# CPSC 231: Introduction to Computer Science for Computer Science Majors I

## Assignment 1: First Program/Analytical Geometry

### Weight: 6%

### Collaboration

Discussing the assignment requirements with others is a reasonable thing to do, and an excellent way to learn. However, the work you hand-in must ultimately be your work. This is essential for you to benefit from the learning experience, and for the instructors and TAs to grade you fairly. Handing in work that is not your original work, but is represented as such, is plagiarism and academic misconduct. Penalties for academic misconduct are outlined in the university calendar.

Here are some tips to avoid plagiarism in your programming assignments.

1. Cite all sources of code that you hand-in that are not your original work. You can put the citation into comments in your program. For example, if you find and use code found on a web site, include a comment that says, for example:

   ```
   # the following code is from
   https://www.quackit.com/python/tutorial/python_hello_world.cfm.
   ```

   Use the complete URL so that the marker can check the source.

2. Citing sources avoids accusations of plagiarism and penalties for academic misconduct. **However, you may still get a low grade if you submit code that is not primarily developed by yourself. Cited material should never be used to complete core assignment specifications. You can and should verify and code you are concerned with your instructor/TA before submit.**
3. Discuss and share ideas with other programmers as much as you like, but make sure that when you write your code that it is your own. A good rule of thumb is to wait 20 minutes after talking with somebody before writing your code. If you exchange code with another student, write code while discussing it with a fellow student, or copy code from another person's screen, then this code is not yours.
4. **Collaborative coding is strictly prohibited**. **Your assignment submission must be strictly your code**. Discussing anything beyond assignment requirements and ideas is a strictly forbidden form of collaboration. This includes sharing code, discussing code itself, or modelling code after another student's algorithm. **You can not use (even with citation) another student's code.**
5. Making your code available, even passively, for others to copy, or potentially copy, is also plagiarism.
6. We will be looking for plagiarism in all code submissions, possibly using automated software designed for the task. For example, see Measures of Software Similarity (MOSS - https://theory.stanford.edu/~aiken/moss/).
7. Remember, if you are having trouble with an assignment, it is always better to go to your TA and/or instructor to get help than it is to plagiarize. A common penalty is an F on a plagiarized assignment.

### Late Penalty

Late assignments will not be accepted.

## Goal

Writing a first program in Python using I/O, casting, expressions, and conditionals.

## Technology

Python 3

## Submission Instructions

You must submit your assignment electronically. Use the Assignment 1 dropbox in D2L for the electronic submission. You can submit multiple times over the top of a previous submission. Do not wait until the last minute to attempt to submit. You are responsible if you attempt this, and time runs out. Your assignment must be completed in **Python 3** and be executable with Python version 3.6.8+. For graphical drawing you must use the **Python turtle** library.

## Description

### Analytical Geometry

You will create a small graphical **Python 3** program that draws a few shapes using the **turtle** library and information taken from the user. You will also use the Python **math** library to do calculations (no other libraries are allowed for your math calculations). Your TA will introduce you to these libraries and their details during tutorials, so please attend the tutorials.

The assignment requires you to understand constants, magic numbers, variables, expressions, as well as importing and using libraries, getting input from users, casting variables to different types, problem-solving, and drawing in a coordinate system. It is possible to make use of functions for this assignment if you understand them. However, if your program does not function correctly due to a failed attempt, you will lose grades. There are no bonus grades for using functions. There are bonus grades for using loops, which is discussed at the end of this document.

Your program will present an 800x600 pixel window with (0,0) being the bottom left corner point and (800,600) being the top right corner point. Within this coordinate system, your program will draw an x and y axes that identify the centre of the window. Your program will prompt the user for values that define a circle and a line in this coordinate system and draw them. Finally, your program will perform calculations to determine whether the line and circle intersect. At each of the points where the line intersects the circle, you will draw a small circle to indicate the intersection point.

You can think of this assignment as a series of stages to complete. First, you import some libraries. Next, you define constants (all upper case) that replace magic numbers. Some of these are obvious like the window size. You will likely identify more constants as you write the rest of the program. Then setup your window and carry out the rest of the assignment requirements. This code should get you started.

```
turtle.bgcolor(BACKGROUND_COLOR)
turtle.setup(WIDTH, HEIGHT, 0, 0)
screen = turtle.getscreen()
screen.screensize(WIDTH, HEIGHT)
screen.setworldcoordinates(0, 0, WIDTH, HEIGHT)
screen.delay(delay=0)
pointer = turtle
pointer.hideturtle()
pointer.speed(0)
pointer.up()
```

End your program with **screen.exitonclick()** so that the window stays open until it is clicked on.

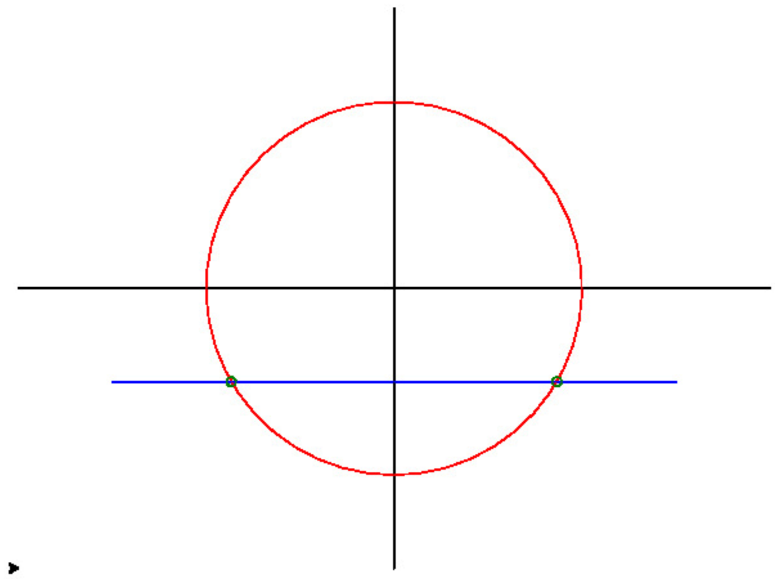
With your window ready, draw your axis. This axis should cross through the middle point of the screen, i.e. (400,300). Remember to use your constants here, and later.

Then prompt the user for input, cast it to the proper type, and store it in variables. Remember to use descriptive variable names. For this assignment, it is sufficient to adopt the same variable names used in the mathematical equations below.

Draw your circle and your line.

Calculate the number of intersection points between the circle and line. Then, produce a conditional statement that lets you display a message indicating the number of intersection points to the user.

Calculate the location of each interaction point (if any) and draw a circle around it.



## Drawing in Turtle:

You can imagine the **turtle** drawing system as a turtle carrying a pen. The turtle starts in some default location within the window and has its pen pressed down and ready to draw. To draw a line from the turtle's location to a new point, tell the turtle to move to the new location. It will drag the pen along as it moves to the new location effectively drawing a line. If you tell the turtle to lift up the pen, then tell it to move to a new point, it will go to this new point without drawing the line.

For this assignment, you must know how to position the turtle in a specific location (a point in the coordinate system, also known as pixel location), how to start and stop drawing, and how to change drawing colours. **Draw the axis in *black*, the circle in *red*, the line in *blue*, and the intersect/text in *green*.**

You can use the **circle** command to draw the circle. The **circle** command, by default, draws circles from the middle of the bottom edge. Therefore, to draw a circle around a centre point, you must determine where the proper start location is (hint: adjust from the centre using the radius). (If you turn your pointer [turtle], then because the circle starting point is to the left of the turtle this will change how your circle is drawn https://docs.python.org/3.3/library/turtle.html?highlight=turtle ).

## Getting Input:

Read the input from the user using the techniques that we have discussed in class. Display a prompt in the terminal window with a print statement, or by providing a parameter to the input function. Note that the user will enter their input in the terminal window, not in the graphics window.

You will need to prompt for 7, and only 7, inputs (unless you are doing the bonus). You will need an **integer** $x_c$ and **integer** $y_c$ for the centre $(x_c, y_c)$ of the circle and a **float** $r$ for the radius. You will need an **integer** $x_1$ and **integer** $y_1$ for the start $(x_1, y_1)$ of the line and an **integer** $x_2$ and **integer** $y_2$ for the end $(x_2, y_2)$ of the line. These are pixel locations in the window. A circle with middle (400,300) and radius 300 would be drawn in the middle of the window touching the top and bottom of the (800,600) axis in the window [i.e., 300-300==0 and 300+300==600].

## Analytical Geometry for Intersection:

We have 7 values that you should have stored as input: $x_c, y_c, r, x_1, y_1, x_2, y_2$. We will use these in our calculations.

We will make use of 3 intermediate calculations to determine intersections.

$$a = (x_2 - x_1)^2 + (y_2 - y_1)^2$$

$$b = 2[(x_1 - x_c)(x_2 - x_1) + (y_1 - y_c)(y_2 - y_1)]$$

$$c = (x_1 - x_c)^2 + (y_1 - y_c)^2 - r^2$$

These fulfill the parameters of the quadratic

$$a\,\alpha^2 + b\,\alpha + c = 0$$

We can solve this quadratic to determine intersections alpha (α). However, we will not do this directly, but instead using the quadratic formula.

$$\alpha = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Before we jump directly to using this to calculate alpha (α) we can look at the value under the root ($b^2 - 4ac$ ) to determine the number of intersections.

If $b^2 - 4ac < 0$ then there are no intersections. We cannot use the imaginary number from square rooting a negative number in this situation.

If $b^2 - 4ac = 0$ then there is 1 intersection. The square root value is the root of 0. The result of adding and subtracting 0 is the same number. So there is only one result to the alpha.

If $b^2 - 4ac > 0$ then there are 2 intersections. Since the value is positive when add or subtract the result of rooting it, we will produce two different alphas; one for each intersection.

After making a decision based on this value, we need to draw the result of the intersections. If there are no intersections, you should draw the words "No Intersect!" in the centre of the window. If there is one, or two intersections, then you should then calculate the alpha of the quadratic formula for each.

With each alpha, we can calculate the coordinates of the intersection point using the following two formulas

$$x = (1 - \alpha)\ x_1 + \alpha\, x_2$$

$$y = (1 - \alpha)\ y_1 + \alpha\, y_2$$

With this coordinate, you should draw a circle around this location. Using a radius of 5 is a good size. In effect the alpha is a ratio spot along the line between $(x_1, y_1)$ and $(x_2, y_2)$ . An alpha of 0 is the point $(x_1, y_1)$ and an alpha of one is the point $(x_2, y_2)$. **Note that an alpha value greater than 1 or less than 0, indicates that the intersection occurs outside the end points of the line segment. These intersections should not be drawn.**

A line where $(x_1, y_1)$ == $(x_2, y_2)$ is a unique case which is not a line, but a single point. This point is either on the circle, or not. An intersection exists if the distance between the centre of the circle and the point is the same as the radius of the circle. (We will check this up to an accuracy epsilon of $\varepsilon = 0.75$). If the distance and the radius are within 0.75 of difference, then we will consider there to be a single intersection.

One final and very important point: although we may calculate an alpha that indicates that there is an intersection, **this intersection may not be between the two points of the line**. The calculation we have used so far also gives us intersections if line continues into infinity off either end. To ensure we only plot intersections between the two points we need to make sure that the alpha greater than or equal to zero and less than or equal to 1.

## Additional Specifications:

Ensure that your program meets all the following requirements:

- You should have the class, your name, your tutorial, your student id, the date, and a description at the top of your code file in comments. Marks are given for these.
- The code file should be **CPSC231F21A1-<Name>.py**
    - (ex. Mine would be CPSC231F21A1-Hudson.py)
- Import the necessary libraries
- Use constants appropriately. Your TA may overlook one or two magic numbers, but more will result in loss of marks.
- Use descriptive naming of variables. The use of variables that follow the math descriptions given are fine for those purposes.
- Draw your axis black, your circle red, your line blue, and your intersection circles/text green.

- Use descriptive prompts for input and cast input to the indicated types.
- Use in-line comments to indicate blocks of code and describe decisions or complex expressions.
- No penalty for use of functions; but avoid adding unrequested functionality to the assignment as this will make grading for TAs harder, and may even result in a loss of grades if mistakes are made that break regular requirements.
- You do not have to worry about making sure input is of the correct type. Your program should crash if the values given cannot be cast from strings into the request types.
- The only thing your program should print otherwise is the input request messages, unless you you have done the bonus.

## Bonus:

Add a loop to your program that will prompt for additional pairs of coordinates that indicate the start and end of more line segments to be drawn over the same circle. The program should continue prompting for these additional lines until the user enters a blank input (presses Enter without typing) for one of the coordinates. Plot each line and circle the intersection points. Do not refresh the circle drawing each time; just draw each line over top of the existing drawings. You do not have to worry about old “No Intersect!” messages on the screen; just let them stay drawn. Each time a new line is drawn, print to output the total sum of how many intersections have been found so far.

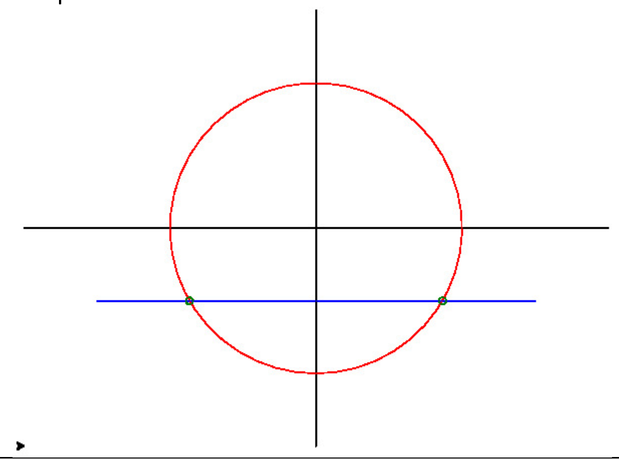
## Examples:

On the course website are 4 sample files for testing your program. Even if they all look correct, this does not guarantee your program is fully correct or that your program will get a perfect grade. TAs will have alternative tests and other penalties they can apply. Each of these files contains a sequence of 7 inputs : $x_c, y_c, r, x_1, y_1, x_2, y_2$ on a separate line of the file. You can type in these entries to get the picture that corresponds below.

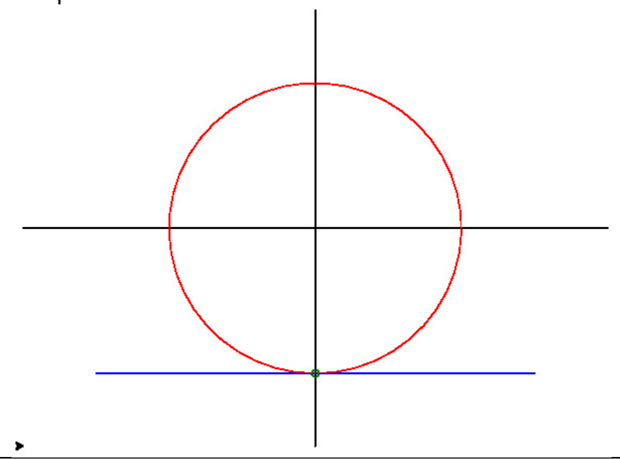
In Linux, Windows cmd shell (not power shell), and MacOS we can push these into the Python program as if we were typing input with the following

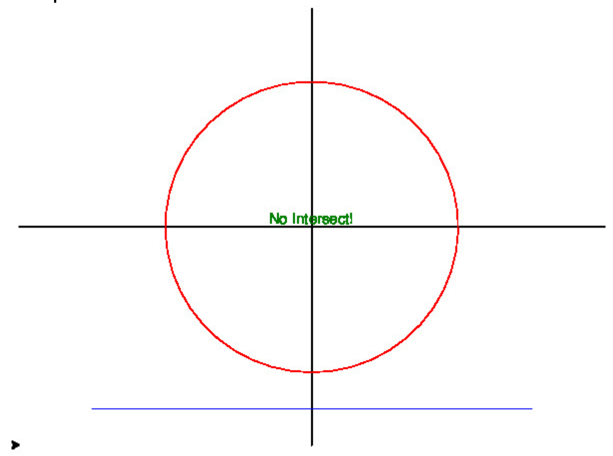
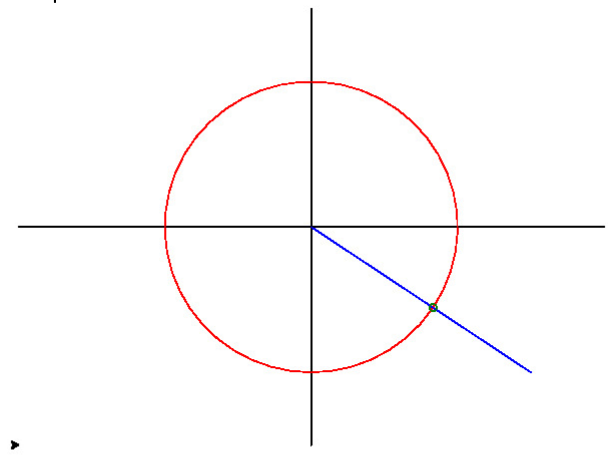
**python CPSC231F21A1-Hudson.py < sample1.dat**

sample1.dat



sample2.dat



| | |
|---|---|
| sample3.dat  | sample4.dat  |

**Example input in the terminal window should look as follow:**

```
Enter circle x coordinate:400
Enter circle y coordinate:300
Enter radius of circle:200
Enter line start x coordinate:100
Enter line start y coordinate:200
Enter line end x coordinate:700
Enter line end y coordinate:200
```

## Grading:

Your TA will begin by grading your code by determining the general functionality of the code and assigning a representative grade as a starting point. The TA will then subtract marks for mistakes and incomplete requirements, such as not commenting your code adequately or at all!

The total mark achieved for the assignment will be translated to a letter grade using the following table:

## Submit the following using the Assignment 1 Dropbox in D2L:

1. CPSC231F21A1-Name.py

## Grading:

| Mark | Letter Grade | Starting Point Guidelines (not a final grading scheme!) |
|---|---|---|
| 13 | A+ | Appears to fulfill assignment and bonus spec |
| 12 | A | Appears to fulfill assignment spec |
| 11 | A- | |

| | | |
|---|---|---|
| 10 | B+ | |
| 9 | B | Code has part of intersection work done, but obvious errors/incomplete |
| 8 | B- | |
| 7 | C+ | |
| 6 | C | Code draws axis, gets input, draws circle/line |
| 5 | C- | |
| 4 | D+ | |
| 3 | D | Code draws axis and gets input |
| 0-2 | F | Syntax errors or barely started code |

*As a reminder, the University of Calgary assigns the following meaning to letter grades:*

*A: Excellent – Superior performance showing a comprehensive understanding of the subject matter*

*B: Good – Clearly above average performance with generally complete knowledge of the subject matter*

*C: Satisfactory – Basic understanding of the subject matter*

*D: Minimal Pass – Marginal performance; Generally insufficient preparation for subsequent courses in the same subject*

*F: Fail – Unsatisfactory performance*