Programming Competition

CSE/IT 107L

NMT Department of Computer Science and Engineering

"First, solve the problem. Then, write the code."

— John Johnson

"Beware of bugs in the above code; I have only proved it correct, not tried it."

— Donald Knuth

"Openness always deserves recognition." ("Offenheit verdient immer Anerkennung.")

— Otto von Bismarck

Contents

1 Introduction

This lab is solely a problem solving lab. There is nothing to turn in.

2 Programming Competition

2.1 Form Groups

Please form groups of up to 3 students. Each group has access to a single computer. You must decide how to solve the problems and how to write the solutions in Python.

2.2 Submit Answers

To submit your answers, please notify a TA. The whiteboard will serve as the scoreboard for your lab. The TA will want to see the answer to the "scoreboard submission." If the TAs are grading another team's submission, move on to a new problem while waiting for them to be free.

Do not share the problem descriptions or the code with the other lab section. Everyone needs to have an equally fair chance of doing well in this competition.

You may not use the internet to find answers to these problems. Think! If the concepts are unclear, please get help from a TA.

2.3 Win Extra Credit

This lab will not count for its own grade. Your standing in the competition will count for extra credit on your lab final. The winners of your lab section will get 4%, the second place group will get 2%, and the group in third place will get 1% extra credit on their final lab project.

2.4 This Lab is Required

Even though this lab "only" counts for extra credit, you have to stay the lab and try your best. **You will LOSE 10% on your lab final if you do not stay and try your best.**

You may choose to do any of the following problems. However, there is no credit for incomplete or incorrect submissions.

List of Exercises

TAs will be checking your code and its output as you finish writing the exercises.

3 Exercises

Please contact a TA when you have finished solving a problem.

Exercise 3.1 (multiples.py).
Multiples of 3 and 5 (5 points)

If we list all the integers (natural numbers) below 10 that are multiples of 3 or 5, we get 3, 5, 6, and 9. The sum of these multiples is 23.

Write a program to find the sum of all the multiples of 3 or 5 below 1000.

Scoreboard submission: The sum of all multiples of 3 or 5 below 1000.

Exercise 3.2 (change.py).
Coin Change (5 points)

Write a program that determines the minimum amount of (dollar) coins and bills to be used to change a certain amount of money. That is, find the largest number of 100s, then 50s, then 20s, then 10s, etc for the change. Assume that you are changing less than $500. Assume that you are using the following denominations:

Bills: 100, 50, 20, 10, 5, 2, 1 dollars

Coins: 25, 10, 5, 1 cents

For example: the optimal change for 27.91 would be

$100: 0	**$20:** 2	**$5:** 1	**$1:** 0	**$0.10:** 1	**$0.01:** 3
$50: 0	**$10:** 0	**$2:** 1	**$0.25:** 3	**$0.05:** 1	

Scoreboard submission: contact a TA for test cases.

Exercise 3.3 (distances.py).
Distances between Points (15 points)

The distance between a pair of points (x_0, y_0) and (x_1, y_1) is equal to

$$\sqrt{(x_0 - x_1)^2 + (y_0 - y_1)^2}$$

Write a program that takes in a series of points, terminated by a blank line, then prints out all but the first point, sorted by the distance from the first point.

For example:

```
Enter a point: 0,0
Enter a point: 1,1
Enter a point: 5,5
Enter a point: 3,3
Enter a point: 100,100
Enter a point:
Result: (1, 1) (3, 3) (5, 5) (100, 100)
```

```
Enter a point: 3,5
Enter a point: 2,0
Enter a point: 7,-2
Enter a point: 3,0
Enter a point: 12,-2
Enter a point:
Result: (3, 0) (2, 0) (7, -2) (12, -2)
```

Scoreboard submission: contact a TA for test cases.

Exercise 3.4 (hands.py).
Crossing Hands (20 points)

The hands of an analog clock occassionally cross as they revolve around the dial. Write a program that determines the times at which times the hour and minute hands cross. Assume that the hands move continuously (that is, the hands do not jump between numbers – the times at which they cross may not necessarily be integer values).

Scoreboard submission Add up the hours, minutes, and seconds of all of the times that the hour and minute hands cross in one twelve-hour cycle and submit it in the format `H-M-S`, for example `30-167-421`.

Exercise 3.5 (pythagoras.py).
Special Pythagorean Triple (10 points)

A Pythagorean triplet is a set of three integers (natural numbers), $a < b < c$ such that $a^2 + b^2 = c^2$.

For example, $a = 3, b = 4, c = 5$ fits: $3^2 + 4^2 = 9 + 16 = 25 = 5^2$.

There exists exactly one Pythagorean triplet for which $a + b + c = 1000$.

Scoreboard submission: Find the product abc of that special Pythagorean triplet.

Exercise 3.6 (fractions.py).
Reducing Fractions (10 points)

Write a program that reads both the numerator and denominator of a fraction and displays the reduced fraction. For simplicity, you may assume that the numbers entered are less than 10,000.

For example:

```
Enter numerator: 4
Enter denominator: 30
Result: 2/15
```

```
Enter numerator: 25
Enter denominator: 5
Result: 5/1
```

Hints: See Appendix A to find out what a GCD is and what Euclid's algorithm is.

Scoreboard submission: Contact a TA for test cases.

Exercise 3.7 (anadist.py).
Anagram Distance (20 points)

The anagram distance between two words is how many letters must be added or removed from one of the words in order to make it an anagram of the other. For example, "dog" and "godot" have an anagram distance of 2 while "sandwich" and "witch" have an anagram distance of 5.

You can assume that input will be purely lowercase letters and will be no longer than 20 characters.

```
Enter first word: sandwich
Enter second word: witch
Result: 5
```

```
Enter first word: interwebs
Enter second word: spiderman
Result: 8
```

Scoreboard submission Contact a TA for test cases.

Exercise 3.8 (regions.py).
Contiguous Regions (25 points)

Write a program that takes in a square region and determines whether two points are connected. The region will consist of three characters: `.`, `x`, and `s`. There will be exactly two `s`'s while there can be any number of `.`'s and `x`'s. Your program will output either `connected` or `not connected`, depending on if the two `s` can be travelled between by only moving across `.`'s and only moving in cardinal directions.

Your program should know when to stop accepting output without requiring a specific "stop" command.

```
..s.
.xxx
.x..
...s
connected
```

```
....x
.s.x.
..x..
.x.s.
x....
not connected
```

Scoreboard submission Contact a TA for test cases.

Exercise 3.9 (path.py).
Finding a Path (25 points)

Write a program that finds the shortest valid path between two nodes in a graph. Each of the lines of input will be the name of a node (a single, upper-case character), followed by a

list of the nodes connected to that node. The final line of input will a single upper-case letter, designating the destination node. You must output a sequence of letters that represents the shortest path to the destination. Assume you start at the first given node. You can assume a path will always exist. There will be less than 10 nodes given. If multiple valid paths exist, any will be accepted.

For example:

```
A B D
B C D
C

A B C
```

```
A B C D
B F
D B C
F E
E

A B F E
```

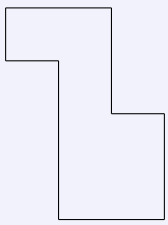
```
A B C D E
B C D E F
C D E F
D E F
E F
F

A B F
```

Scoreboard submission: Contact a TA for test cases.

Exercise 3.10 (blocks.py).
Block areas (35 points)

Write a program to take in a series of coordinate points and finds the enclosed area. The points will all be non-negative integers not exceeding 100. All angles in the shape will be 90 degrees. You can assume that no shape will be defined by more than 20 points.

For example:



Example shape.

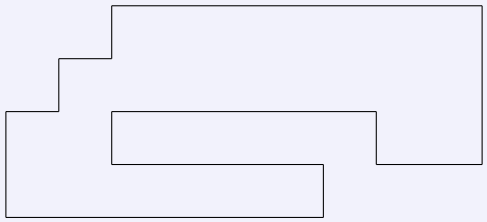
Hint: The order that the points are input shouldn't matter for your algorithm.

```
Enter number of points: 8
Enter point: 2 1
Enter point: 4 1
Enter point: 4 3
Enter point: 3 3
Enter point: 3 5
Enter point: 1 5
Enter point: 1 4
Enter point: 2 4

Area: 7
```

Scoreboard submission: Find the area of the polygon described by the following 14 coordinates.

$$(0,0)\ (6,0)\ (6,1)\ (2,1)\ (2,2)\ (7,2)\ (7,1)$$

$$(9,1)\ (9,4)\ (2,4)\ (2,3)\ (1,3)\ (1,2)\ (0,2)$$



Scoreboard submission shape.

Exercise 3.11 (dice.py).
Nontransitive dice (50 points)

A set of dice is called nontransitive if it contains three dice, A, B, and C, with the property that A rolls higher than B more than half the time, and B rolls higher than C more than half the time, but it's not true that A rolls higher than C more than half the time. Intuitively, the existence of such a set might not seem possible - If $A > B > C$, how could it be true that $C > A$? But consider the following set of dice:

- Die A has sides 3, 3, 3, 3, 3, and 6
- Die B has sides 2, 2, 2, 5, 5, and 5
- Die C has sides 1, 4, 4, 4, 4, and 4

By checking all possible die-rolls, we can see that A will roll higher than B 21 out of 36 times, and likewise B will roll higher than C 21 out of 36 times, but C will roll higher than A 25 out of 36 times. Neat, right?

Write a program that reads an arbitrary number of six-sided dice configurations followed by a single 0, and displays "Nontransitive" if there exists a subset of at least three nontransitive dice in the set, or "Transitive" if no such nontransitive subset exists. One die will be input per line, with the numbers of each side seperated by spaces. All side numbers will be between 0 and 999. The order of the input dice does not matter.

```
5 5 5 1 1 1
3 3 3 3 3 3
4 4 4 4 0 0
6 6 2 2 2 2
8 6 7 5 3 0
0

Nontransitive
```

```
1 2 3 4 5 6
7 8 9 10 11 12
10 20 30 40 50 60
100 200 300 400 500 600
0

Transitive
```

Scoreboard submission: Contact a TA for test cases.

Appendices

A Reducing Fractions – GCD and Euler's Algorithm

Reducing a fraction involves dividing the numerator and denominator by the same integers until there is nothing that divides both numerator and denominator anymore.

For example, $\frac{60}{6} = \frac{30}{3} = \frac{10}{1} = 10$. First, I divided both numerator and denominator by 2, and then by 3.

This is where the concept of a GCD comes in. GCD stands for "greatest common divisor" – given two numbers such as 60 and 6, the GCD is the greatest whole number that both numbers are divisible by. Since both 60 and 6 are divisible by 6 and since there is no greater integer than 6 that one can divide them both by, the $\gcd(60, 6) = 6$. For another example, $\gcd(40, 6) = 2$.

If you think about it more, it becomes clear that one can reduce a fraction by dividing both the numerator and denominator by the GCD of both. In essence, given a fraction $\frac{a}{b}$, the reduced fraction would be:

$$\frac{a}{b} = \frac{\frac{a}{\gcd(a,b)}}{\frac{b}{\gcd(a,b)}}$$

How do we find the GCD though? Here, a neat algorithm called Euclid's algorithm comes in. Euclid's algorithm takes advantage of the fact that when an integer a is divided by another integer b, we get a quotient q and a remainder r. This is called the Division Algorithm (see Theorem A).

[Division Algorithm] Given integers a and b, where $b > 0$, there exist unique integers q (the quotient) and r (the remainder) such that

$$a = bq + r, \text{ where } 0 \leq r < b.$$

For example, when 5 is divided by 2, we get a quotient of 2 and a remainder of 1, since $5 = 2 \times 2 + 1$. You have already taken advantage of this algorithm when you validated credit card numbers.

In Python, we can find the quotient by doing integer division and the remainder using the modulus operator %.

A.1 Euclid's Algorithm

Euclid's algorithm finds the GCD by repeatedly using the division algorithm. Given integers a and b, where $a > b$, that we want to find the GCD of, we find the quotient q and r. If the remainder is 0, b is the GCD. If the remainder is not 0, we repeat the same steps, but we let $a = b$ and $b = r$.

In essence, it goes: RCLCL a = bq + r where $0 \leq r < b$
$b = rq_2 + r_2$ where $0 \leq r_2 < r$
$r = r_2q_3 + r_3$ where $0 \leq r_3 < r_2$ So we are saying that $\gcd(a, b) = \gcd(b, r) = \gcd(r, r_2) = \cdots = \gcd(r_n, 0) = r_n$. Notice that the quotient is not used at all, only the given integers a, b and the remainder r matter.

Notice that $r_n < \cdots < r_3 < r_2 < r < b$.

It seems natural to write this recursively. In pseudocode:

```
euclids(a, b)
  if(a < b)
    return euclids(b, a)

  r = a modulus b
  if(r == 0)
    return b
  else
    return euclids(b, r)
```