CSCI 220L

Lab 9

Learning Objectives

- Practice writing, documenting, and testing functions.
- Read and write data in files.

Guidelines

Now that you understand functions, you can create modular programs that more closely resemble a real piece of software. Here is what you should keep in mind while working on this lab:

- Each function should include a short docstring that explains what it does.
- Whenever possible, call your existing functions instead of copying their code. Remember the DRY principle—"Don't Repeat Yourself".
- The `input()` and `print()` functions should not be used in place of parameters and return values.
- Include a void function `test()` to test your solutions. Some problems will ask you to add test cases to this function. A test case is a call to your function that has a hardcoded input (chosen by you) and that displays its output on the console so you can verify it. For example, your test cases for `mean()` might look like this:

  ```
  print("Testing mean():")
  print("One number:", mean([12]))
  print("Two numbers:", mean([5.5, 10]))
  ```

  If you're a fan of f-strings, you might be interested in the new debug specifier "=" added in Python 3.8 that makes this even more concise to write.
- You don't need to include a `main()` function, though you may use one for your own debugging.

Activities

Create a file `lab9.py` containing the following functions.

1. Averages

   (a) Python has a built-in function `sum(nums)` that returns the sum of a list of numbers. Write your own function `mean(nums)` that accepts a list of numbers and returns their mean. You may assume the mean of an empty list is zero. Include at least two test cases for `mean()`.

   (b) Write a function `mean_str(nums)` that accepts a list of strings representing numbers and returns their mean rounded to two digits. An example input might look like `["10", "15.5"]`. Include at least two test cases for `mean_str()`.

2. Usernames

   (a) Write a function `get_username(name)` that accepts a full name as a string and returns the corresponding username in this format: last name + first initial + middle initial. The username should be all lowercase. For example, "Homer Jay Simpson" should produce "simpsonhj".

(b) Write a void function `usernames(names)` that accepts a list of full names and converts each name to its corresponding username. The function should modify the input list, not return a new one. Include at least two test cases for `usernames()`.

Note: Think carefully about how to test this function. You can't print it directly because it has no return value. What else can you print to show that the function worked?

3. Line Numbering

Write a void function `number_lines(file)` that accepts a filename, reads through the file, and prints each line with its line number in front of it. For example:

```
1. To be, or not to be, that is the question:
2. Whether 'tis nobler in the mind to suffer
3. The slings and arrows of outrageous fortune,
...
```

Include at least one test case for `number_lines()` and a corresponding .txt file to open. You may use any (short) .txt file as your test file.

4. 3D Modeling Math

Many 3D modeling programs (Blender, Maya, AutoCAD, etc.) allow you to export your models to a file so that they can be read by other programs. In this exercise, you will create a program to find the surface areas (and average surface area) of a set of sphere models.

The modeling software you use can export your work to a special `.txt` file, where each line of the file has the following format. A sample file `planets.txt` is attached.

```
Sphere (x=___, y=___, z=___, radius=___, texture=___)
```

Write a function `models(infile, outfile)` that reads the infile (which is in the above format), calculates the surface area of each sphere, writes the surface areas to the outfile, and returns the average surface area. The specific format of the outfile is up to you. You may wish to write one or more helper functions to keep your solution organized and elegant.

Finally, include a test case for `models()`. To verify that it works, your code should both display the return value and open the outfile to display its contents (you may choose to display only the first few lines to keep your test output readable).

Upload `lab9.py`, along with any test files, to the OAKS dropbox before the deadline. Make sure you have most of the exercises completed before your lab meeting.