

1. Go to the class web page under Labs and download **lab05_files.zip**. Unzipping this file will create a new folder named **lab05_files**. Put it where you normally keep your other Python files. This folder contains several supporting files for the lab. Open the file **startcode_lab5.py** in IDLE and use that as your starting point. You will need to uncomment and edit the appropriate `os.chdir` line at the top of the file, depending on where you put the **lab05_files** folder on your computer. In general, when writing programs to process files, you should always use `os.chdir` at the top of your code to set your working directory, so that Python will be able to find your files whenever you load your code into IDLE.
2. The **currency()** program converts an amount of money in U.S. dollars to several foreign currencies and prints out the equivalent amounts, but in a messy way. Run the program with an input amount of \$75 to see the output that it produces. Then rewrite the `print` statement as shown below using an f-string to improve the appearance of the output, so that the currency amounts line up neatly in a single column. Next, try changing the formatting code for `newAmount` (the part after the colon) to `<12.2f` or `^12.2f` or `7.2f` or `15.3f` to see the resulting effects.

```
print(f"{newAmount:12.2f} {name} ({code})")
```

3. Try out the **countletters()** program on the file `haunting.txt`. Currently the program counts just vowels. Rewrite this program so that it reports the letter counts for all 26 letters of the alphabet, using a for-loop! Hint: define a new string variable called `alphabet` containing all 26 uppercase letters of the alphabet, then loop through it letter by letter, reporting how many times each letter appears in the content string. How many E's or Z's are there in *Moby Dick* (`moby.txt`)? How about *Huckleberry Finn* (`huckfinn.txt`) or *Alice in Wonderland* (`alice.txt`)?
4. Write a program called **wordlen()** that asks the user for a filename and calculates the average length of all of the words in the file, rounded to three decimal places. To do this, have your program read in the file content as a string and split it into a list of all of the words. Then use a for-loop to cycle through the word list, adding up the individual word lengths. From this you can calculate the average length. For example:

```
>>> wordlen()
Enter filename: haunting.txt
The average word length is 5.048 characters
```

5. Write a program called **linenums()** that asks the user for a filename and prints out each line of the file preceded by its line number and a colon, starting from 1, as shown below. Since it needs to process the data line by line, your program should call `readlines()` instead of `read()` to read in the data as a list of lines, instead of as a single string. Use a for-loop to cycle through the lines one at a time. To prevent the output from being double spaced, you can use `s.strip()` to remove the trailing newline character (`\n`) from the end of a string `s`. For example:

```
>>> linenums()
Enter filename: haunting.txt
1 : No live organism can continue for long to exist
2 : sanely under conditions of absolute reality; even
3 : larks and katydids are supposed, by some, to
4 : dream. Hill House, not sane, stood by itself
...
```

6. The file **airportData.txt** contains information about airports. Each line stores information about a single airport, and is divided into several fields (separated by colons) containing the airport's 3-letter code, time zone offset, map coordinates, and location. Write a program called **airports()** that prints out the 3-letter airport code and location, as shown below. Hint: use `line.split(":")` to break a line into a list of its component strings based on colons instead of spaces as separators. Then grab the appropriate component string from the list.

```
>>> airports()
ABQ Albuquerque, New Mexico
ATL Atlanta, Georgia
BNA Nashville, Tennessee
BOS Boston, Massachusetts
DCA Washington, D.C.
...
```

7. The file **peopleData.txt** contains information about people. Each line stores a person's name, their height in feet and inches, and their weight in pounds. Write a program called **people()** that reads in everyone's information and prints it out as shown below, including their equivalent weight in kilograms in parentheses. Hint: one kilogram equals 2.2 pounds, so to convert pounds to kilograms, just divide by 2.2. You will need to use the `float` function to convert a string such as "170.2" to a number before you can convert it to kilograms. Use f-string formatting to reproduce the output exactly as shown below:

```
>>> people()
Fred is 6 feet 1 inches tall and weighs 170.2 pounds (77.36 kg)
Susan is 5 feet 9 inches tall and weighs 155.3 pounds (70.59 kg)
Oliver is 4 feet 11 inches tall and weighs 143.5 pounds (65.23 kg)
...
```

8. The three files **Movies.txt**, **Viewers.txt**, and **Ratings.txt** contain information from the MovieLens movie ratings database. Each line in **Movies.txt** consists of two fields separated by a | character: the movie ID#, and the movie title/year. Each line in **Viewers.txt** describes an individual movie viewer, and consists of four fields: the viewer's ID#, their age, their gender, and their occupation. Each line in **Ratings.txt** corresponds to a particular viewer's rating of a particular movie: field 1 is the viewer ID#, field 2 is the movie ID#, and field 3 is the viewer's rating of that movie on a scale of 1-5. Open each of these files in IDLE and examine their contents and format.
9. Write a program called **movie()** that asks the user for a movie ID# in the range 1-1682, and then prints out the title and year of the movie. Your program should read the lines from **Movies.txt** into a single list, using the method `readlines()`, and then loop through the list one line at a time, using an if/else to check if the ID# of the current line matches the ID# input by the user. When a matching ID# is found, the information for that movie is then printed out. You can use the `directors()` program that we wrote in class as a guide in writing this program and the others below.

```
>>> movie()
Enter movie ID (1-1682): 50
Star Wars (1977)
```

10. Extend your **movie()** program so that it also analyzes the ratings information in **Ratings.txt**, and prints out the movie's overall average rating rounded to one decimal place, and the total number of reviews, as shown below.

```
>>> movie()
Enter movie ID (1-1682): 50
Star Wars (1977)
Average rating is 4.4 stars, based on 583 reviews
```

11. Write a program called **viewers()** that calculates the average age of all of the viewers in **Viewers.txt**, and then prints out this information rounded to two decimal places, in the format shown below:

```
>>> viewers()
Average viewer age is NN.NN years old
```

12. Python's built-in functions `min` and `max` can be used to easily find the minimum and maximum values of a list of numbers. For example, if `nums` is the list `[6, 10, 3, 4, 9, 2]`, calling `max(nums)` would give 10 and `min(nums)` would give 2. Using `min` and `max`, modify **viewers()** so that it also determines the ages of the youngest and oldest viewers in the database. Hint: start with an empty list of ages and append each viewer's age to the list as you read through the lines of **Viewers.txt**. Make sure to convert the age from a string to a number before appending it to the list, otherwise you'll get inaccurate results!

```
>>> viewers()
Average viewer age is NN.NN years old
Youngest viewer is NN
Oldest viewer is NN
```

13. As a final code check, run all of your functions through the automated tester program by typing the command `testall()` at the Python prompt. Do your functions pass all of the test cases? If not, go back and fix the problems and try again, until everything passes all of the tests. You can re-run the test sequence starting from a particular function by just typing `testall(function_name)`.