Introduction to Computer Science CMSC105

Lab 1: Problems, Algorithms, and Python Functions

Please remember that you may discuss these problems with each other, but any notes that you make together *must be discarded or erased* before you start writing up your own algorithm notes and programs.

For each of the problems below, you will eventually need to develop a test suite, an algorithm, and an implementation of that algorithm as a program (which should pass your tests, as well as ours). You should address each problem by doing those steps *in order* — first build a test suite, then sketch out an algorithm, and then write your program. In this first week's lab work, you will do *only* the test suite for the hardest problem (the algorithm and program will be done as part of next week's lab).

Note that the problems listed below get harder as you go on, and that the last one is optional; if you find the first one or two easy, *please* use them to establish the habit of building your test suite, then thinking, and then programming — the CS teaching lab has a nice big table where you can work to design algorithms before programming. The practice of thinking first about the problem, then should serve you well as you move into the harder problems. You are welcome to take this approach one step further, and do Steps a and b for both of the first two problems before you touch any of the computers.

In future labs, the instructions will be more concise, and you will often be expected to come up with your own sequence of steps to approach the problem.

1. Recall the `power` function from lecture which works for values of the exponent (assumed integer) greater than 0 (*i.e.,* $\exp > 0$). Extend this function so it works for negative integer exponents as well, by taking the following steps:

   a) Come up with an example that illustrates what your extension should do, and jot it down on paper (this should be extremely easy).

   b) Look for a mathematical fact that can help solve this problem, such as those discussed in Exercises 3.1 or 3.2 of the course notes, and then write your algorithm on paper at the lab table. Show your work for Steps a and b to the instructor before moving on to c and d.

   c) Log in on one of the lab computers, set up Eclipse, and obtain the "`calculation`" project (the instructor or lab monitor will help with this). Add your example from Step a to the "doctest" comment at the start of `power.py`; also edit any other examples that should produce different results with your extension; also add a comment giving your name. You should be able to run the program and watch it fail the new and changed tests (since you haven't done Step d yet).

   d) Edit the `power` function to implement your algorithm from Step b.

   e) Use your tests to confirm that your program works. Enter more tests in the interactive mode, and watch to see that the answers are right. If you find any new tests that identify bugs in your program, add them to your test suite, and *then* debug your program. **Note** that Python will compute only the integer part of an expression for which no operand was written with a decimal point, so if you get 0 rather than 0.5 for `power(2,-1)`, try entering it as `power(2.0,-1)`.

   f) Right-click on the project name ("`calculation`") and use Team→Commit in Eclipse to submit your work when you are done, or at any other important intermediate steps.

   Note that the `calculation` project also contains a file `equation_solver.py`; ignore it until next week.

2. The region enclosed in a circle can be specified with three numbers: two giving the horizontal (x) and vertical (y) coordinates of the center, and one giving the radius (r). Develop a test suite and algorithm for this problem:

   a) Obtain the "`computational_geometry`" project through Eclipse — this will be used for the remaining problems.

   b) Run the `overlap-test-graphics.py` program, select the circle overlap option, and draw some examples of tests for this problem (note that the user interface is far from perfect, as the circles disappear when you have drawn the second one, so before you release the mouse button, you should note carefully whether or not the circles are supposed to overlap). Note that the sample question is shown in the Eclipse "Console" pane, and (until you write your own function) the sample answers are used to check for answers; if the sample algorithms do not agree (which should happen for every instance of `circle_overlap`), the different answers are listed. Using this interface and (if you wish) by just coming up with some numbers, create a full test suite for this problem and add it to the comments at the start of `circle.py` (you should be able to copy and paste the test from the Console, or you may also explore the use of "stored tests", possibly with the help of a classmate or instructor). Break as many of the "sample answers" as you can with your test suite, and add a comment after your tests about which samples you think are wrong and why. Then use Team→Commit to submit your test suite before continuing to the next step.

   c) Look for a mathematical fact that would help you distinguish overlapping from non-overlapping circles. (Do not enumerate all the points inside a circle.) Feel free to ask an instructor for help if you are having trouble.

   d) Edit the `circle_overlap` function to implement your algorithm under the `if MODE=='mine'` line, and change the definition of `MODE` (above that `if`) to say `MODE='mine'`.

   e) Test your program with your test suite, and explore additional tests by running `overlap-test-graphics.py` — if you find any new tests that identify bugs in your program, add them to your test suite and *then* debug your program.

3. As we saw in lecture, the region enclosed by a rectangular "window" with borders that are parallel to the x and y axes can be described with four numbers, giving the minimum and maximum values for x and y. Develop a test suite and algorithm to test whether a circle and rectangle overlap (i.e. if there is at least one point inside (or on the edge of) both regions)):

   a) Create a test suite for the `circle_rectangle_overlap` function in `circle_rectangle.py`, using the same procedure you used for circle overlap; once again, include a comment about which sample answers are correct — the set of samples includes between one and three that we believe are correct, and many that are wrong.

   In **next week's lab**, you will have a chance to solve this "circle-rectangle overlap" problem, but do *not* worry about doing so this week.

4. **EXTRA CREDIT:** *(Do not work on this until you have completed the work above.)* Develop a test suite for the "line segment intersection problem". The goal is to determine whether or not two line segments intersect, given the coordinates of the endpoints of two line segments (*i.e.,* one segment connecting (x1, y1) and (x2, y2), and the other segment connecting (x3, y3) and (x4, y4)). Extra credit will also be available in next week's lab session for producing an algorithm and program for this problem.

Remember: use "Team→Commit" to submit your work when you're done and at important intermediate points.