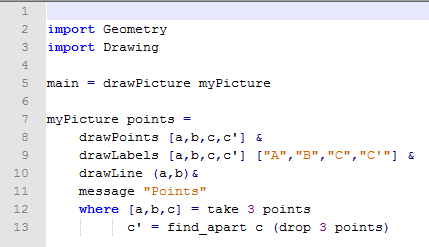
**Lesson 2: Intersections**

\*All exercises should be opened in the editor and ran in the terminal as stated in the introduction.

**Lesson 2 Part 1:**  Open the editor and then open lesson2a.hs Look at the code and write down what you understand in the code. Run the code.



Teachers need to discuss that this program has 2 auxiliary definitions in lines 12 and 13, respectively. Line 12 is similar to previous definitions, but line 13 has new concepts. The function drop is kind of the opposite of take, as drop 3 points is an infinite list with the first 3 points of list points removed. In other words, it drops [a,b,c] from points. The function find\_apart will scan that list to find the first point in the list that is apart from point c.

There is a subtle difference between two points being apart and two points being *different.* When two points are too close to each other, we may or may not be able to write a proof that establishes whether or not they are the same point. The reason is that the machine can only approximate the points to a finite precision, so we cannot know the exact position of the points. On the other hand, if the points are sufficiently far apart from each other, then we can actually prove that they are not the same point. How apart two points must be in order to be distinguishable from each other depends not only on the precision of the machine running our code but also on our tolerance to errors. These lessons use a threshold of 0.01 units to decide apartness.

Questions:

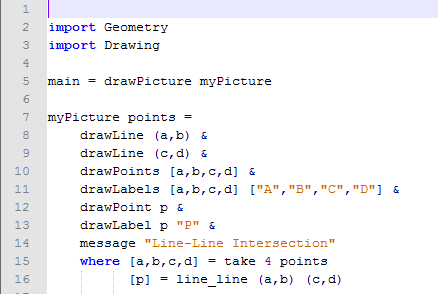
1. What does the function find\_apart do?

Gives you a point from a list that is separate from a specified point.

1. What does the drop 3 points do?

Consist of the list of points without the first three points.

**Lesson 2 Part 2:** Open the editor and the open lesson2b.hs. Look at the code and then run the program in the terminal.



Questions:

1. What does the program do?

Draws two lines and their intersection point

1. What does p represent?

The intersection point

1. What does the function line\_line (a,b) (c,d) used for?

Finds where the two lines intersect.

1. How many points could the function line\_line (a,b) (c,d) return?

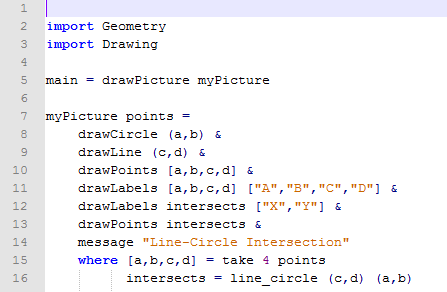
One point of intersection

Zero points of intersection if the lines are parallel.

Line 16 is an incomplete definition. Look at the section named “Understanding runtime errors” in the Coding Overview. If the lines AB and CD are parallel, this program will crash because it (wrongly) expects the two lines to have always one point in common. However, the probability of that case is so low that we do not want to add complexity to the program to handle it.

Nevertheless, it is important to remark the tradeoff we are doing in terms of correctness versus simplicity.

**Lesson 2 Part 3:** Open lesson2c.hs. Look at the code carefully.



Questions:

1. What do you think intersects = line\_circle (c,d) (a,b) will compute?

The intersection point(s) of a line and circle.

Run the program and compare your answers to these questions.

1. What do the letters correspond to in drawCircle (a,b)?

Circle with center at A and B on the circle.

1. What does (c,d) correspond to in line\_circle (c,d) (a,b)?

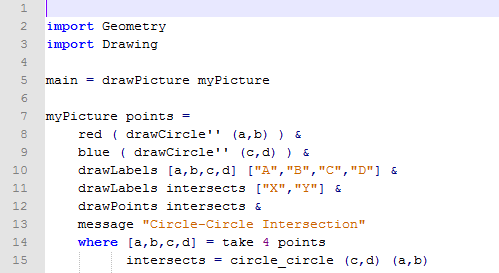
Line CD.

Note to teachers: It may be beneficial to discuss the variable intersects. The reason we must assign a variable intersects in the code is because there are three different possibilities of how this program can run: 1. Two intersection points, 2. One intersection point, 3. No intersection points. We cannot specify that the intersection will only be one point like we did in demo03lineline.hs.

**Lesson 2 Part 4:** Open lesson2d.hs.

***Exercise:*** *Determine what the program will draw.*

*Run the program to check.*



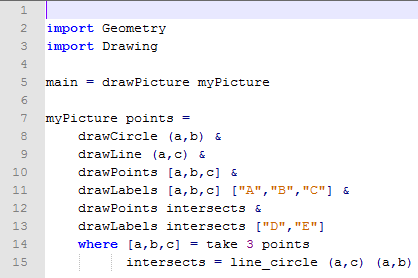
**Lesson 2 Ending Exercise:**

***Exercise:*** *Open lesson2e.hs. Remove the word* undefined *to get started. Create a program to draw the following:*

1. *Circle with center a and point b on the circle.*
2. *Line AC*
3. *Label the intersection points of the circle and line as D and E.*

*Save the program as yourname\_lesson2e.hs*

*Run the program to check.*



***Lesson 2 Further Applications:***

*Teachers: you can easily extend this lesson by removing the randomness from the program. The program lesson2d.hs has three possible outcomes.*

1. *The circles intersect two times.*
2. *The circles intersect one time. (Tangent Circles)*
3. *The circles do not intersect.*

*By removing the randomness such as in lesson2dN.hs, you can demonstrate all three of these situations.*

*You can also have students use this as a tool to discuss whether two circles do not intersect or how many times they intersect. It would be a good tool for checking answers in class or at home.*

*The only factor you need to change is in the* where *statement.*

* *Tangent Circles (one intersection):* [a,b,c,d]=[(-1,0), (-1,-1), (1,0), (1,1)]
* *Two intersection points:* [a,b,c,d]=[(2,0), (2,-1), (1,0), (1,1)]
* *No intersection points:* [a,b,c,d]=[(0,0), (5,5), (0,0), (1,1)]

Example:

