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 fucttions 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 an ASCII file with suffix `.py`, and also upload the output of running our tests (as an ASCII file with suffix `.txt`).

Problems

1. (10 points) [Programming] Define a Python procedure, `sales_total()`, which reads from stdin and prompts to and writes from stdout. When run, the `sales_total()` procedure prompts for the prices of 3 pairs of shoes, reads these 3 prices in interactively (i.e., one after each prompt). (You can assume that all prices are input in the form of positive floating point literals.) It prints (on stdout) "`Sales tax: `" (without the quotation marks) and then the cost of the 6% sales tax on the total (printed as a dollar sign

with a two place decimal number) and then prints "`Total cost:  `" followed by the total cost with sales tax added (also printed as dollars).

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

```
1st price? 99.99
2nd price? 100.00
3rd price? 300.00
Sales tax: $30.00
Total cost: $529.99
```

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 `sales_total()` procedure yourself first, before running our tests, so you can see what it is doing.

Sample test inputs appear in the files `payless.in`, `fancy.in`, and `tap.in`, and tests for this problem are in the file `test_sales_total.py`.

Remember to turn in both your file `sales_total.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 ASCII text files that you upload.

2. (10 points) [Programming] Define a Python procedure, `minmax()`, which reads from stdin and prompts to and writes from stdout. When run, the `minmax()` procedure prompts for 3 integers, reading these in interactively. (You can assume that all integerss are given in the form of int literals.) Then it prints (on stdout) "`min:  `" (without the quotation marks) and a number that is a minimum of the three integers input, and next it prints "`max:  `" (without the quotation marks the maximum of the three numbers input.

   In your solution, you can use the built-in **`min`** and **`max`** functions of Python.

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

   ```
   number? 3
   number? 99
   number? 1
   min: 1
   max: 99
   ```

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

   Sample test inputs appear in the files `minmax1.in`, `minmax2.in`, `minmax3.in`, and `minmax4.in` and tests for this problem are in the file `test_minmax.py`.

   Remember to turn in both your file `minmax.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 ASCII text files that you upload.

3. (10 points) [Programming] Define a Python procedure, `isascending()`, which reads from stdin and prompts to and writes from stdout. When run, the `isascending()` 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 ascending order
```

or the following.

```
No, these are not in strictly ascending order
```

The Yes output occurs when the numbers entered are in strictly ascending order (with each number strictly greater than the one preceeding 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? 5
number? 13
number? 27
number? 42
Yes, these are in strictly ascending order
```

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

```
number? 5
number? 13
number? 13
number? 42
No, these are not in strictly ascending order
```

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

Sample test inputs appear in the files `isascending1.in`, `isascending2.in`, `isascending3.in`, and `isascending4.in` and tests for this problem are in the file `test_isascending.py`.

Remember to turn in both your file `isascending.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 ASCII text files that you upload.

Points

This homework's total points: 30.