),2.99
USB-C Charging Cable,11.95
AA Batteries (4-pack),3.84
42~ > cat data/expensive_products.csv
34in Ultrawide Monitor,379.99
Flatscreen TV,300.0
27in FHD Monitor,149.99
iPhone,700.0
ThinkPad Laptop,999.99
Macbook Pro Laptop,1700.0
Google Phone,600.0
20in Monitor,109.99
LG Dryer,600.0
Apple Airpods Headphones,150.0
Vareebadd Phone,400.0
LG Washing Machine,600.0
27in 4K Gaming Monitor,389.99
```

Problem 04: `search_product` (3 points)

Another frequently needed task is searching for a specific item in a sequence. The most straightforward approach to the search problem is looking at all elements of the sequence one by one until a match is found or until all elements have been inspected.

Looping through all elements of a sequence in search of a specific value is called a linear search. In the worst case, it requires comparing the sought value with all elements of the list. If the list contains n elements, this means making n comparisons. We say that the cost of a linear search grows linearly with n, or that the complexity of a linear search is $O(n)$.

Assignment:

> Write a program that: (i) asks the user to insert the name of a product `p`; (ii) loops through the list of products returned by `get_prices` until it finds `p` or it reaches the end of the list; prints the index of `p` in the list and its price, or the message "Sorry, we do not sell that product".

Turn-in directory:	ex04/
Files to turn in:	search_product.py

Examples:

```
42~ > python3 search_product.py
What product do you need? iPhone
Product iPhone found at index 5, the price is $700.0

42~ > python3 search_product.py
What product do you need? iPad
Sorry, we do not sell that product
```

Problem 05: `binary_search` (6 points)

If the list is sorted, searching for an item can be much faster than the previous linear search. Suppose that you need to find where Chapter 6 starts in a book of 256 pages:

- If you open the book at the middle (p.128) and find yourself in Chapter 8, you immediately know that Chapter 6 is in the first half. You can ignore the other half of the book.
- Now, if you open the book at the middle of the first half (p.64) and find yourself in Chapter 3, you know that Chapter 6 starts between p.64 and p.128, that is, in the second quarter of the book. You can ignore the other three quarters.

By iterating this process, you need to look at, at most, 8 pages to find where Chapter 6 begins (why 8?). This is called a binary search: it makes good use of the list being sorted to cut the problem size in half at each step, so that the complexity of searching an item becomes $O(\log n)$. Check this video for more, but be careful: they use a different programming language and a recursive solution.

Assignment:

Write a program that: (i) reads the list of products from data/sorted_products.csv using `get_prices`; (ii) asks the user to insert the name of a product `p`; (iii) if p is in the list, prints the number of comparisons needed to find `p` and the price of `p`; (iv) otherwise, prints the message "Sorry, we do not sell that product". The program must implement a binary search, making use of the following functions:

- `choose_half` must take as arguments a list `l`, a value `x`, and two indices `i` and `j`; it must compare `x` with the element `z` in position `k=(i+j)/2` of `l`, returning: `k,k` if `x` is equal to `z`; `i,k` if `x` is less than `z`; `k+1,j` if `x` is greater than `z`.
- `binary_search` must take as arguments a list `l` and a value `x`; to search for `x` in `l`, it must: (a) initialize `i` and `j` to the first and last index of `l`; (b) repeatedly call `choose_half`, updating the values of `i` and `j` at each iteration to the values returned by `choose_half`; (c) stop the iteration when `j`$\leq$`i`, returning `j` and the number of times `choose_half` has been called.

Turn-in directory:	ex05/
Files to turn in:	binary_search.py

Examples:

```
42~ > python3 binary_search.py
What product do you need? iPhone
Product iPhone found after 4 comparisons, the price is $700.0

42~ > python3 binary_search.py
What product do you need? ThinkPad Laptop
Product ThinkPad Laptop found after 2 comparisons, the price is $999.99

42~ > python3 binary_search.py
What product do you need? iPad
Sorry, we do not sell that product
```

Problem 06: `build_dictionary` (3 points)

The binary search used in Problem 05 requires the products being sorted. If your sequence is not already sorted, sorting it comes at a price: the best possible sorting algorithm runs in time $O(n \log n)$, which (with an over-simplification) is the cost of finding $O(\log n)$ elements with a linear search. Sorting the sequence and using a binary search is convenient if at least one of the following conditions holds:

- You plan to perform a relatively high number of searches.
- In your application, it makes sense to consider an "offline" phase, during which you can sort the sequence with no hurry, and an "online" phase, during which you need extremely quick answers.

Another option to make repeated "queries" to a dataset extremely fast is building a data structure that is optimized to perform look ups – a fancy name for the process of finding a piece of information in a database. In particular, when you want to find the value (in the example, the price) associated to a specific key (the product), you can use what Python calls a dictionary. A dictionary is a mapping key→value that allows getting the value associated to a given key in time $O(1)$, that is, in some constant (and very small) time that does not depend on the size of the dictionary.

Assignment:

> Write a program that: (i) transforms the lists returned by `get_prices` into a dictionary product→price; (ii) prints the first 2 "items" of the dictionary; (iii) asks the user to insert the name of a product `p`; (iv) making use of the above dictionary, prints the price of `p` or the message "Sorry, we do not sell that product".

Turn-in directory:	ex06/
Files to turn in:	build_dictionary.py

Examples:

```
42~ > python3 build_dictionary.py
[('Lightning Charging Cable', 14.95), ('34in Ultrawide Monitor', 379.99)]
What product do you need? iPhone
The price is $700.0

42~ > python3 build_dictionary.py
[('Lightning Charging Cable', 14.95), ('34in Ultrawide Monitor', 379.99)]
What product do you need? iPad
Sorry, we do not sell that product
```

Bonus problem: `compare`

This is a bonus problem that will not contribute to your final score. Solve it and send the solution to the Teacher if: you are interested in a thesis that involves coding and/or data analysis; you want to join the 42 Programming School; you want to get a better understanding of the topic in view of module 2 of the Coding in Action Labs.

Write a program that compares the three approaches seen in the previous problems:

- Find the average time needed for a linear search.
- Find the average time needed to sort the list and for a binary search.
- Find the average time needed to build the dictionary and to access the value associated to any given key.
- Discuss these numbers and the circumstances that make one of these options preferable.

Suggestions: you can use the `time` Python package; you should find the average time over a huge number or searches, over sufficiently large sequences; you might want to see how the numbers change as the size of the sequence increases.