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.

Also note that different parts of this lab are due at different times: you are not expected to complete the entire project in one week.

For each of the problems below, you will 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. To encourage this approach, the test suite will be due before that program for Problems 3 and 4, and most labs for the rest of the course.

In this lab, the problems get harder as you go on; 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 (the instructor or lab monitor will help with this), get access to the files for Lab 1, and 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 \( \text{power}(2, -1) \), try entering it as \( \text{power}(2.0, -1) \).

   f) Right-click on the project name (“1-intersect”) and use Team → Commit in Eclipse to submit your work when you are done, or at any other important intermediate steps.
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) Draw a few examples of different pairs of circles. How many tests are needed to illustrate all the interesting cases of this problem?

   b) For each interesting test case from Step a, write down the x and y values for each circle's center and the value of each circle's radius. You may either estimate these values yourself, or run the overlap-test-graphics.py program, select the circle overlap option, and draw your examples. If you use the graphical interface, note that the program prints out the example function call and the answer (or answers) given by our "sample answer" programs, and copy all of your tests into the doctest comment to produce a test suite at the start of the file circle.py. Make sure you give the six parameters in the order that they are needed by the circle_overlap function. Note that the "sample answer" programs are only available when you run the lab on our lab computers.

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

   d) If you have not already done so, log in on the lab computer and add your tests to the comments at the start of circle.py. Use Team→Commit to submit your updated test suite before starting work on Step d.

   e) Edit the circle_overlap function to implement your algorithm (remove the comment about Replace the rest of this function with your answer and the entire rest of the function below that comment).

   f) 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.

   g) Submit your work when you are done, and also at any important intermediate steps.

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) Draw a few examples of different problem instances on paper. Think carefully about how to create a suite of tests that illustrates all the interesting cases of this problem.

   b) Run the overlap-test-graphics.py program and select the circle-rectangle overlap option, and enter each of your examples from Step a. Note that the program prints out the example function call and the answer (or answers) given by our "sample answer" programs. You should copy all of your tests into the doctest comment to produce a test suite at the start of the file circle_rectangle.py, once again being careful to get the numbers in the order needed by the function being tested. The set of sample answers includes between one and three that we believe are correct, and many that are wrong — how well does your test suite do at finding the broken programs?
c) Look for a mathematical fact that would help you distinguish overlapping from non-overlapping shapes, based on only the numbers from Step a. (Do not enumerate all the points within one of the shapes.) Feel free to ask an instructor for help if you are having trouble with this (or any other step).

d) Edit the circle_overlap function to implement your algorithm.

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.

f) Submit your work when you are done, and also at any important intermediate steps.

4. EXTRA CREDIT: (Do not work on this until you have completed the work above.) Develop a test suite, algorithm, and implementation of your algorithm, 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 \((x_1, y_1)\) and \((x_2, y_2)\), and the other segment connecting \((x_3, y_3)\) and \((x_4, y_4)\)). Do not enumerate all the points on the line segments (enumerating all integer points is not correct, and enumerating all rational-numbered points is not possible). You may want to read Exercise 4.2 in the course notes before starting work on this problem. As in the earlier questions, you should start by creating a test suite that lets you explore the problem, then write out notes about this algorithm on paper, and then develop and debug your program.

Remember to use “Team → Commit” to submit your work when you are done.