

Homework 2: Python Interaction and Comparisons

See Webcourses and the syllabus for due dates.

General Directions

This homework should be done individually. Note the grading policy on cooperation carefully if you choose to work together (which we don't recommend).

In order to practice for exams, we suggest that you first write out your solution to each problem on paper, and then check it typing that into the computer.

You should take steps to make your Python code clear, including using symbolic names for important constants and using helping functions or procedures to avoid duplicate code.

Tests that are provided in `hw2-tests.zip`, which consists of several python files with names of the form `test_f.py`, where *f* is the name of the function you should be writing, and also some other files. Your function *f* should go in a file named `f.py` and the function itself should be named *f*. These conventions will make it possible to test using `pytest`.

`Pytest` is installed already on the Eustis cluster. If you need help installing `pytest` on your own machine, see the course staff or the running Python page.

Running Pytest from the Command Line

After you have `pytest` installed, and after you have written your solution for a problem that asks for a function named *f*, you can run `pytest` on our tests for *f* by executing at a command line

```
pytest test_f.py > f_tests.txt
```

which puts the output of the testing into the file `f_tests.txt`.

Running Pytest from within IDLE

You can also run `pytest` from within IDLE. To do that first edit a test file with IDLE (so that IDLE is running in the same directory as the directory that contains the files), then from the Run menu select “Run module” (or press the F5 key), and then execute the following statements:

```
import pytest
pytest.main(["test_f.py", "--capture=sys"])
```

which should produce the same output as the command line given above. Then you can copy and past the test output into the file `f_tests.txt` to hand in.

What to turn in

For problems that ask you to write a Python procedure, upload your code as a text file with suffix `.py`, and also upload the output of running our tests (as a text file with suffix `.txt`).

Problems

1. (10 points) [Programming] Define a Python procedure, `average4()`, which reads from `stdin` and prompts to and writes on `stdout`. When run, the `average4()` procedure prompts for 4 numbers, reads these 4 numbers in interactively (i.e., one after each prompt). (You can assume that all the inputs are in the form of positive floating point literals.) Then the procedure prints (on `stdout`) “average is ”

(without the quotation marks) and then the average of the 4 numbers (printed as a two place decimal number).

For example a sample interaction would look as follows, where the text following each question mark on a line is a user input.

```
number? 30
number? 60
number? 40
number? 50
average is 45.00
```

In order to print floating point numbers to two decimal places, you must use the following Python module, which is provided in the file `twoplaces.py` (and included in the `hw2-tests.zip` file).

```
# $Id: twoplaces.py,v 1.1 2017/01/24 20:15:53 leavens Exp $
def twoplaces(num):
    """Return a string that represents num rounded to 2 decimal places."""
    return "{:.2f}".format(num)
```

It is a good idea to run your `average4()` procedure yourself first, before running our tests, so you can see what it is doing.

Sample test inputs appear in the files `averageTens.in`, `averageLexus.in`, and `averageHouse.in`, and tests for this problem are in the file `test_average4.py`.

Remember to turn in both your file `average4.py` and the output of running our tests with `pytest`. Your code and also the output of running our tests should be submitted to webcourses as text files that you upload.

- (10 points) [Programming] Define a Python procedure, `oldmacdonald()`, which reads from `stdin` and prompts to and writes from `stdout`. When run, the `oldmacdonald()` procedure prompts for an animal name and then its sound, reading these in interactively. Then it prints (on `stdout`) a verse of the song "Old MacDonald Had a Farm" using that input.

Hint: use `sep=' '` to change the default behavior of the procedure `print` when you want it to not add a space between printing its arguments. You can also use `+` on strings in Python to concatenate two strings together; for example `"cluck"+"-"+ "cluck"` has the string `"cluck-cluck"` as its value.

For example, an interaction could look like the following, where the text following each question mark on the first two lines is a user input.

```
animal name? Chicken
animal sound? cluck
Old MacDonald had a farm,
E-I-E-I-O!
And on this farm he had a Chicken,
E-I-E-I-O!
With a cluck-cluck here,
And a cluck-cluck there.
Here a cluck, there a cluck,
Everywhere a cluck-cluck;
Old MacDonald had a farm,
E-I-E-I-O!
```

It is a good idea to run your `oldmacdonald()` procedure yourself first, before running our tests, so you can see what it is doing.

Sample test inputs appear in the files `oldmacdonald1.in`, `oldmacdonald2.in`, `oldmacdonald3.in`, and `oldmacdonald4.in` and tests for this problem are in the file `test_oldmacdonald.py`.

If your program fails our tests, check your output very carefully. Each character is important.

Remember to turn in both your file `oldmacdonald.py` and the output of running our tests with `pytest`. Your code and also the output of running our tests should be submitted to webcourses as text files that you upload.

3. (10 points) [Programming] Define a Python procedure, `isdescending()`, which reads from `stdin` and prompts to and writes from `stdout`. When run, the `isdescending()` procedure prompts for 4 integers, reading these in interactively. (You can assume that all integers are given in the form of `int` literals.) Then it prints (on `stdout`) either

`Yes, these are in strictly descending order`

or the following.

`No, these are not in strictly descending order`

The `Yes` output occurs when the numbers entered are in strictly descending order (with each number strictly less than the one preceding it) and the `No` output occurs when that is not the case.

For example, here is one interaction, producing a `Yes` output, where following each question mark on a line is a user input.

```
number? 42
number? 27
number?13
number? 7
Yes, these are in strictly descending order
```

The following is another interaction, producing a `No` output, where following each question mark on a line is a user input.

```
number? 95
number? 13
number? 13
number? 6
No, these are not in strictly descending order
```

It is a good idea to run your `isdescending()` procedure yourself first, before running our tests, so you can see what it is doing.

Sample test inputs appear in the files `isdescending1.in`, `isdescending2.in`, `isdescending3.in`, and `isdescending4.in` and tests for this problem are in the file `test_isdescending.py`.

Remember to turn in both your file `isdescending.py` and the output of running our tests with `pytest`. Your code and also the output of running our tests should be submitted to webcourses as text files that you upload.

Points

This homework's total points: 30.