Review Activity

Setup

- Download and unzip the provided code:
  - :code: review2.zip
- If needed, add this file to your `data` folder:
  - :code: fulltracks.json

Instructions

- Work together to complete the review exercises below
- After each exercise, run the corresponding test function
  - In VS Code, click the ":material-play:" button
  - In Thonny, run the shell command:

```
!pytest -q -k test_ch09
```

Chapter 9: Sequences

- String and list methods
- The slice operator (`[:]`)
- Splitting/joining strings

!!! question "Exercise 9 (`ch09.py`)"

````
**Function 1: `indent_stars(text)`**

Write a function that processes a multi-line string representing notes.
For each line in the string, if the line starts with `"* "`, add four spaces to
the beginning of that line.
Return the updated string with the appropriate indentation applied.

For example, given the string:
```
Quiz 5
* Chapter 9: Sequences
* Chapter 10: File I/O
Quiz 6
* Chapter 11: Nested Data
* Chapter 12: Recursion
```
Return the string:
```
Quiz 5
    * Chapter 9: Sequences
    * Chapter 10: File I/O
````
````

```
Quiz 6
    * Chapter 11: Nested Data
    * Chapter 12: Recursion
```

Function 2: `unindent_stars(text)`

Write a function that does the opposite as before:
for each line that starts with <code>"   * "</code>, remove the four leading spaces.

Chapter 10: File I/O

- Reading/writing files
- Command-line arguments
- The `csv` module

!!! question "Exercise 10 (`ch10.py`)"

Function 1: `search_file(path, term)`

Write a function that searches a file for a specified term (a word or short phrase).
The function should return the line number and column number of the first occurrence.

For example, given the file contents:
```
Practice makes progress.
Time and tide wait for none.
```
The word `tide` is on line 2, column 10.

Function 2: `strip_file(path)`

Also write a function that reads a text file, removes trailing whitespace (spaces and tabs) from the end of each line, and overwrites the file with the updated content.

Chapter 11: Nested Data

- Lists of lists
- Dicts of dicts
- The `json` module

!!! question "Exercise 11 (`ch11.py`)"

````
**Function 1: `print_stats(label, strings)`**

Write a function that prints statistics about the lengths of strings in a set.
The statistics include the minimum, median, mean, and maximum string lengths.
Also print a label before the statistics to identify the output.

For example, `print_stats("Letters:", {"A", "BB", "CCC", "DDD"})` would print:
```
Letters: min = 1.0, median = 2.5, mean = 2.2, min = 3.0
```

**Function 2: `def track_stats(tracks)`**

Write a function that processes the `fulltracks.json` data from Last.fm and
creates three sets:
one containing all track names, another with all artist names, and a third with
all album titles.
After building the sets, call the `print_stats()` function for each set, using the
labels
`"Track names:"`, `"Artist names:"`, and `"Album titles:"` to identify the output.
````

Chapter 12: Recursion

- Base case(s)
- Recursive call
- Keeping track

!!! question "Exercise 12 (`ch12.py`)"

````
**Function: `search_json(data, term, expr="data")`**

Write a function that searches for a given term in a JSON-like object (a dict,
list, or string).
The function should return a Python expression (as a string) that leads to the
term if found, or `None` if not found.

Some of the code is already provided.
The function uses `isinstance()` to check if `data` is a string, dict, or list.
If `data` is a string, the function searches the string for the term.
If `data` is a dict, the function recursively searches each key-value pair of the
dict.
If `data` is a list, the function recursively searches each index-value pair of
the list.

For example, given the data:
``` json
{
    "name": "Alice",
````
````

````
    "address": {
        "city": "Wonderland",
        "zipcode": "12345"
    },
    "hobbies": ["reading", "gardening", "coding"]
}
```
Searching for the term `"Wonderland"` would return:
```
'data["address"]["city"]'
```
````
````