CSM0120 - Programming for Scientists

Assessed Coursework 1

Samantha Pendleton

2017

Contents

0.1 Executive Summary

In this assignment I have set up scripts, using various `python` features including: `lists`, `for loops`, and `dictionaries`.

I created 7 `functions` in total: one for each question 1, 2, and 3; three `functions` were made for question 4; and a `main function`.

`Function` 1 (question 1): `percentage_bikes()` - takes in a string of Traffic as a parameter and returns the percentage of bikes from the string.

```
>>> percentage_bikes("BBCBLLMCCCSVSS")
    28.57142...
```

Listing 1: Python example for Question 1 in console

`Function` 2 (question 2): `bridge_prices()` - takes in a string of Traffic as a parameter and returns the total price of bridge toll.

```
>>> bridge_prices("BBCBLLMCCCSVSS")
    '£23.30'
```

Listing 2: Python example for Question 2 in console

`Function` 3 (question 3): `counts()` - takes in a string of Traffic as a parameter and returns a total count of all types of vehicles.

```
>>> counts("BBCBLLMCCCSVSS")
    [3, 4, 2, 1, 3, 0, 1]
```

Listing 3: Python example for Question 3 in console

`Function` 4 (question 4(a)): `read_student_data()` - takes in a file name as the parameter (e.g. "traffic_file.txt") and returns a list of counts for each line in the input file.
For example, if a teacher has collected a file where each line represents one child's traffic string (from 400 children), the output will be a list of counts for each child: so there will be a list of 400 lists inside.

`Function` 5 (question 4(b)): `write_traffic_csv()` - takes in an output file name (e.g. "output.csv") plus the list of lists (returned by read_student_data) and writes each count list as a row in the csv.
The csv will intake that list of 400 lists, and output columns (each a vehicle type) with the student data in rows.

The `list` of `lists` from 4a (return of `read_student_data()`, is the transposed using another function called `transpose` that was imported from module another module (`transpose.py`).
The returned `parameter/variable` is a list of 7 lists: each list being a vehicle of counts.

`Function` 6 (question 4(d)): `print_histogram()` - takes in the transposed data and prints histograms of this data - one histogram for each vehicle with the counts represented as `*`.

`Function` 7 (question 5): `main()` - is the `main function` demonstrating use age of other functions. Here simple strings are passed through `function` 1, 2, and 3. Whilst a `txt` is passed through function 4a to create a list of lists, which then follows through the other `functions` (creating an csv from the list of lists and transposing this data), until histograms are produced in 4d.

0.2 Technical Overview

The specification of the assignment stated to create a `function` for each question answered, I instead created separate `functions` for question 4: a, b, and d. This method of implementation choice creates a much more useable `python` script as users can use `function`: `print_histogram()` with their own list of lists.
Numerous `functions` creates reusability for `python` scripts, also allows calling/`importing` from other modules. Majority of my `functions` take in one parameter, except `write_traffic_csv()` which takes in two parameters: the aim is to have an output csv file plus a list of lists.

I used `lists` data structures frequently throughout as `tuples` are immutable (cannot be updated). `Dictionaries` were used in my `python` scripts as the key/value system is useful when calling from and updating data.

I used `for loop` numerous times in my various `functions`: making the code loop for a fixed number of times is best compared to `if` statements - `if` statements don't loop. We used `variables` to loop into as the data is even more fixed: rather than using `range()` - `for loops` also allows us to loop over a `list`.
Despite `while loops` also being loops, they are focused on `while statement == TRUE`, which is not what we are interested in with any `function` in this assignment.

The `functions` all work as planned as the data provided was successfully presented in the csv output and printed histograms. Throughout the creation process of building the scripts, I would continuously test `loops`/`functions` - at first I would print out all `function` outputs to ensure the parameters were correct. Another method knowing the code works was some in-depth testing - I conducted multiple tests in the next section.

0.3 Software Testing

I did a variety of tests on each `function` and input different parameters: I kept the testing simple as these `functions` aren't complex.

`main()` tests all the functions together: I had input parameters set up and created variables so each `function` could call each return statement.

Function 1 (Q.1): `percentage_bikes()`		
Input Parameter	Expected Output	Pass?
("BBCBLLMCCCSVSS")	28.57142...	Y
("BB")	100.0	Y
("VCCLLTCCCSVSS")	0.0	Y
(VCCLLMCCCSVSS)	NameError	Y

Function 2 (Q.2): `bridge_prices()`		
Input Parameter	Expected Output	Pass?
("BBCBLLMCCCSVSS")	'£23.30'	Y
("MCSSTBTCCC")	'£10.80'	Y
("MCSSTBTCCC")	NameError	Y
(22)	AttributeError	Y

Function 3 (Q.3): `counts()`		
Input Parameter	Expected Output	Pass?
("BBCBLLMCCCSVSS")	[3, 4, 2, 1, 3, 0, 1]	Y
("CCLBMMLMCCCVVTTSSMS")	[1, 5, 2, 4, 3, 2, 2]	Y
(22)	AttributeError	Y

I created a new file of traffic strings for the following `function` tests, called "trafficstr_test.txt".

Function 4 (Q.4a): `read_student_data()`		
Input Parameter	Expected Output	Pass?
("trafficstrings.txt")	[[1, 12, 7, 2, 1, 4, 3], ...]	Y
(22)	OSError	Y
("trafficstr_test.txt")	[[3, 27, 11, 1, 0, 3, 6], ...]	Y

Function 5 (Q.4b): `write_traffic_csv()`		
Input Parameter	Expected Output	Pass?
("output.csv", all_student_counts)	*output.csv*	Y
("output.csv")	TypeError	Y
("out_test.csv", all_student_counts)	out_test.csv* Fig: 1	Y

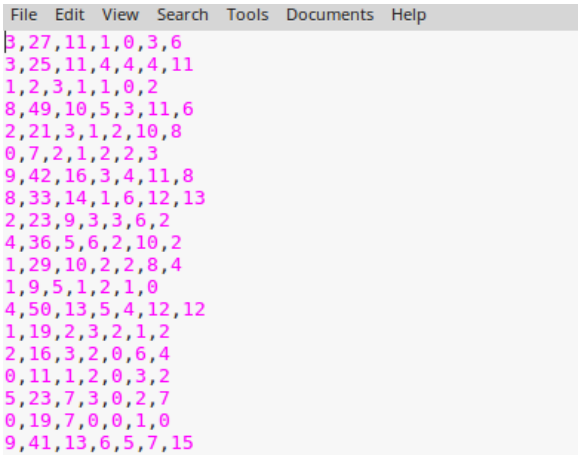

Figure 1: *Figure of the output test csv; all numbers first in each row represent Bikes.

After doing the `transpose function`, we then continue onto running the next and final function in the console.

Note: `tranpose(all_student_counts)` of the test data results in a list of 7 lists - each list in this list represents a vehicle: [[3, 3, 1, 8, 2, 0, 9..]...]

Function 6 (Q.4d): `print_histogram()`		
Input Parameter	Expected Output	Pass?
(transposed_data)	**console	Y
(22)	TypeError	Y

Note: **console output is the test data I created.

```
>>> print_histogram(transposed_data)
Bikes
0 ******************
```

```
1 ******************************
2 ***************************
3 ************************
4 ******************
5 **********
6 ***
7 **
8 *****
9 ****
10 **
12 **
Car
2 *
3 ***
4 ****
5 **
6 *
7 ***
8 ****
9 ***
10 ******
11 ****
12 ***
13 ***
14 ***
15 *****
16 *****
17 *
18 **
19 *****
20 ****
21 ***
22 **
23 **********
24 **
25 **
26 *
27 ***
28 **
29 *********
30 **
31 *
32 **
33 ********
34 **
35 ***
36 ******
37 ***
38 *
```

```
40 *
41 **
42 ***
43 **
45 ***
46 *****
47 ***
48 *
49 *
50 ***
52 *
Lorry
0 ***
1 ***********
2 **********
3 ***********
4 **************
5 ***********
6 ******
7 ***********
8 **********
9 **********
10 ********
11 **********
12 *******
13 ********
14 ****
15 ****
16 ***
20 *
Motorbike
0 ************************
1 ************************************
2 ********************
3 ********************
4 ***************
5 *********
6 *********
7 ******
8 ***
10 *
Bus
0 ********************************
1 ************************
2 ************************************
3 ********************
4 ****************
5 *******
6 ***
```

```
7 **
Taxi
0 *****
1 ****************
2 ************
3 **************
4 ************
5 ***********
6 ********
7 **********
8 ***********
9 **********
10 ***********
11 ******
12 *****
13 *****
14 ***
15 **
16 *
17 *
Van
0 ****
1 **********
2 ************
3 ********************
4 ***********
5 **********
6 ************
7 ***********
8 **************
9 ******
10 *********
11 ********
12 ********
13 ***
14 *
15 *
16 *
17 *
```

Listing 4: Python console for `function` 4d

0.4 Reflections and Future Work

I could have improved the names of some `functions` and `variables`, such as: `counts()` - I believe majority are named appropriately however `counts()` specifically could be improved for better user understanding.

I believe my code could be improved in the future if the `functions` were upgraded to be more general, for example: `percentage_bikes` could be updated to accept a new parameters: it could ask users to input other vehicles it want's percentages of; to do this, the list of bikes can be updated to include all vehicle ID and the `function` will need another parameter which it'll call from the `list`: e.g. `if parameter in vehicle_list:`

The `function` `bridge_prices()` can also be updated to having a `dictionary` that calls the different prices: currently the prices are hard-coded.

Furthermore, there could be a centralised configuration to set up vehicles: currently they are hard-coded, we can expand this someway by having a vehicle `dictionary/list` to delete vehicles from the system (list), add new ones, or update vehicle IDs (dictionary key/value).

```
1 >>> vehicle_list = ["Bike", "Boat", "Aeroplane", ...]
2 >>> vehicle_dict = {"B":"Bike", "C":"Car", "Z":"Blimp", ...}
```

Listing 5: Python examples of methods to create a vehicle system: line 1 is a `list`; line 2 is a `dictionary`.

Specifically, reflecting over my work, I could've conducted better tests: in future I'll use the `python` library `pytest` for writing unit tests.
This library can be really useful for this script as there are numerous small `functions` that aren't complex.