Lab 2 (09.09.19) CS103 Fall 2019

author john k johnstone jkj at uab dot edu
course CS103 Fall 2019

version Fall 2019

---

materials

- lab02_19fa103.pdf (this document)
- lab02_19fa103.py (with docstrings of the functions you will implement)

lab partner

Please work with your choice of lab partner.
If you have not already chosen a partner, a TA will randomly assign you one.
You will work with this partner for the next 4 labs (2-5).

purpose

- learn about docstrings (not used in HW1)
- explore functions, strings, Euclidean length
- simulate the environment of HW1
- create test data to test your functions
- learn about, and use, assertions (not used in HW1)

An aside about `in`, a valuable tool for strings

A word about a syntactical element, `in`, that I forgot to tell you about in lecture 4 and you will need today. You may discover if a string contains a character or substring using `in`.
`s1 in s2` returns True if and only the string s1 is a substring of s2.
A few examples in the Python interpreter) should illustrate:

```
>>> s = 'garden'
>>> 'r' in s
True
>>> 'z' in s
False
>>> 'den' in s
True
>>> 'dn' in s
False
```

So you could think of `in` as a function with 2 string parameters, returning a Boolean, but with a weird calling mechanism like `+`.

Preamble

In future, CS103 functions will usually include docstrings to clarify their intent.
Otherwise, the type of the parameters and return value is unclear.
It is also standard good practice to clearly and concisely describe the purpose of a function.

Here is a simplified template for the CS103 Python docstring.

```
def fnName (param1, param2):
    """1 line description of function (can be more descriptive than precise).
    >>> fnName (testInput1a, testInput1b)
    testOutput1
    Params:
        param1 (type): precise description of param1
        param2 (type): precise description of param2
    Returns: (type) precise description of return value
    """

    body of function
```

And some definitions to add to your glossary, and understand for this lab:

docstring a comment at the beginning of a function that clarifies the types of the parameters and return value, and the purpose of the function; optionally, other documentation of the function may be added

stubbed function (or stub) a function with an empty body (except for return or pass or something similar), although often with a docstring (especially in CS103); effectively a function you know you want to write but have not gotten around to yet (a common feature in the early stages of code development)

assertion a statement in code that something should be true; an assertion uses an assert statement which has the syntax in Python `assert (<condition>)` or `assert <condition>`; if the condition is true, execution of the code continues normally; if the condition is false, execution of the code is halted immediately with a diagnostic message

test data data that is fed into function calls to explore the behaviour and correctness of your functions; actually test data really involves a pair: the input data and the known output on this input.

We give at least one piece of test data in the docstring of a function (see below).
The act of examining the behaviour of your code on test data is called testing.
The construction of good test data is an important skill and art form that computer scientists must develop.

The relationship of Lab02 and HW1

There are some similarities between Lab02 and HW1. In particular, (1) the use of stubbed functions and (2) the creation of test data. I now clarify the similarities and differences.

docstrings (difference)

In Lab02, the functions contain docstrings to clarify their intent.
In HW1, the functions do NOT contain docstrings, simply because we had not introduced docstrings yet when I handed out HW1.

stubbed functions (similarity)

In Lab02, you will complete the bodies of stubbed functions.
In HW1, you will also complete the bodies of stubbed functions.

assertions (difference)

In Lab02, you will explore the use of assertions to check the validity of your function parameters.
In HW1, you will NOT add assertions to check the validity of code.
(Please don't add assertions in your HW1 code: you will get the chance to do that in future HW.)

test data (similarity)

In Lab02, you will create test data.
In HW1, you will create test data, beyond the examples that I have already provided in the tester.
Before you submit, please remember that you must remove your function calls associated with test data and any associated print statements. Our testers will expect code including just functions (no function calls, no prints). Always run the tester just before you submit to verify that you have valid code.

Exercises

Note that all answers to these exercises are to be entered into lab02_19fa103.py.
Similarly, all answers to the exercises in HW1 are to be entered into hw1_19fa103.py

string function Python exercises

First 15 minutes: alone.
Next 15 minutes: with your partner.

Here are two example problems and their solutions.

- find the first letter of a string (the function *firstLetter*)

I have provided the solution in the script. The prompt would have been this function with only the docstring and return.

- variant of this function where you verify the type of the parameter, using the type function and an assertion, and verify that the string is valid (length is at least 1) (the function *firstLetterNice*)

I have provided the solution in the script. The prompt would have been this function with only the docstring and return.

Now your exercises.

- find the second letter of a string (the function *secondLetter*)
- variant of this function where you verify the type of the parameter, using the type function and an assertion, and verify that the string is valid (length is at least 2) (the function *secondLetterNice*)
- create test data for secondLetter and secondLetterNice, including examples that illustrate their different behaviour; the test data that fails the assertion will need to be commented out (once you have tested it), so that future calls work

---

- find the last letter of a string (the function *lastLetter*)
- add assertions to verify the parameter (the function *lastLetterNice*)
- create test data for lastLetter and lastLetterNice

- determine if a character is punctuation, with assertions to guarantee that the parameter is a string of length 1 (the function *isPunctNice*);
  first find and read the *Python 3 documentation on the string module*, and in particular on string.punctuation.
- create test data for isPunctNice

Euclidean distance thought exercise

Entire 30 minutes: with your partner.

Discuss and explore Euclidean distance (similar to the exercises from page 30 of lecture 3).
In all of these questions, the length/distance is Euclidean length/distance (as the crow flies, the one you studied in high school).
There are many different ways to measure distance, but the Euclidean metric is the standard one.

Develop mathematical formulae for the following:

- length of a vector (x,y) in 2-space
- length of a vector (x,y,z) in 3-space
- distance between two points (x_1, y_1) and (x_2, y_2) in 2-space
- distance between two points (x_1, y_1, z_1) and (x_2, y_2, z_2) in 3-space

Write Python functions for each of these, as follows (I have provided docstrings). Optionally, add assertions to verify the type of the parameters.

- length2 (x,y)
- length3 (x,y)
- dist2 (x1, y1, x2, y2)
- dist3 (x1, y1, z1, x2, y2, z2)

Challenge (optional, but fun): basically, prove your formulae are correct. You know the proof of the Pythagorean theorem, which leads directly to the length formula in 2-space. Review this proof. Then provide a proof of the generalization to 3-space. If you do it right, this generalization idea will actually prove the formulae is correct in arbitrary dimensions, quite the beautiful result.

Deliverables (on Canvas lecture page, under Lab2 assignment)

Please submit the following subset of your work.

In the Python script lab02_19fa103.py (stripped down to contain only this code):

1. 2 (exactly 2) of the string functions;
   at least one of these should use assertions;
   (e.g., secondLetterNice and lastLetter)
2. the length3 function, including its docstring

In a multi-line comment at the end of this Python script:

3. the 4 formulae for Euclidean distance (one per line please)

For simplicity, you are submitting all these answers in lab02_19fa103.py, with the formulae included as comments at the bottom of the script (use the multi-line comment feature of Python).

due date: 09.10.19 (tomorrow) 11:59pm