COP 3223H — Honors Introduction to Programming with C January 19, 2017

Homework 1: Python Numeric Functions

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.
Tests that are provided in hw1-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. 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. To install pytest on your own machine, execute the following command from the shell (or terminal window):

```
pip install -U pytest
```

You can check on the installation by running the command `pytest --version`, which should say the version number of pytest you are using. See pytest.org's installation page for more details. (See the running Python page if you use cygwin.)
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`.
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 function, 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. (6 points) [Programming] Define a Python function, `aroundsun(years)`, which takes a number, `years`, and returns return the number of miles that a person that age has traveled around the sun in their lifetime.

   For this problem you may assume that the earth's orbit around the sun is circular (which is nearly true), and that the distance from the sun to the earth is 9.2955807e7 miles. The formula for the circumference of a circle is $2\pi r$, where r is the radius of the circle.

   Tests for this problem appear in Figure 1 on the following page.

```
# $Id: test_aroundsun.py,v 1.3 2017/01/14 20:01:47 leavens Exp $
from aroundsun import aroundsun
from math import pi, isclose
AU = 9.2955807e7 # astronomical unit in miles
def test_aroundsun():
    assert isclose(aroundsun(1), 2 * pi * AU)
    assert isclose(aroundsun(18), 36 * pi * AU)
    assert isclose(aroundsun(19), 38 * pi * AU)
    assert isclose(aroundsun(20), 40 * pi * AU)
    assert isclose(aroundsun(30), 60 * pi * AU)
    assert isclose(aroundsun(40), 80 * pi * AU)
    assert isclose(aroundsun(50), 100 * pi * AU)
    # oldest human lived 122 years and 164 days
    assert isclose(aroundsun(122.449315), 244.89863 * pi * AU)
    # some turtles and a clam have lived 500 years
    assert isclose(aroundsun(500), 1000 * pi * AU)
    # a bristlecone pine is thought to be about 5000 years old
    assert isclose(aroundsun(5000), 10000 * pi * AU)
```

Figure 1: Tests for `aroundsun`, found in the file `test_aroundsun.py`.

Remember to turn in both your file `aroundsun.py` and the output of running our tests. These should be submitted to webcourses as ASCII text files that you upload.

2. (8 points) [Programming] Define a Python function, `average3(n1,n2,n3)` that when given 3 numbers, `n1`, `n2`, and `n3`, returns their average.

   Recall that the average of a set of numbers is the sum of those numbers divided by how many numbers are being averaged.

   Tests for this problem appear in Figure 2.

```
# $Id: test_average3.py,v 1.1 2017/01/15 13:12:30 leavens Exp leavens $
from average3 import average3
from math import isclose
def test_average3():
    assert isclose(average3(2, 4, 6), 4)
    assert isclose(average3(50.1, 50.3, 50.2), 50.2)
    assert isclose(average3(27.0, 2.0, 1.0), 10.0)
    assert isclose(average3(50.342, 0.0, -50.342), 0.0)
    # some numbers from the Dow Jones Industrials follow
    assert isclose(average3(19897.67, 19950.78, 19888.61), 19912.353333333333)
    assert isclose(average3(15998.08, 17949.37, 19945.04), 17964.163333333333)
```

Figure 2: Tests for `average3`, found in the file `test_average3.py`.

Remember to turn in both your file `average3.py` and the output of running our tests. These should be submitted to webcourses as ASCII text files that you upload.

3. (8 points) [Programming] Define a Python function, `teaspoonsIn(gallons)` that when given a liquid volume in (US) gallons, returns the number of (US) teaspoons that is the equivalent of that number of (US) gallons.

There are 128 (US) fluid ounces in a (US) gallon, and 6 (US) teaspoons in a (US) fluid ounce.

Tests for this problem appear in Figure 3.

```
# $Id: test_teaspoonsIn.py,v 1.1 2017/01/15 00:11:37 leavens Exp leavens $
from teaspoonsIn import teaspoonsIn
from math import isclose
def test_teaspoonsIn():
    assert isclose(teaspoonsIn(1/128), 6.0)
    assert isclose(teaspoonsIn(1.0), 768.0)
    assert isclose(teaspoonsIn(0.25), 192.0)
    assert isclose(teaspoonsIn(0.5), 384.0)
    assert isclose(teaspoonsIn(10.0), 7680.0)
```

Figure 3: Tests for `teaspoonsIn`, found in the file `test_teaspoonsIn.py`.

Remember to turn in both your file `teaspoonsIn.py` and the output of running our tests. These should be submitted to webcourses as ASCII text files that you upload.

4. (10 points) [Programming] Define a Python function, `plutoweight(earthweight)` that when given a person's weight in pounds (of force) on Earth, `earthweight`, returns the number of pounds (of force) that a scale (brought from earth) would register for that person on the surface of Pluto.

   You can calculate the weight by using Newton's law of gravitation, which says that the gravitational force (which is what weight is) between two masses m_1 and m_2 separated by a distance of r meters is given by
   $$F = G \cdot m_1 \cdot m_2/r^2. \tag{1}$$
   The mass of Pluto is approximately $1.303e22$ kg. The radius of Pluto is approximately $1187e3$ meters. Here the force would be in units of Newtons (N). Newton's gravitational constant, G, is equal to $6.764e-11\ N \cdot m/kg^2$. On Earth, a pound (of weight) is equivalent to 0.453592 kg (of mass). Conversely, 1 Newton (N) of force is equivalent to 0.224809 pounds of force

   Hint: convert the given weight to a mass (in kg), then use Newton's law of gravitation to solve for the force (in N) on the surface of Pluto between Pluto and that mass, then covert back to pounds of force.

   Tests for this problem appear in Figure 4.

```
# $Id: test_plutoweight.py,v 1.2 2017/01/19 20:54:24 leavens Exp leavens $
from plutoweight import plutoweight
from math import isclose

def test_plutoweight():
    assert isclose(plutoweight(1), 0.06293732208480671, rel_tol=2e-2)
    assert isclose(plutoweight(120), 7.552478650176802, rel_tol=2e-2)
    assert isclose(plutoweight(150), 9.440598312721002, rel_tol=2e-2)
```

Figure 4: Tests for `plutoweight`, found in the file `test_plutoweight.py`.

Remember to turn in both your file `plutoweight.py` and the output of running our tests. These should be submitted to webcourses as ASCII text files that you upload.

Points

This homework's total points: 32.