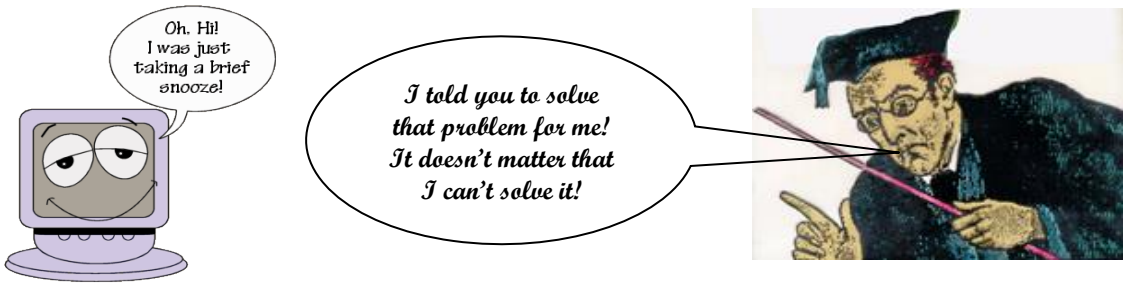
Unit 1 – Introduction to Programming through App Inventor

The most Important Lesson of the Entire Course

The Most Important Lesson of the Entire Course

The process of writing a program can be viewed as a form of "teaching." Whenever you write any computer program, you are, in a sense, "teaching" a computer how to solve a particular problem. KEEP IN MIND THAT YOU ***CANNOT*** "TEACH" A COMPUTER TO SOLVE A PROBLEM THAT YOU DO NOT KNOW HOW TO SOLVE!

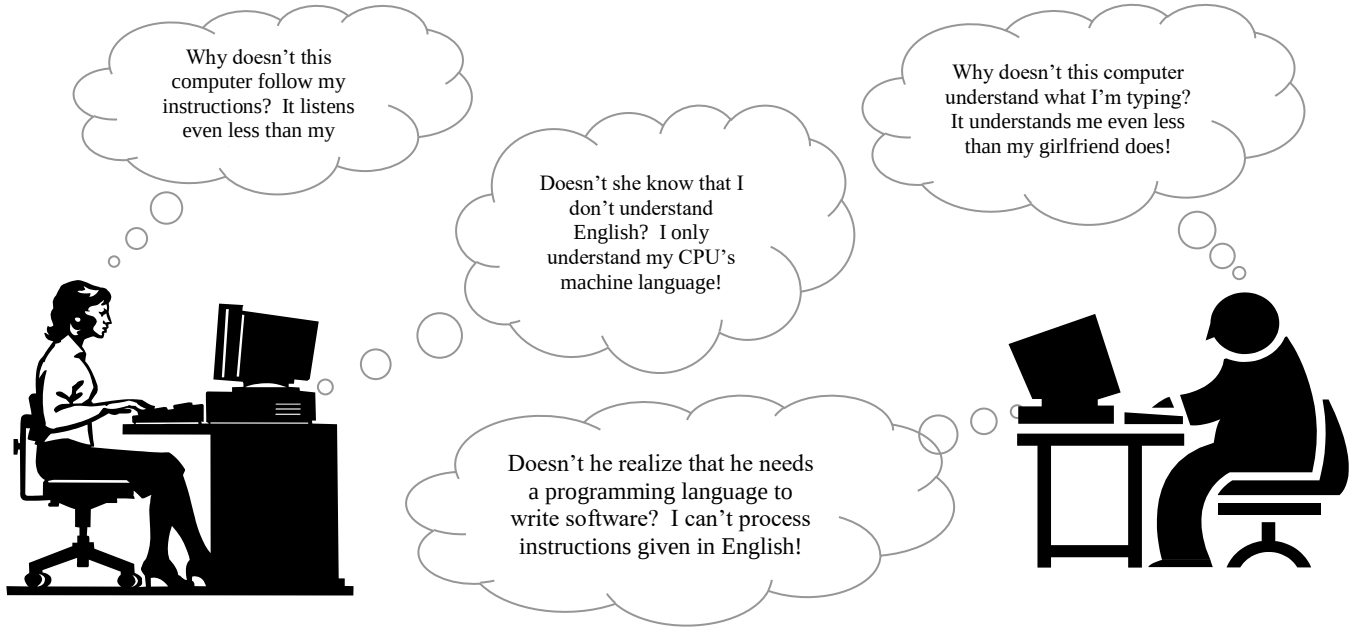


BEFORE YOU EVEN ATTEMPT TO WRITE CODE (PROGRAMMING INSTRUCTIONS), FIRST YOU MUST DEVISE A STRATEGY! BEFORE YOU CAN DEVISE A STRATEGY, YOU MUST ENSURE THAT YOU UNDERSTAND THE PROBLEM! THE FOLLOWING TABLE DESCRIBES A SOUND APPROACH TO SOFTWARE DEVELOPMENT. IF YOU HOPE TO BE SUCCESSFUL, FOLLOW THE GUIDELINES IN THE SECOND AND THIRD COLUMNS. **DO NOT FOLLOW THE STEPS IN THE FIRST COLUMN!**

Planning and Developing Solutions to Software Development Problems

Wrong!!!!!	***Right!!!!!! (George Polya)***	***How these Steps Apply to Software Development***
	1. Understand the problem ***(Analysis)***	**1.** Before you begin constructing a solution to a problem, you must know exactly what is required. Otherwise, you run the risk of solving the wrong problem or providing an incomplete solution to a given problem. In particular, you need to know what the output of the program should be given every possible input. ✓ **Input** → **Output** **?**
~~**1.** Read problem~~	**2.** Choose a strategy ***(Design)***	**2.** You may design an interface for your program at this stage but you MUST NOT write any code yet! In addition, it is important to consider a variety of algorithms. Break up the large problem into ***several smaller sub-problems***. Choose algorithms that best balance user ease, execution speed, programming complexity (ease of implementation) and storage requirements.
~~**2.** Type code~~	**3.** Carry out the strategy ***(Implementation)***	**3.** At this stage, you write code but NOT all in one fell swoop. Write code for one sub-problem at a time. Do not integrate a solution to a sub-problem into the larger solution until you are confident that it is correct. It is also wise to save each version of your program. In case of a catastrophe, you can always go back to an earlier version.
~~**3.** Run the program~~	**4.** Check the solution ***(Validation)***	**4.** Extensive testing should take place to find bugs that were not noticed in the implementation phase. It is best to allow the testing to be done by average computer users who are not programmers. Because of their computer expertise, programmers unconsciously tend to avoid actions that cause computer programs to fail. Once the software is released, additional bug fixes will usually be necessary as users report previously undiscovered bugs. This is known as the ***maintenance phase***.

What is a Program? What is a Programming Language?



Important Terminology

Program

- A ***program*** is a sequence of ***instructions*** that a computer can ***interpret*** and ***execute***. ("Execute" means "carry out" in this context.)

Programming Language

- A ***programming language*** is a very ***precise*** and ***unambiguous*** language that is designed to allow ***instructions*** to be given to a computer.

Code

- Programming instructions are often called "code." Programmers say that they are "writing code" when they write programs.

Bug

- A ***fault*** or ***defect*** in a computer program, system, or machine. Bugs are usually due to design flaws.

Algorithm

- An ***algorithm*** is a systematic procedure (finite series of steps) by which a problem is solved. Long division is an example.
- The steps of a particular algorithm remain the same whether you solve a problem by hand or by computer.
- In cooking, algorithms are called ***recipes***.
- Algorithms have been worked out for a wide range of problems.
- For many problems, there exist many different algorithms.
- For some problems, there are no known efficient algorithms (too slow and/or require too much memory). **e.g.** What are the prime factors of a number?
- Some problems cannot be solved by a computer (**i.e.** no algorithm exists that can be implemented on a computer).

Learning the Essential Features of App Inventor

Brief Description and History of App Inventor for Android

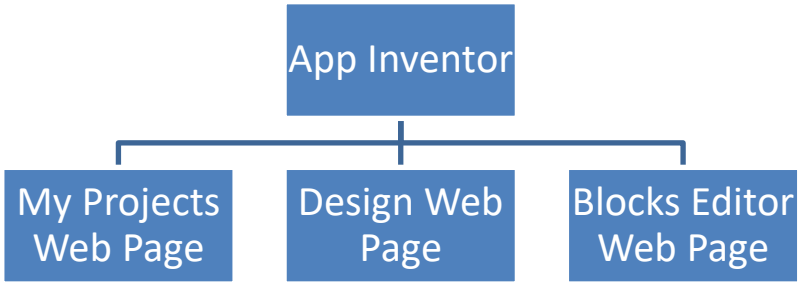
- Allows anyone to create software applications ("apps") for the Android Operating System
- The Android Operating System is used on several different mobile devices including models made by Samsung, HTC, LG, Motorola, Sony, Alcatel, Archos, Kyocera, Dell, Xperia, Excite, Asus, Sanyo, Acer, etc
- Originally provided by Google and called "Google App Inventor"
- Google terminated support for App Inventor on December 31, 2011 but donated the project to MIT
- Since then, the application has been maintained by MIT (Massachusetts Institute of Technology)
- Now called "MIT App Inventor 2" (original version now called "MIT App Inventor Classic")

Overview of App Inventor 2

How to Access App Inventor

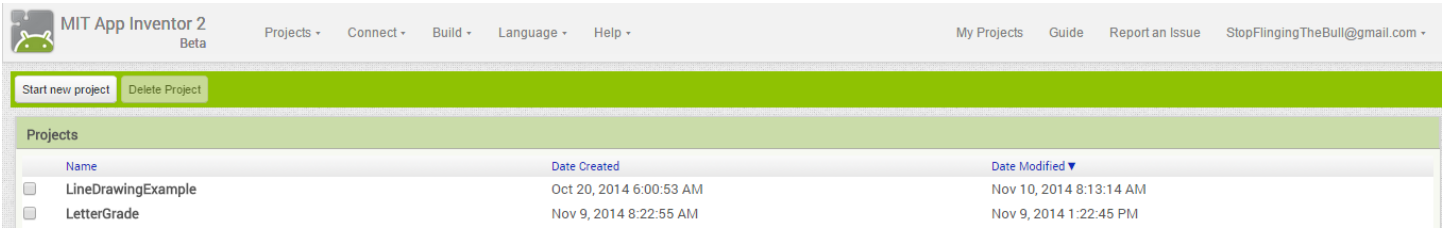
- Requires a Google account. Visit https://accounts.google.com/NewAccount if you don't have one.
- Once you have a Google account, log on to App Inventor at http://ai2.appinventor.mit.edu/
- To test your app while you are developing it, you may choose one of the following options:
  - **(a)** **Preferred Method:** Use any supported mobile device running the Android operating system. Such a device can be connected to App Inventor by Wi-Fi or with a USB cable. In either case, the "MIT AI2 Companion" app must be installed on your mobile device.
  - **(b)** Use the **App Inventor Android Emulator**. This option is provided to allow app development even when an Android device is not available. (Unfortunately, the emulator tends to be slow and crash prone.) To use this option, the "App Inventor 2 Setup Package" must be installed on your computer. Details can be found at http://appinventor.mit.edu/explore/ai2/setup.html.

How to Work with App Inventor

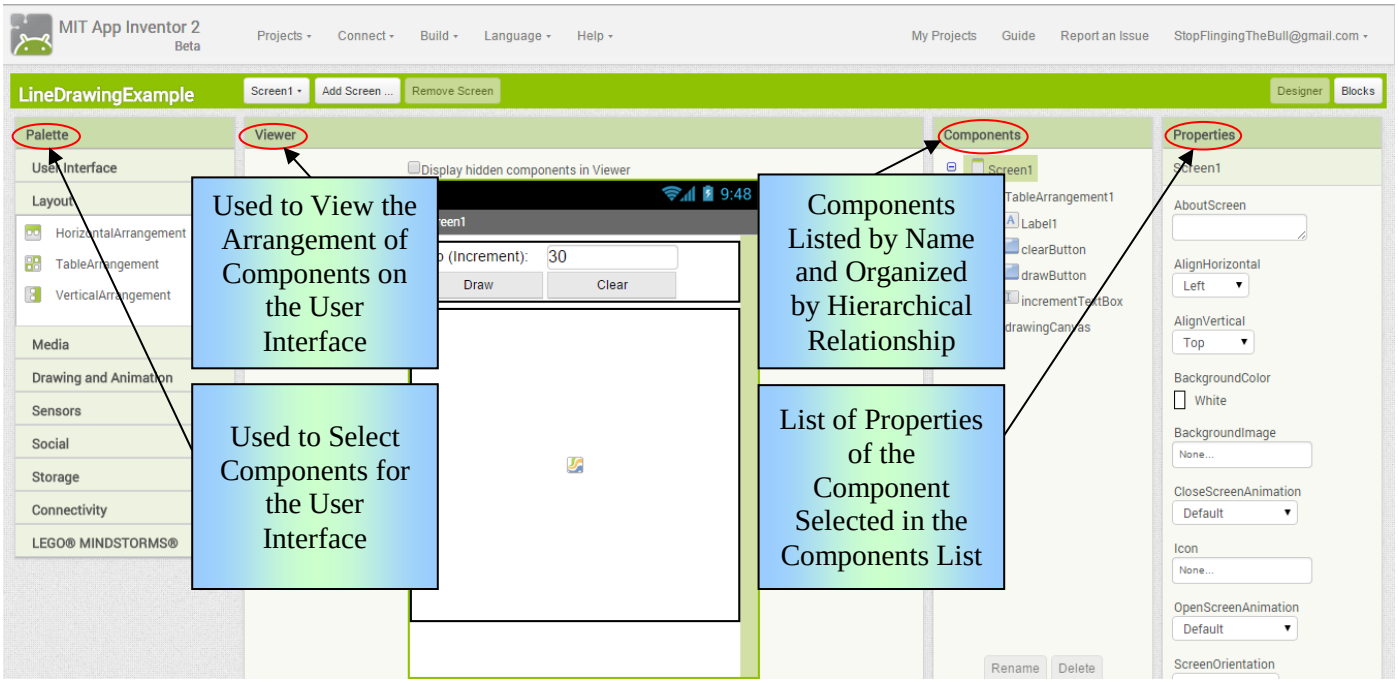


- **My Projects Web Page**
  What you usually see when you first log on
  Create New Project, Open Existing Project, Delete Projects, etc
- **Design Web Page**
  Tools for Designing the User Interface
  Palette, Viewer, Components List, Properties List
- **Blocks Editor Web Page**
  Tools for Specifying the *Logic* (i.e. Behaviour) of the App
  In other words, the blocks editor allows the programmer to specify *instructions* for the app

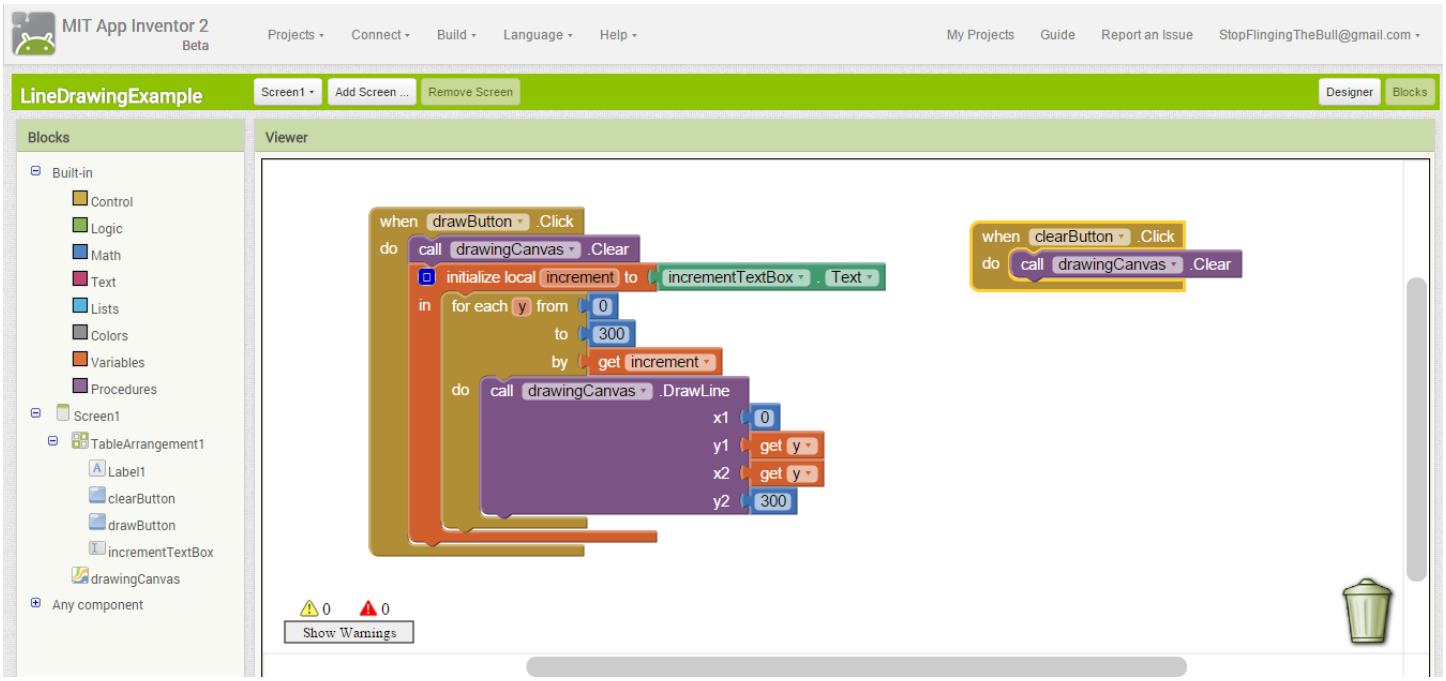
The MyProjects Page



The Design Page



The Blocks Editor



PaintPot: Creating your First App

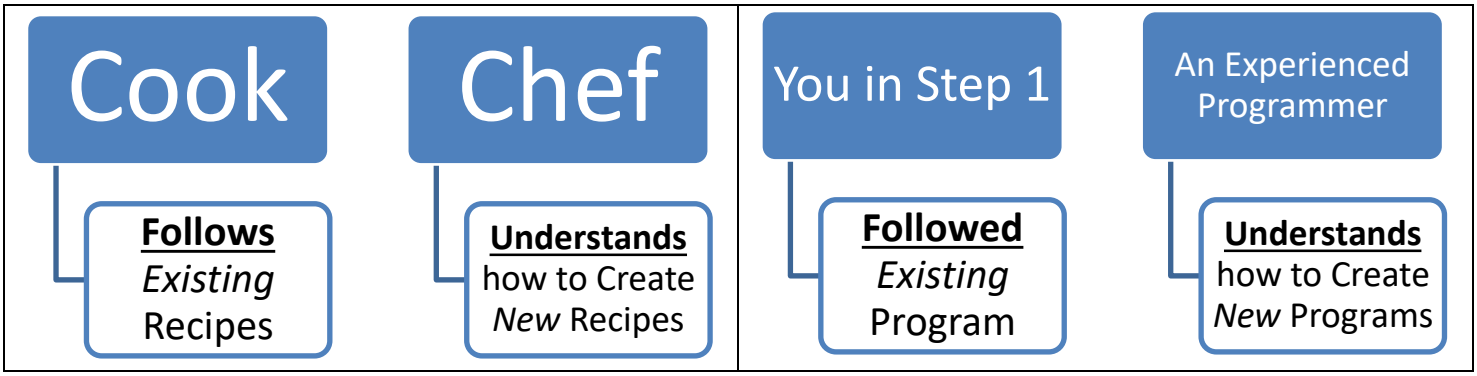
Step 1: Creating the PaintPot App

This part is easy! All you need to do is follow the instructions on the following Web page:

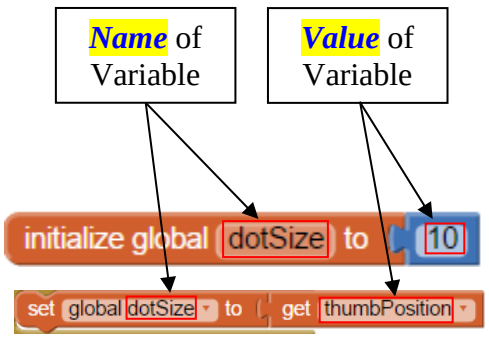
http://www.appinventor.org/paintpot2-steps

If you follow the instructions very carefully, the app should function correctly. In the event that it does not work as expected, check your blocks carefully to ensure that they are exactly as shown in the above document.

Step 2: Using the Cook/Chef Analogy to Understand the Logic of the PaintPot App



As can easily be appreciated from the above analogy, it is not enough merely to *follow* existing programs. All programmers must also be able to develop new software from scratch. To accomplish this, it is obviously very important to ***understand*** programming concepts. A detailed description of the programming concepts used in PaintPot is given below.

Picture	***Programming Concepts***
 **(a)** In the first example, the variable "dotSize" is *created* and given an *initial* (starting) value of 10. **(b)** In the second example, the value of "dotSize" is *changed*. Its value is ***set*** to the same value as that of the variable "thumbPosition." **(c)** The modifier "global" means that the variable is can be accessed and changed by *any* procedure. In addition, the value of the variable will remain stored in memory for as long as the app is running.	***Variable*** • A ***variable*** is a name that is used to represent a value that is stored in a computer's main memory (i.e. in the RAM). • Variables are used whenever information needs to be "**remembered**" (i.e. "memorized") for later use. • The concept of variable in computer science is similar but not identical to the concept of variable in mathematics. • One key difference is that in mathematics, variable names must have a length of exactly one character. For example, the variable name "*x*" is allowed but the variable name "*xavier*" is not allowed because it would be interpreted as "*x* times *a* times *v* times *i* times *e* times *r*." • In computer science, variable names ***can*** contain more than one character because the multiplication operator (*) cannot be omitted. Thus, the name "*xavier*" would be seen as a single entity and not a series of multiplications. • In most cases, variable names in programming should contain more than one character because ***descriptive names*** make programs far easier to understand. For example, "dotSize" is far more meaningful than "x," "y" or "d."

Picture

Name of ***Component*** (aka Object)

Name of the ***Event*** that causes execution of procedure block

Name of a ***Property*** of a Component

Procedure. The instructions within the block are executed when the "Click" event occurs on "RedButton."

Value of the "PaintColor" Property

Programming Concepts

Procedure *(aka Subprogram, Subroutine)*

- In App Inventor, a ***procedure*** is used to group together one or more instructions.
- Each procedure has a unique ***name***.
- Some procedures are executed automatically when a specific ***event*** occurs. These are called ***event-handling procedures*** or just ***event handlers***.
- Other procedures are executed in response to a specific ***instruction*** called a "call" of the procedure.

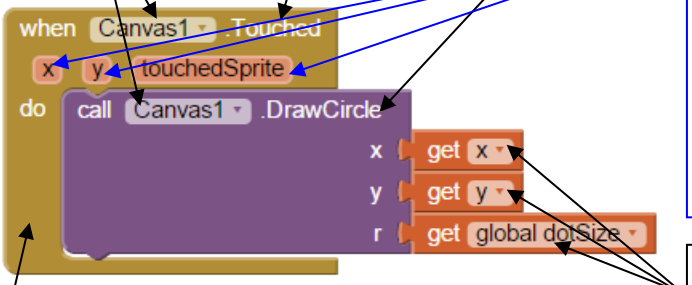
Event

- An ***event*** is an occurrence that takes place while a program is running. Events are used to trigger the execution of specific instructions.
- Examples of events include "Click," "LongClick," "GotFocus," "LostFocus," "Dragged" and "Touched."

Name of ***Component*** (aka Object)

Name of the ***Event*** that causes execution of the procedure

Name of a ***Method***



The ***Parameters*** of the "DrawingCanvas.Touched" procedure block. These are special variables that are used to pass information to the procedure block. In this example, the parameters *x* and *y* store the co-ordinates of the point that is touched on "DrawingCanvas." The parameter "touchedSprite" is used for animations.

The ***Arguments*** passed to the "DrawCircle" method. The values of *x* and *y* come from the parameters *x* and *y* of the procedure block "DrawingCanvas.Touched." The radius of the circle comes from the value of "dotSize."

Procedure Block with Parameters
The instructions within the block are executed when the "Touched" event occurs on "DrawingCanvas." The parameters of this procedure block are the variables *x*, *y* and *touchedSprite*.

Property

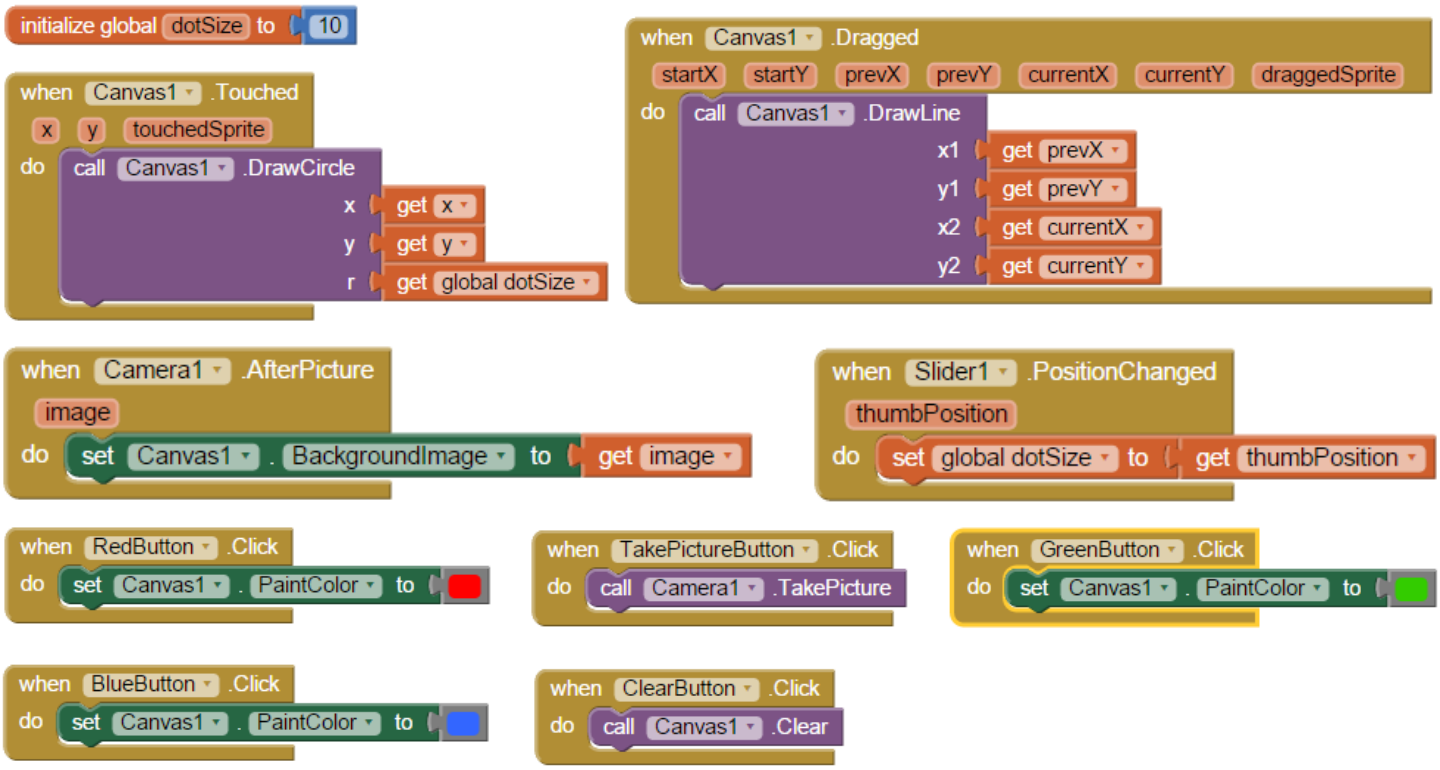
- Every component has ***Properties***, which store information on characteristics of the component.
- A ***property*** can be considered a ***variable*** that belongs to a component.
- Examples of properties include "Height," "Text" and "Width."

Method

- Every component has ***Methods***, which are actions that are associated with the component.
- A ***method*** can be considered a ***procedure*** that belongs to a component.
- Examples of methods include "DrawCircle," and "DrawLine."

Exercises

Study the following diagram. Then answer the questions found below the diagram.



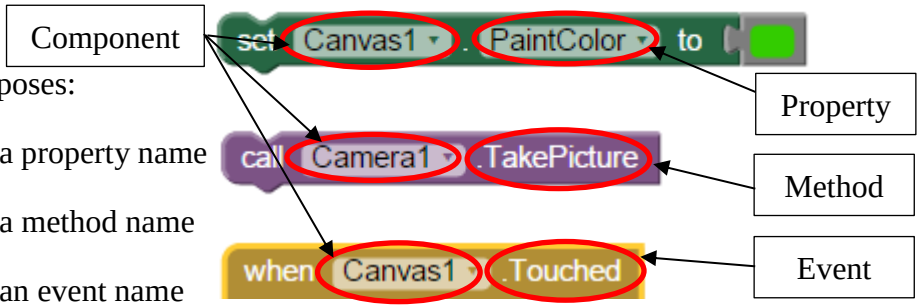
1. "dotSize" is the name of a ________. These are used to _____ _____.
2. "Touched," "Dragged," "AfterPicture," "PositionChanged" and "Click" are names of _____.
3. "Canvas1," "Camera1," "Slider1," "RedButton," "BlueButton," "GreenButton," "ClearButton" and "Canvas1TakePictureButton" are names of _____.
4. "Canvas1.Touched," "Canvas1.Dragged," "Camera1.AfterPicture," "Slider1.PositionChanged" and "RedButton.Click" are names of _____ , which are procedures that are executed automatically in response to _____.
5. "PaintColor" and "BackgroundImage" are _____ , which are _____ or _____ of components.
6. "DrawCircle," "DrawLine," "Clear,"and "TakePicture" are _____ , which are _____ that belong to _____ components.
7. The names "x," "y," "touchedSprite," "startX" and "startY" are all examples of _____. These are used to _____ .

Some Important Notes on MEANING!

The Meaning of the "Dot"

In App Inventor, the dot serves three purposes:

1. Separates the component name from a property name
2. Separates the component name from a method name
3. Separates the component name from an event name



The Meaning of "Set" and "Get"

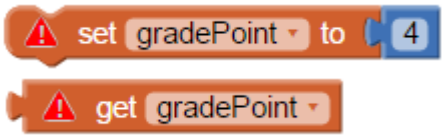
According to the Oxford English Dictionary (OED) "Dictionary Facts" Web page (http://public.oed.com/history-of-the-oed/dictionary-facts/), the verb "set" can be used in *430 different senses*. In fact, as shown in the following passage copied directly from the Web page mentioned above, the verb "set" is the longest entry in the entire OED!

Longest entry in Dictionary: the verb 'set' with over **430 senses** consisting of approximately 60,000 words or 326,000 characters

The word "get" also can also be used in numerous senses.

For our purposes, it is very important that we understand the senses in which "set" and "get" are used in App Inventor. This is explained below.

Verb: set	**Verb: get**
Decide upon definitely; ***give a value***	Go or come after and ***bring or take back***
"Set the thermostat to 20 degrees Celsius, please."	"Get me those books over there, please."



Assign (give) the variable "gradePoint" a value of 4.

Retrieve (bring back) the value of "gradePoint" from memory.

The Meaning of "Initialize"

Verb: initialize

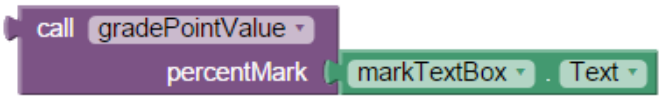
Assign (give) an initial (starting) value to

This creates the ***global variable*** "x" and assigns it an initial value of 3. The value of a global variable can be accessed and changed by any procedure. A global variable should be used whenever two or more procedures need to access or change its value.

The Meaning of "Call"

Verb: call

Execute a procedure (a *named* subprogram)



This ***calls*** (executes) the procedure (subprogram) called "gradePointValue." The variable "percentMark" can be thought of as an "input" of the procedure.

Creating More Apps – More Examples

Resources

1. App Inventor 2 Course-In-A-Box: http://www.appinventor.org/content/CourseInABox/Intro/courseinabox
2. App Inventor 2 Book: http://www.appinventor.org/book2
3. S:\OUT\Nolfi\ICS3U0\00-AppInventor\0. App Inventor 2 Files

Warning!

By following the instructions in the resources listed above, you will be able to create many impressive and interesting apps. However, you must always keep in mind that the ultimate objective is to ***UNDERSTAND PROGRAMMING CONCEPTS.*** This means that you must ***THINK CRITICALLY AS YOU WORK***. Once you develop a sufficient understanding of the concepts, you will be well on your way to developing your own apps and more importantly, you will be well on your way to being able to ***THINK FOR YOURSELF!***

Creating More Apps – Making your OWN!

Introduction

Now that you have gained experience creating apps by following detailed instructions, it's time to "cut the umbilical cord." It should be obvious to you that to be a genuine software developer, you should be able to create apps without following detailed instructions. If this seems difficult at first, don't despair! Just keep the following simple equation in mind and eventually you'll develop the instincts that will allow you to create software at will.

Problem Solving Skills	+	Creativity	+	Logic	+	Understanding of Concepts	+	Discipline and Perseverance	=	Great Apps!

Method 1 – Improve an Existing App: MoleMash Extreme Version

A. Create the "MoleMash" app. (http://www.appinventor.org/bookChapters/chapter3.pdf). Before proceeding to the next step, please ensure that you understand the "MoleMash" app fully!

B. Add the following features to the MoleMash app:

1. **Levels of Difficulty:** "Easy," "Medium," "Difficult" (e.g. the game can be made more challenging by increasing mole speed, decreasing size of the mole picture, etc)
2. A pleasant sound is played when the mole is hit.
3. The mole picture changes briefly when the mole is hit.
4. A rude sound is played when the mole is missed.
5. The game ends after a certain number of hits and misses, after which the player is declared either a winner or a loser.
6. To begin the game, the player enters his/her name.

7. A “bonus image” is occasionally displayed for a brief time. Bonus points are awarded for tapping the bonus image.

8. A “penalty image” moves about the canvas in proximity to the mole image. If the player taps the penalty image instead of the mole, the player loses points.

9. List any other improvements you can think of in the space provided below:

Method 2 – Create your own Apps!

There is no better way to learn about programming than to create your own apps! You are strongly encouraged to unleash your imagination and explore whatever ideas come to mind!

Upcoming Assignment

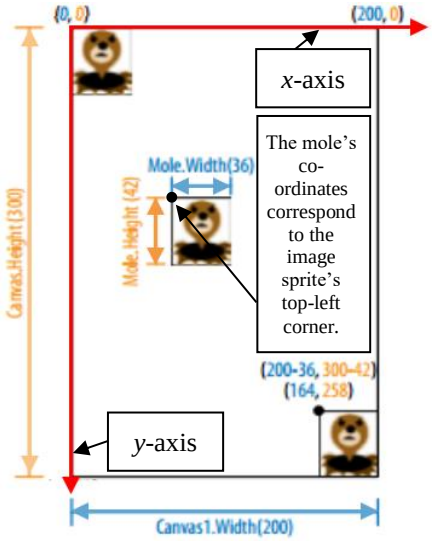
Very soon, you will create your first app that is to be submitted to the teacher for evaluation.

At this stage, you should begin collecting ideas. Keep a record of ALL your ideas, regardless of how crazy you might think they are. When the time comes to develop your app, you can select your most realistic ideas and save the others for future projects.

Understanding MoleMash

The Co-Ordinate System of a Canvas

The diagram at the right shows the co-ordinate system used in the MoleMash game. Notice the following features of the co-ordinate system:



A. The origin is at the top-left corner of the canvas.

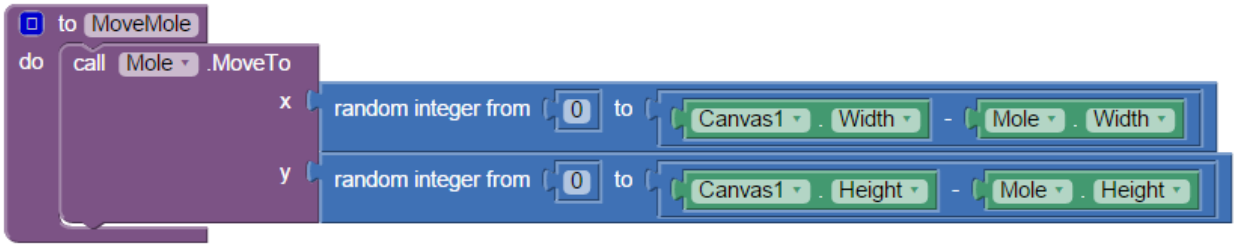
B. The *x*-axis is horizontal.

C. The *y*-axis is vertical but upside-down.

Questions

Refer to the diagram at the right whenever necessary.

1. Fill in the blanks. Your goal is to understand the ***logic*** and the ***meaning*** of the given blocks.



(a) "MoveMole" is a _____ , which is a _____

_____ .

(b) "MoveTo" is a _____ , which is a _____

_____ .

(c) The special variables "x" and "y" are called _____ . They represent the ___ and ___ co-ordinates of the top-left corner of the _____ image sprite. Both of these variables are given random integer values. Explain the following:

Why is "x" given a random integer value ranging from 0 to Canvas1.Width – Mole.Width?

_____

_____

Why is "y" given a random integer value ranging from 0 to Canvas1.Height – Mole.Height?

_____

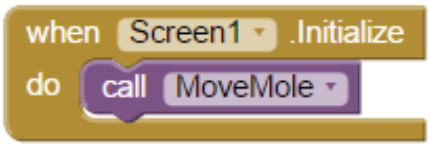
_____

2. The "MoleMash" app uses a "clock" component. Explain in detail how this component works and its specific purpose in the "MoleMash" app.

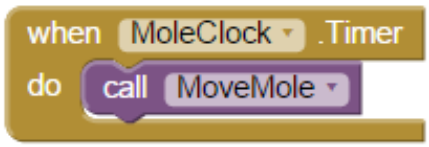
Using "MoleMashImproved" to Review App Inventor Main Ideas

- Upload "MoleMashImproved" from "**S:\OUT\Nolfi\ICS3U0\00-AppInventor\0. App Inventor 2 Files**" to your "My Projects" page.
- After testing "MoleMashImproved" and studying its blocks carefully, answer the following questions:

1. *Identify and Explain Purpose*

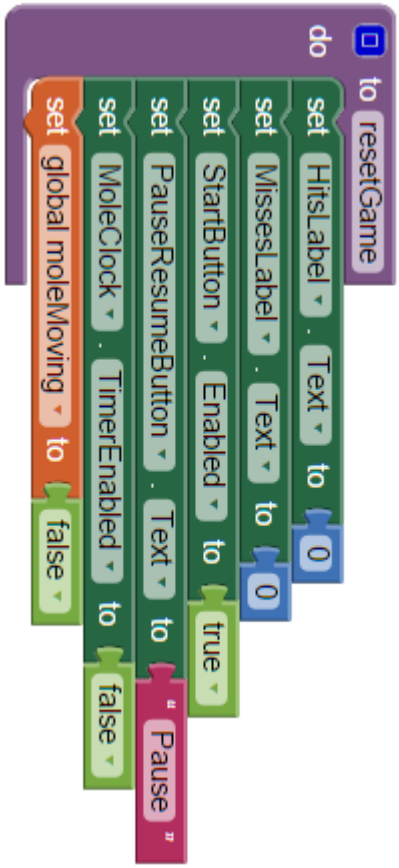


(a) List all the ***procedure names*** in the blocks shown at the left.



(b) List all the ***component names*** in the blocks shown at the left.

(c) List all the ***event names*** in the blocks shown at the left.



(d) List all the ***property names*** in the blocks shown at the left.

(e) Explain the purpose of "Screen1.Initialize."

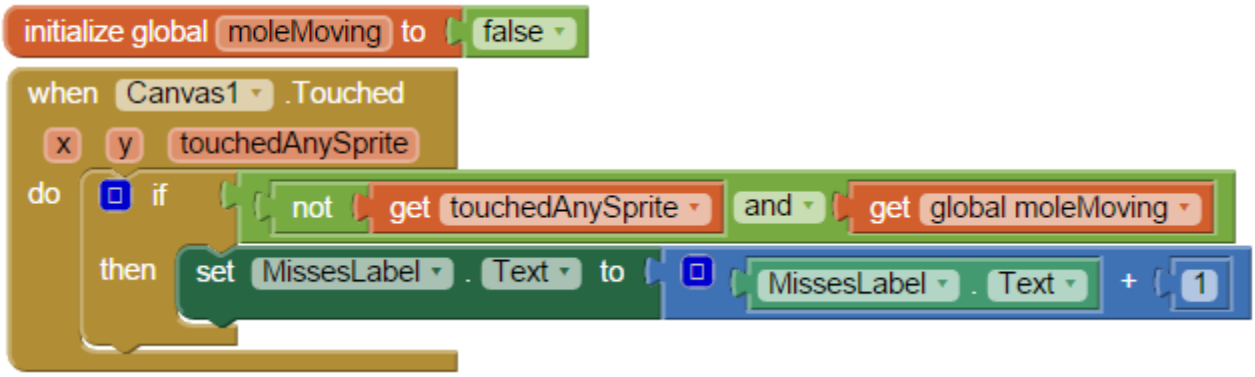
(f) Explain the purpose of "MoleClock.Timer."

(g) Explain the purpose of "call MoveMole."

(h) Explain the purpose of "call resetGame."

(i) Explain the purpose of the "resetGame" procedure. Is "resetGame" an event handler?

2. *Explain Purpose*



(a) What are "x," "y" and "touchedAnySprite?" What is their purpose? Be specific!

(b) Explain the purpose of the "if" block as well as the "not" and "and" blocks. Again, be specific!

(c) What is the purpose of "set MissesLabel.Text to MissesLabel.Text +1?"

3. *Explain Concepts*

(a) Explain why it makes sense to create a procedure like "MoveMole" rather than copying and pasting instructions.

(b) How many times is "MoveMole" called in "MoleMashImproved?" Explain the purpose of these calls.

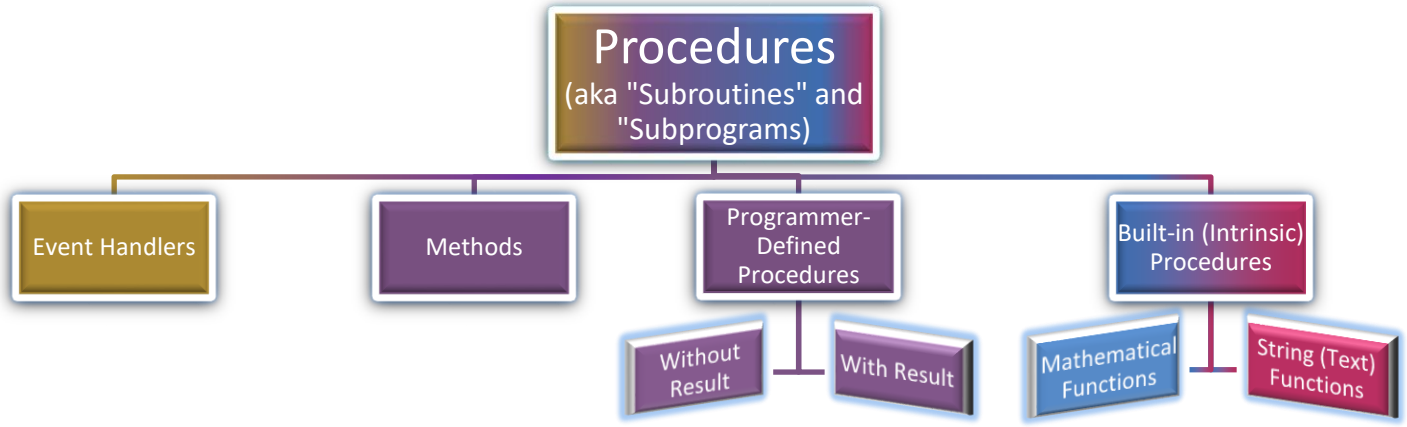
Review: Explain App Inventor Main Ideas

Explain each of the following:

1. Component	
2. Property	
3. Method	
4. Event	
5. Procedure	
6. Event Handler	
7. Click Event	
8. Initialize Event	
9. Timer Event	
10. Text Property	
11. Variable	
12. Call	
13. Parameter/Argument	
14. if block	
15. Image	
16. Sprite	
17. random integer	
18. Canvas	
19. Width Property	
20. Height Property	
21. Co-ordinate System	

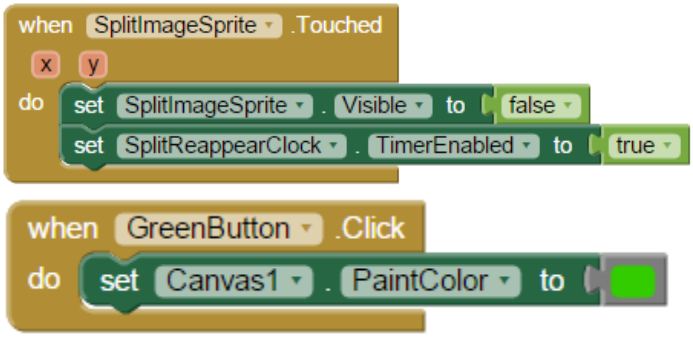
Appreciating the Power of Programmer-Defined Procedures

Overview of Procedures

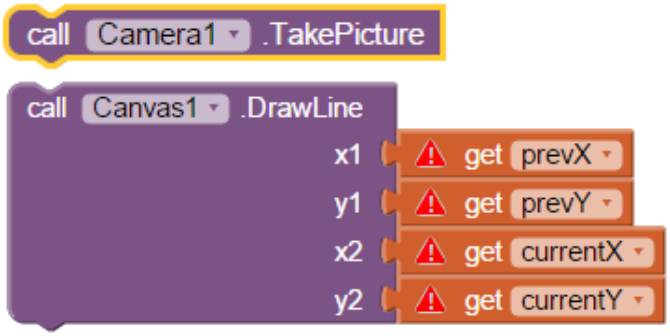


Examples

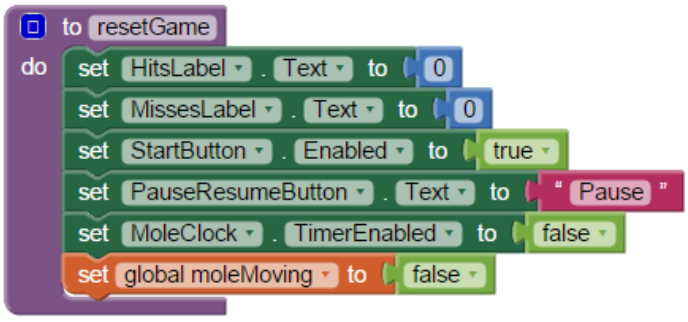
Event Handlers



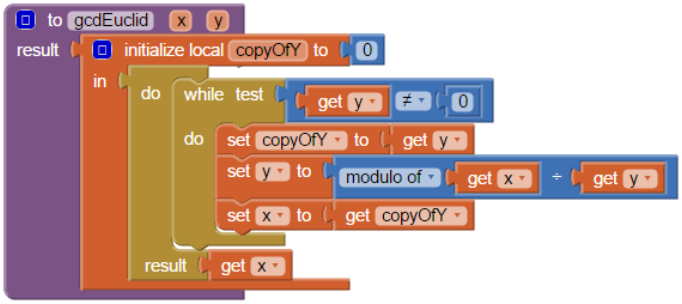
Methods



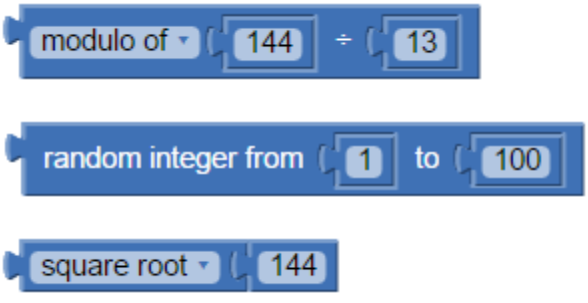
Programmer-Defined (without Result)



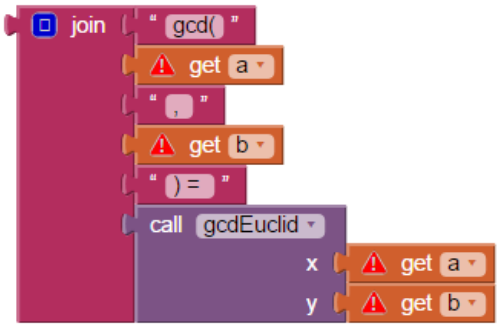
Programmer-Defined (with Result)



Mathematical Functions



String (Text) Functions

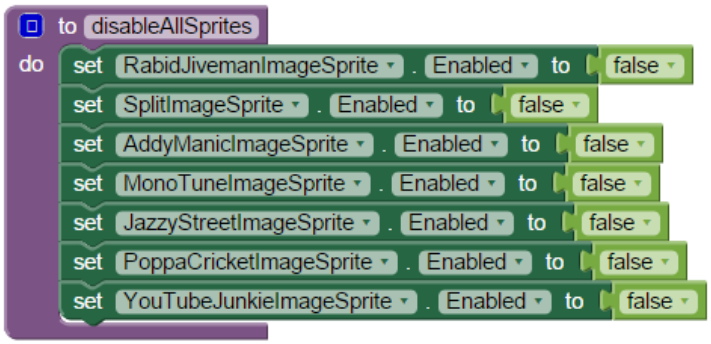


Advantages of Programmer-Defined Procedures

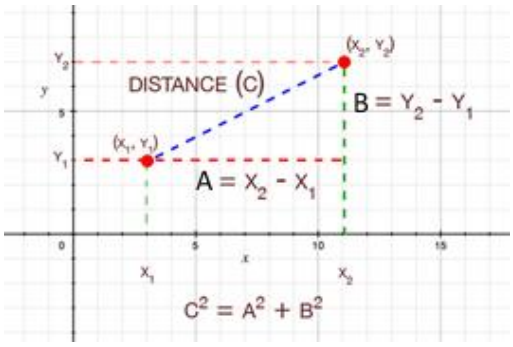
- Eliminate repetitive code by ***encapsulating*** (i.e. enclosing) the solution to a problem in a procedure.
  - The program is more concise and better organized, making it much easier to understand.
  - Debugging is much easier because bugs are localized to a particular procedure.
  - Making changes is much easier because modifications only need to be made within a particular procedure.
- Well-designed procedures can be reused in other programs. (The problem is solved once and for all!)

Examples

1. Write a procedure that disables all the sprites in the "SplitterBust" game.



2. Write a procedure (with a result) that calculates the length of a line segment given the co-ordinates of its endpoints.



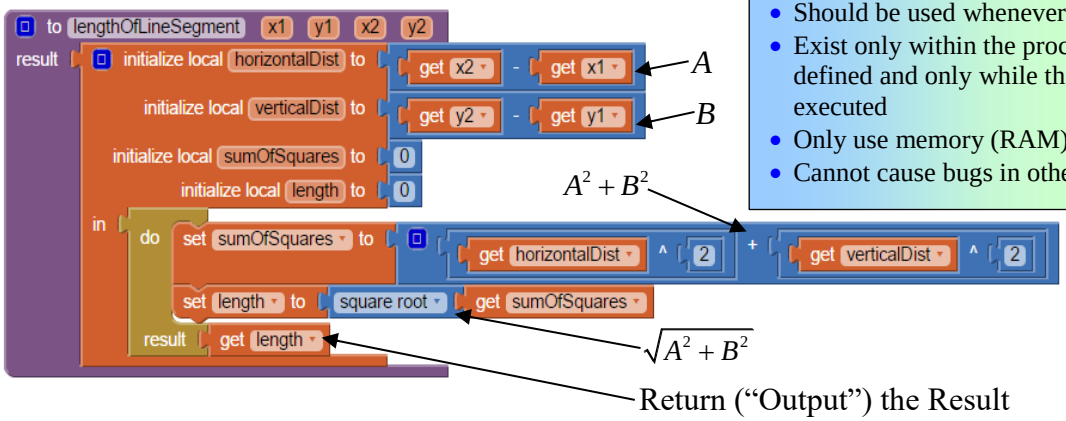
A = horizontal distance = difference of the *x*-co-ordinates

B = vertical distance = difference of the *y*-co-ordinates

C = distance between the two points = hypotenuse of right triangle

By the Pythagorean Theorem, $C^2 = A^2 + B^2$ and therefore,

$$C = \sqrt{A^2 + B^2}$$

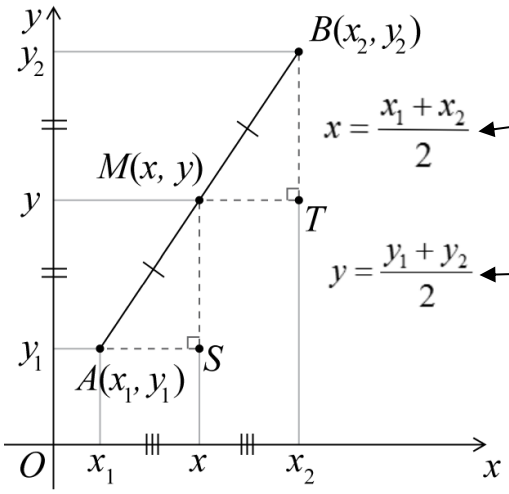


Local Variables

- Should be used whenever possible
- Exist only within the procedure in which they are defined and only while the procedure is being executed
- Only use memory (RAM) while procedure is running
- Cannot cause bugs in other procedures

Exercises

1. Write a procedure with a result that takes the endpoints of a line segment as inputs and returns (i.e "outputs") the midpoint of the line segment. **Hint:** Use a list to store the co-ordinates of the midpoint.



The *x*-co-ordinate of the midpoint is the average of the *x*-co-ordinates of the endpoints. This means that it is located exactly halfway between the *x*-co-ordinates of the endpoints.

The *y*-co-ordinate of the midpoint is the average of the *y*-co-ordinates of the endpoints. This means that it is located exactly halfway between the *y*-co-ordinates of the endpoints.

2. Write a procedure with a result that takes the coefficients of a quadratic function $f(x) = ax^2 + bx + c$ as inputs and returns (i.e "outputs") the roots (solutions) of the quadratic equation $ax^2 + bx + c = 0$.

Hint: Use a list to store the roots of the quadratic equation.

Challenge: Return the roots even when there are no real roots. Use *i* to represent $\sqrt{-1}$.

Theory

The ***quadratic formula*** $x = \dfrac{-b \pm \sqrt{b^2 - 4ac}}{2a}$ is the general solution of the ***quadratic equation*** $ax^2 + bx + c = 0$.

Examples

1. $x^2 + 5x + 6 = 0 \quad \rightarrow \quad x = \dfrac{-5 \pm \sqrt{5^2 - 4(1)(6)}}{2(1)} \quad \rightarrow \quad x = -2 \text{ or } x = -3$ (two real roots)

2. $x^2 - 2x + 1 = 0 \quad \rightarrow \quad x = \dfrac{-(-2) \pm \sqrt{(-2)^2 - 4(1)(1)}}{2(1)} \quad \rightarrow \quad x = 2$ (one real root)

3. $x^2 + 4x + 5 = 0 \quad \rightarrow \quad x = \dfrac{-4 \pm \sqrt{4^2 - 4(1)(5)}}{2(1)} = \dfrac{-4 \pm \sqrt{-4}}{2} = \dfrac{-4 \pm 2\sqrt{-1}}{2} = \dfrac{-4}{2} \pm \dfrac{2\sqrt{-1}}{2} = -2 \pm \sqrt{-1}$

In this case there are no real roots but there are two ***complex*** roots: $x = -2 + i$, $x = -2 - i$

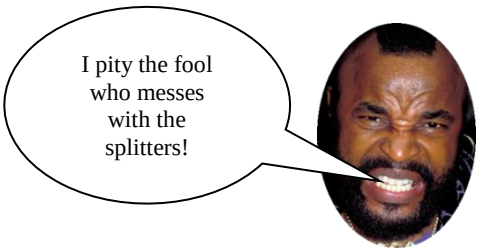
Genuine Problem Solving
SplitterBust: A Variation of MoleMash/MileyBash

Understanding the Nature of the Game

SplitterBust is a variation of the much loved MoleMash and MileyBash games. Instead of one or two sprites moving about a canvas, SplitterBust has *seven* sprites moving simultaneously! Each sprite contains a picture of one of the members of the notorious rappers known as the *Split Personalities*.

In addition to more sprites moving at the same time, *SplitterBust* also has ***ALL*** the following features:

- Each splitter moves at a *different* speed. Some move slowly while others move more quickly. The faster-moving sprites move *in front of* the slower-moving ones (see the "Z" property).
- The slow-moving sprites move in straight lines. They change direction only when bouncing off an edge.
- The fast-moving sprites move rapidly from one random location to another. Unlike the slow-moving sprites, the path taken from one random location to another is not seen by the user. The sprites simply "pop" quickly from one location to another.
- Each time a picture of one of the slow-moving splitters is touched, the picture disappears for exactly *thirty seconds*. Immediately after the thirty-second period elapses, the splitter reappears moving *twenty-five percent faster*. Once the slow-moving sprite achieves a speed that is at least *twice* that of its original speed, its behaviour changes to that of a fast-moving sprite.
- Each time a picture of one of the fast-moving splitters is touched, the picture disappears for exactly *thirty seconds*. Immediately after the thirty-second period elapses, the splitter reappears moving *twenty-five percent slower*. Once the fast-moving sprite achieves a speed that is *half* that of its original speed or slower, its behaviour changes to that of a slow-moving sprite.
- If the user is fast enough to make all the pictures disappear before any of them reappear, the game ends and the user wins.
- If any of the pictures are still in motion after ten minutes of play, the game ends and the user loses the game. In this case, Mr. T's picture appears and the line, "I pity the fool who messes with the splitters!" is played over the speakers.



Planning BEFORE Coding

Programming is the *art* of "*TEACHING*" computers how to solve problems.
You *CANNOT* "teach" a computer to solve a problem that you DO NOT know how to solve!

A. Important Components and their Properties/Methods/Events

ImageSprite

Properties: Picture, Enabled, Interval, Rotates, Visible, Heading, X, Y, Z, Speed, Width, Height

Methods: Bounce, CollidingWith, MoveIntoBounds, MoveTo, PointInDirection, PointTowards

Events: CollidedWith, Dragged, EdgeReached, NoLongerCollidingWith, Touched, TouchDown, TouchUp, Flung

Clock

Properties: TimerEnabled, TimerInterval, TimerAlwaysFires

Methods: SystemTime, Now, MakeInstant, MakeInstantFromMillis, GetMillis, … (too many to list)

Events: Timer

References

http://ai2.appinventor.mit.edu/reference/components/animation.html
http://ai2.appinventor.mit.edu/reference/components/sensors.html

B. THINKING about the Features of SplitterBust

Write a brief explanation of *how* you will implement *each* of the features described above.

1.
2.
3.
4.
5.
6.
7.

Building the App

(a) Implement only ONE feature at a time.

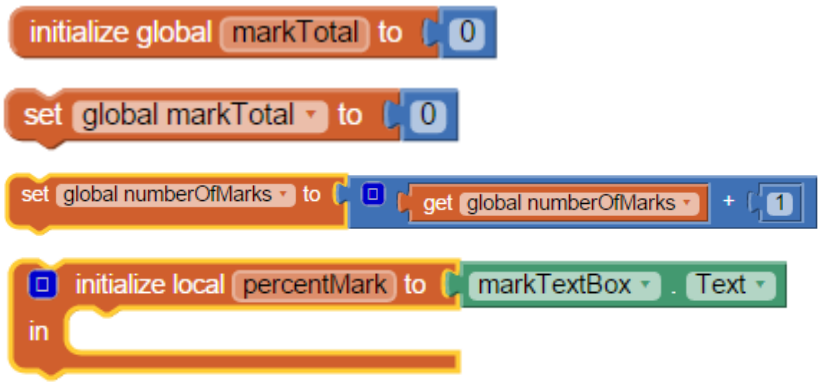
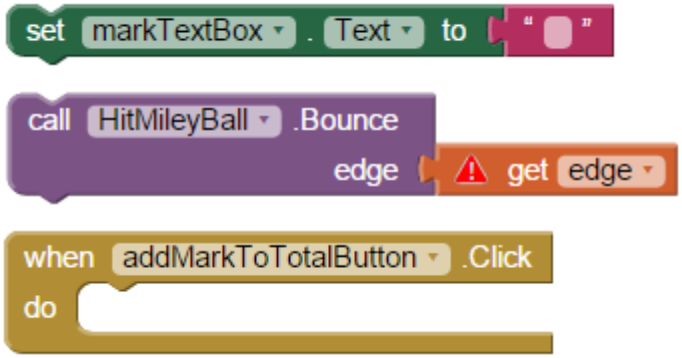
(b) Test thoroughly after adding each feature. Do not wait until it is too late!

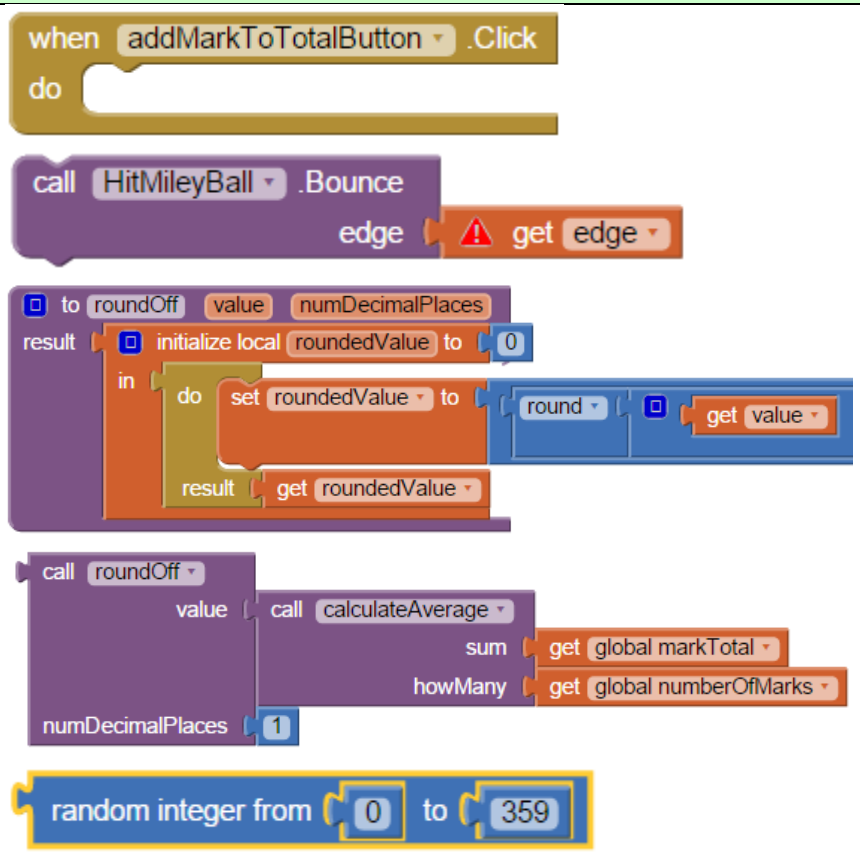
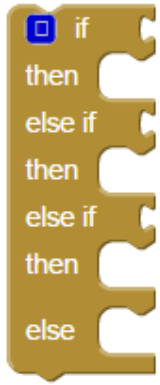
(c) Reflect upon your work. Could your coding be made simpler, more efficient or easier to understand?

(d) Be creative! Think of new features that could make the game more challenging or more enjoyable.

App Inventor: Review of Main Ideas

Complete the following table. Please ensure that your responses are brief, to the point and accurate.

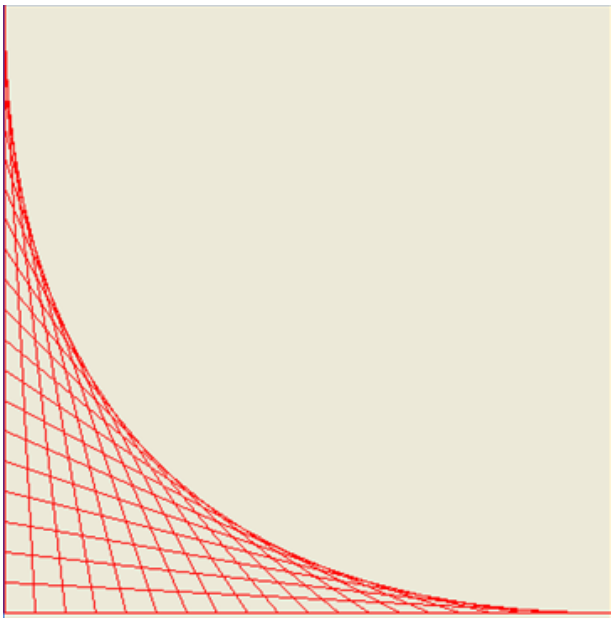
Concept	***Appearance/Examples of Use***	***Explanation***
Variable • Global • Local • Parameter		Address the following in your answer: ***purpose*** of variables, ***difference*** between "get" and "set," why it is preferable to use local variables whenever possible, the circumstances under which global variables ***must*** be used
Component • Properties • Methods • Events		Address the following in your answer: ***purpose*** of properties, methods, events, ***why*** a property can be considered a variable that belongs to a component, ***why*** a method can be considered a procedure that belongs to a component

Concept	*Appearance/Examples of Use*	*Explanation*
Procedure • Event Handler • Method • Programmer-defined (with and without result) • Built-in Function/Proc-edure		Address the following in your answer: ***purpose*** of procedures, ***purpose*** of parameters, how an event handler differs from other procedures, how a method differs from other procedures, how a programmer-defined procedure differs from a built-in procedure, meaning of ***call***
Conditional • If…Then • If…Then Else • If…Then ElseIf … Then ElseIf … Then … Else		

Concept	***Appearance/Examples of Use***	***Explanation***
Math Operations • +, –, *, /, ^ • Random Numbers • Rounding • Quotient and Remainder		Address the following in your answer: ***How*** does a computer generate random numbers? Are these numbers truly random? What is the difference between "/" and "quotient?" On what type of data can mathematical operations be applied?
Types of Data • Numeric • Text ("String") • Logical		Address the following in your answer: ***Why*** are these three types of data incompatible with one another?

Concept	*Appearance/Examples of Use*	*Explanation*
List • Create list • Add item to list • Delete from list • Select item from list • Search for an item in a list		Address the following in your answer: ***Why*** would you use a list? What does a list allow you to do?

Introduction to Loops: Line Drawing Problems



Introduction

Although we most definitely *perceive* a curve in the picture at the right, the picture itself was created by drawing only a series of ***straight lines***. No curves were actually drawn! Why then do we seem to see a curve? The answer to this question has everything to do with how our brains *construct* the "reality" that we "see." Since every straight line in this diagram is ***tangent*** to the curve that we "see," our brains take all the points of tangency and "connect" them to create the perception of a curve.

Problem

Use App Inventor to create an app that can generate the diagram at the right.

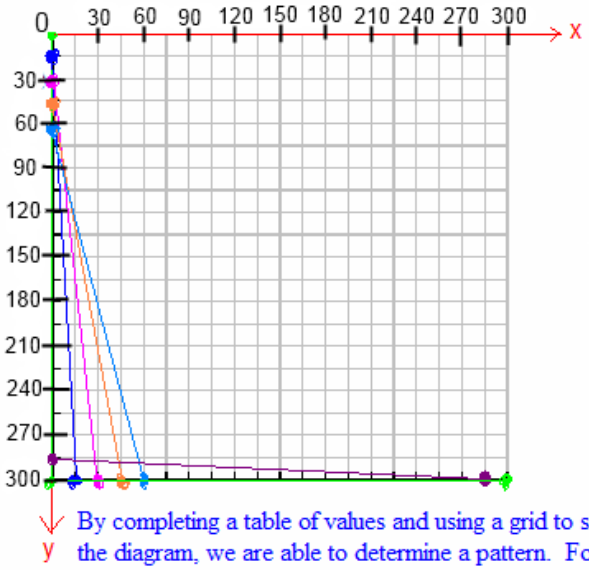
Hints

1. Use a "Canvas" component.
2. In addition, your app will need to use the "DrawLine" method of a "Canvas" component.
3. You need to understand the co-ordinate system that is used for "Canvas" components.
4. There is a definite pattern that governs how the lines are drawn. Before attempting to create blocks for your app, you must figure out the pattern!

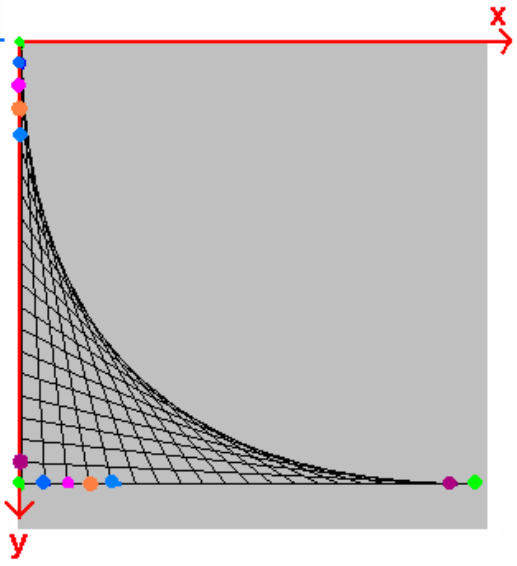
Explanation

The main idea behind reproducing pictures like the one given above is to use the idea of "reverse engineering." That is, we try to "take apart" the picture to understand how it was created in the first place.

For convenience, the canvas size is set to 300 pixels by 300 pixels. This not only fits nicely on most cell phone screens but it also takes advantage of the fact that the number 300 has many divisors (i.e. 1, 2, 3, 4, 5, 6, 10, 12, 15, 20, 25, 30, 50, 60, 75, 100, 150, 300).



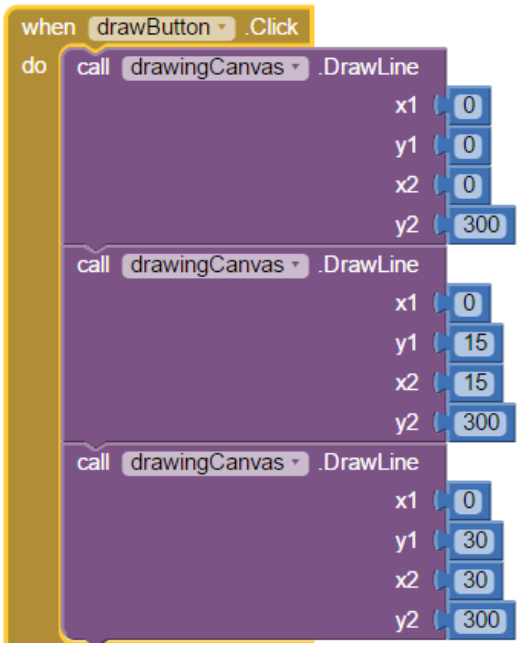
Start	End
(0,0)	(0,300)
(0,15)	(15,300)
(0,30)	(30,300)
(0,45)	(45,300)
(0,60)	(60,300)
(0, I)	(I,300)
(0,285)	(285,300)
(0,300)	(300,300)



By completing a table of values and using a grid to sketch a few of the lines in the diagram, we are able to determine a pattern. For each line in the diagram, the y-co-ordinate of the "start" point is equal to the x-co-ordinate of the "end" point. In addition, we observe that the x-co-ordinate of the "start" point is always equal to 0 and the y-co-ordinate of the "end" point is always equal to 300 .

Generating the Pictures in App Inventor

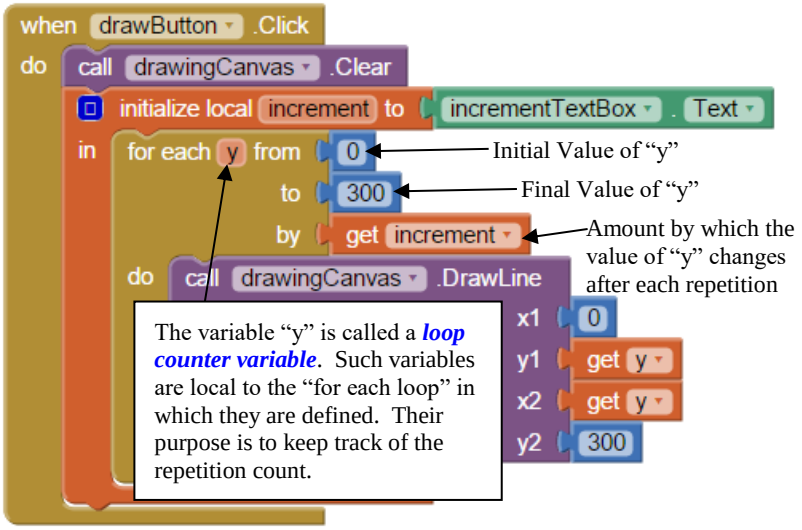
Method 1- Easy to Understand but Tedious



-
-
-

The remaining ***eighteen*** blocks have been omitted to save space.

Method 2 – Harder to Understand but much more Efficient



Explanation of the more Efficient Method

The "for range" block is an example of what computer scientists call ***counted loops***. Counted loops are used to repeat one or more instructions a set number of times. The following explains the details of the above "for each" loop:

- The call to the "DrawLine" method is shown only ***once*** BUT it is ***repeated*** a number of times determined by the values of "from," "to" and "by." Assuming that the value of "by" is 15, the loop repeats exactly ***twenty-one*** times.
- The variable "y" is called a ***loop counter variable***. Its value is changed automatically after every repetition.
- The amount by which the loop counter's value changes is specified by the value of "by." To generate the picture on the first page of this section, the value of "by" must be set to 15. In the above example, the value of "y" increases by 15 after each repetition because the value of "by" is 15.
- The value of "y" ***ranges*** from 0 to 300 because the "from" and "to" values are set respectively to 0 and 300. This explains the name of the "for each" block.
- Thus, assuming that the value of "increment" is 15, "y" takes on the values 0, 15, 30, 45, …, 270, 285, 300, after which the loop terminates (i.e. stops repeating).

Summary

Counted Loops ("For Loops")

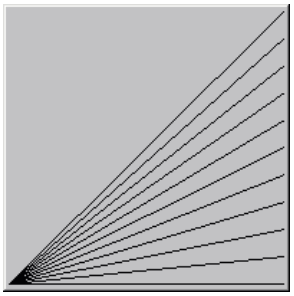
These are used when the number of repetitions is known at ***design-time*** (i.e. while the program is being designed) or can be calculated at ***run-time*** (i.e. when the program is running). Whether looping continues or terminates is based on a ***count***. The number of repetitions of such loops is always ***predictable***.

Analogy: Add three teaspoons of sugar to the coffee. Repeat the act of adding one teaspoon of sugar ***3 times***.

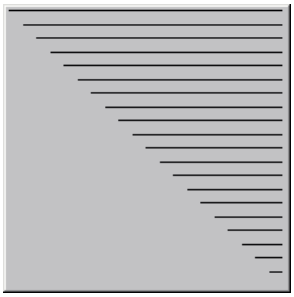
Problems

Use graph paper and a table of values to determine the pattern that is used to create each picture. Then create App Inventor apps that generate each of the following pictures. Your app must be able to draw the picture all at once and in an animated way (i.e. one line at a time)

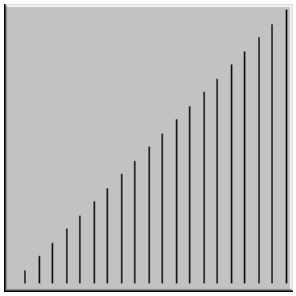
1.



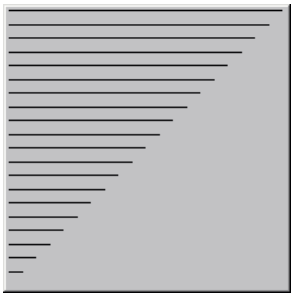
2.



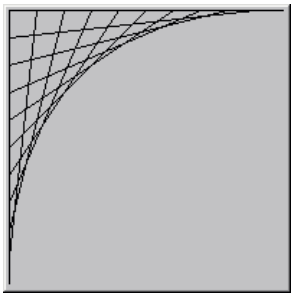
3.



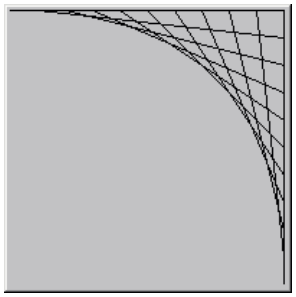
4.



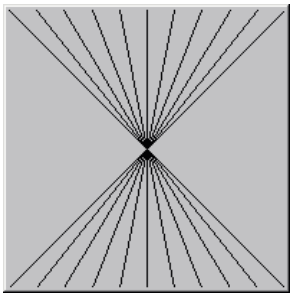
5.



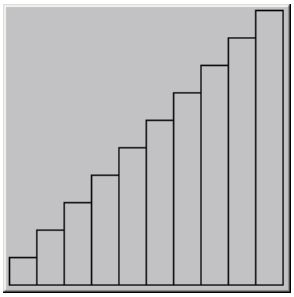
6.



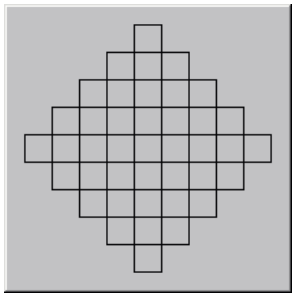
7.



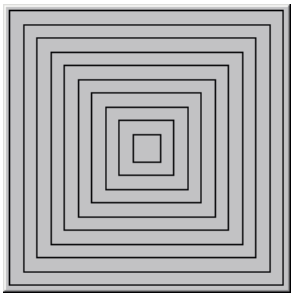
8.



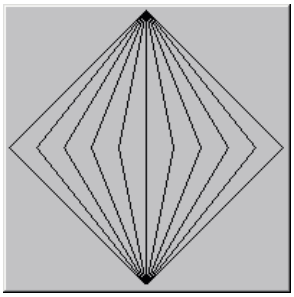
9.



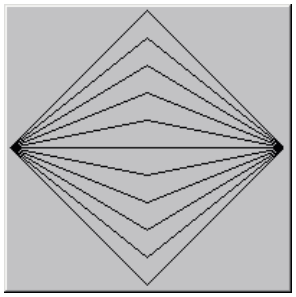
10.



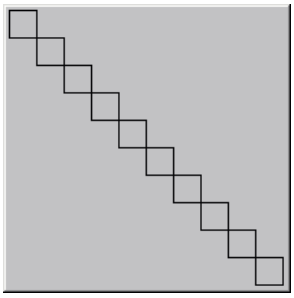
11.



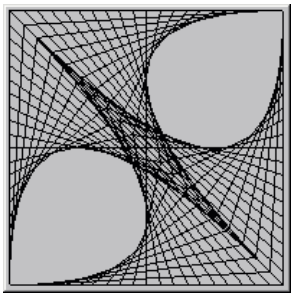
12.



13.



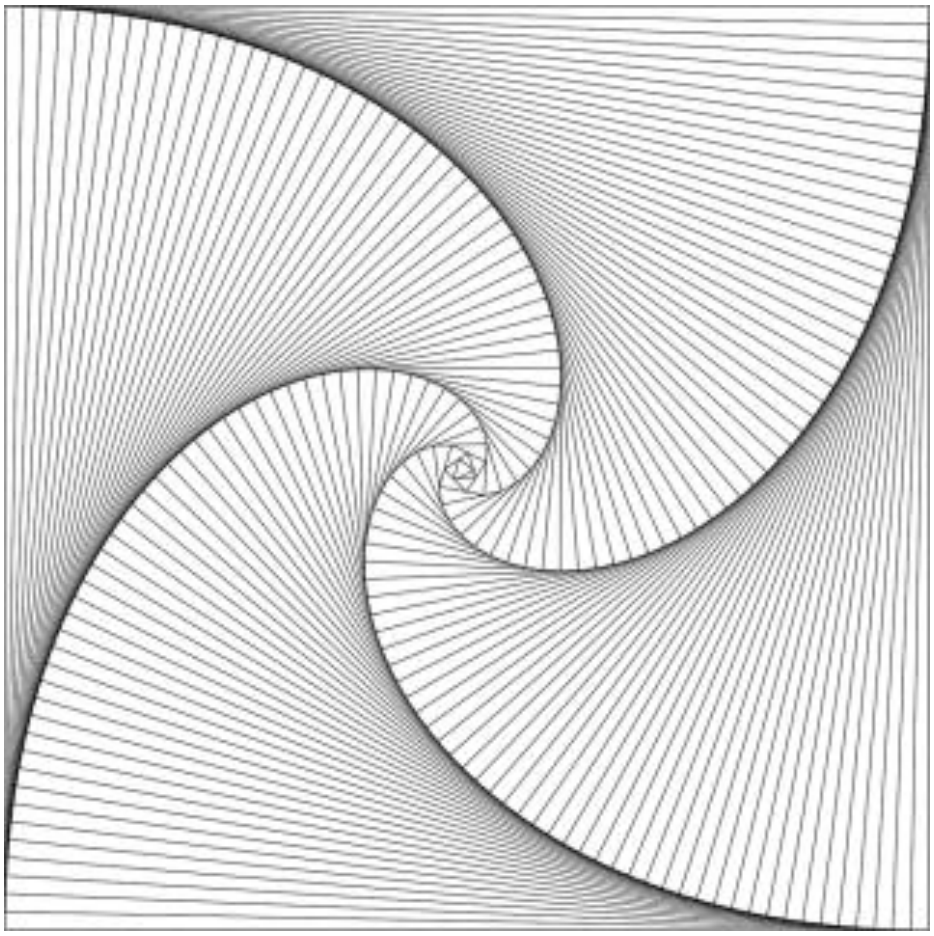
14.



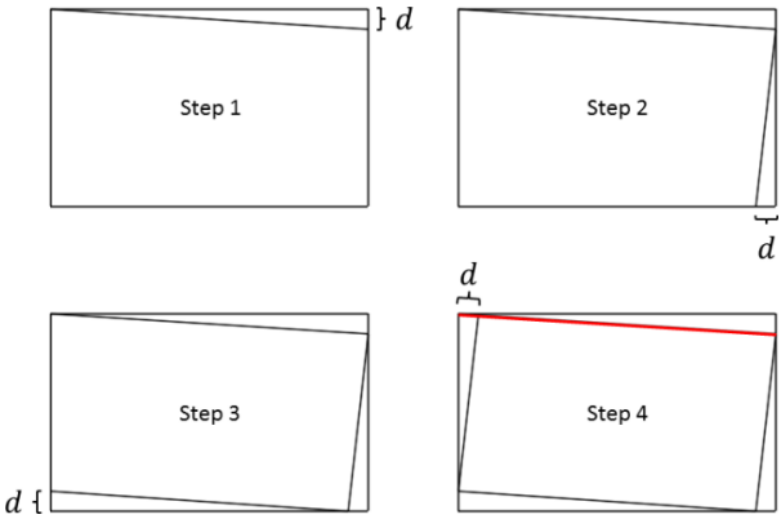
Line Drawing Challenge – The Placemat Spiral

Challenge

Create an App Inventor app that generates the following picture (often called a "placemat spiral") using nothing but straight lines!



Hints

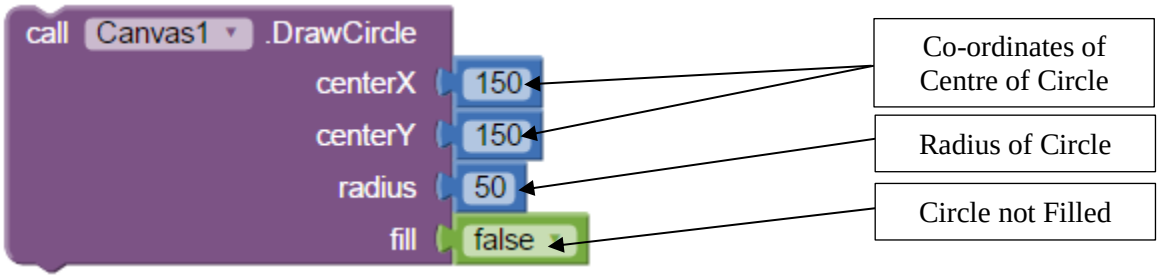


For more information, see http://larc.unt.edu/ian/art/4ants/ .

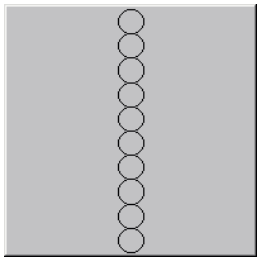
Circle Drawings

Exercises

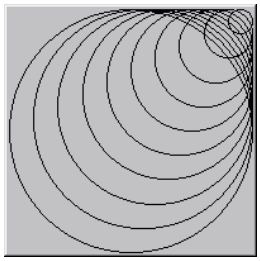
Create App Inventor apps that generate each of the following pictures. Your app must be able to draw the picture all at once and in an animated way (i.e. one circle at a time)



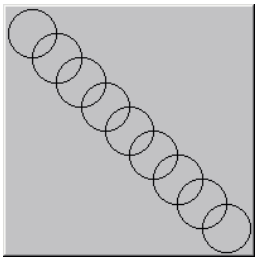
1.



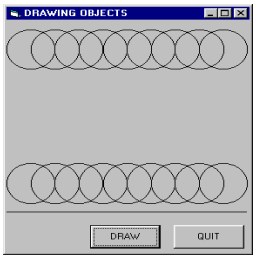
2.



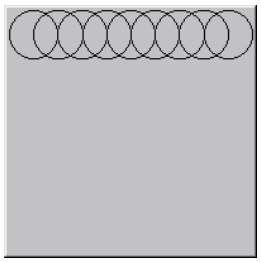
3.



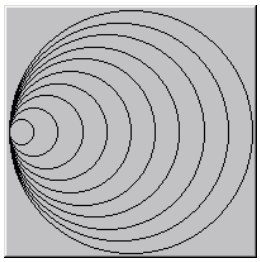
4.



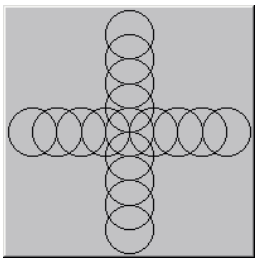
5.



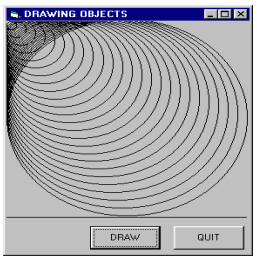
6.



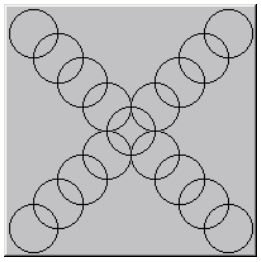
7.



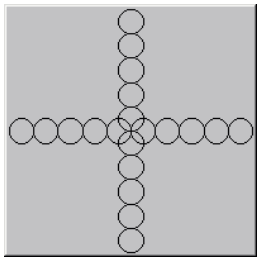
8.



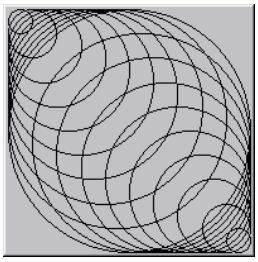
9.



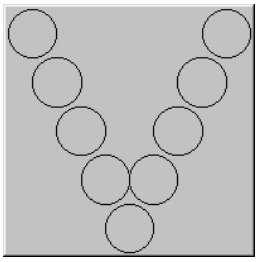
10.



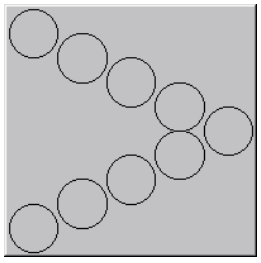
11.



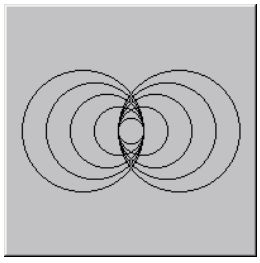
12.



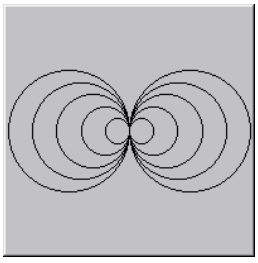
13.



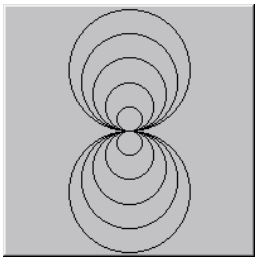
14.



15.



16.



GPA: App Inventor Challenge Problem

1. Most high schools in the United States, as well as almost all universities in North America, use a grading system known as the *grade point system*. In the grade point system, letter grades (i.e. A, B, C, etc.) are replaced with numeric values. This allows an average, known as the *grade point average* or GPA, to be computed. The actual scale used for grade point averaging purposes varies from jurisdiction to jurisdiction and from institution to institution. The one shown below is used at the University of Toronto.

Percentage Grade	***Grade Point Value***
85% – 100%	4.0
80% – 84%	3.7
77% – 79%	3.3
74% – 76%	3.0
70% – 73%	2.7
67% – 69%	2.3
64% – 66%	2.0
60% – 63%	1.7
57% – 59%	1.3
54% – 56%	1.0
50% – 53%	0.7
0% – 49%	0.0

Example

Subject	***Percentage Mark***	***Grade Point Value***
Math	76%	3.0
Computer Science	84%	3.7
Chemistry	63%	1.7
Physics	45%	0.0
English	49%	0.0

$$\text{GPA} = \frac{3.0+3.7+1.7+0.0+0.0}{5} = 1.68 < 60\%$$

$$\text{Percent Average} = \frac{76+84+63+45+49}{5} = 63.4\%$$

Create an App Inventor app that allows the user to enter any number of percentage grades. After the user clicks "Submit," the app displays the user's GPA as well as his/her percentage average.

2. Improve your app in the following ways:

(a) Calculate the averages "on the fly." (The averages are updated every time a mark is entered.)

(b) Allow the user to delete one or more of the entered marks.

(c) Display all the marks and courses entered, as well as the GPA and percent average.

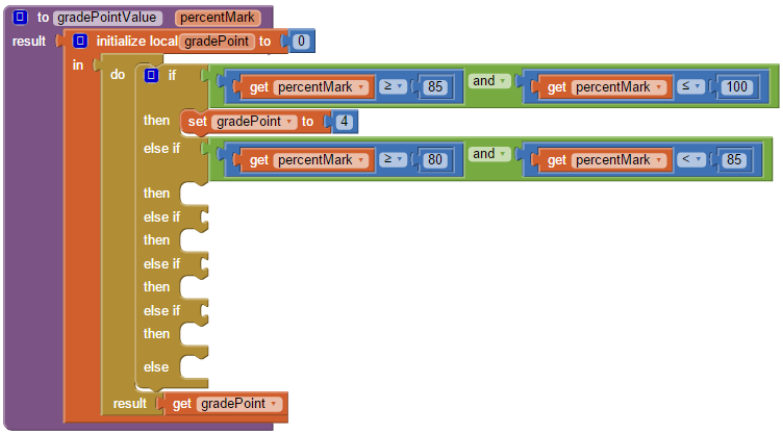
(d) Anything else that comes to mind!

Hints

1. Create a programmer-defined procedure like the one shown at the right.

2. Use an "if" statement within the procedure that has a structure like the one shown at the right.

3. Use lists to "remember" the courses and marks entered by the user.

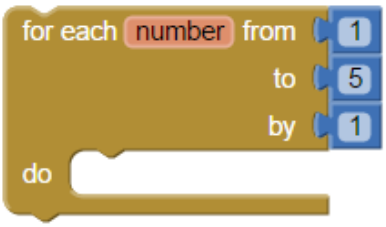
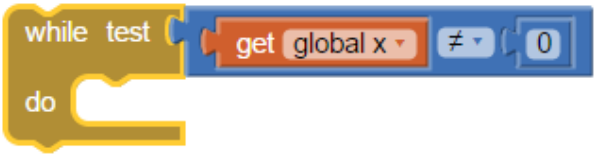


Programming Problems whose Solutions Require the use of Counted ("For") or Conditional ("While") Loops

Algorithm

- An ***algorithm*** is a systematic procedure by which a problem is solved. Long division is an example.
- In cooking/baking/mixing drinks etc, algorithms are called ***recipes***.
- Many of the problems given below can be solved using an ***exhaustive search*** (aka ***brute-force search***) algorithm. An algorithm that employs an exhaustive search systematically checks ***all possible candidates*** for the solution to see which of them, if any, satisfies the statement of the problem.
- Exhaustive search is guaranteed to find a solution if one exists. However, when the number of possible candidates is ***very large***, brute-force methods are excruciatingly slow. Shortly, we'll be investigating a better solution to (g) to help us understand the limitations of brute-force algorithms.

Looping Structures: Counted versus Conditional

Counted Loops ("For")	Conditional Loops ("While")
 • Use a counted loop when the # of repetitions is known at design-time • Analogy: "Add three spoonfuls of sugar to the coffee."	 • Use a conditional loop when the # of repetitions is ***not*** known at design-time • Analogy: "Keep stirring the coffee while the sugar has not dissolved yet."

Complete the following table. Then write App Inventor programs to solve each problem.

Programming Problem	***Can you write a solution that only requires a counted loop? Explain.***
(a) Write a program to calculate the ***sum*** of all positive even integers less than or equal to 1000.	Yes / No (Circle One) Why?
(b) Write a program to calculate the ***sum*** of all positive odd integers until the sum exceeds 1000.	Yes / No (Circle One) Why?
(c) Write a program to calculate the ***product*** of all positive integers divisible by 5 and less than or equal to 645. (What happens if you try a value ≥ 650?)	Yes / No (Circle One) Why?
(d) Write a program to calculate the ***product*** of all positive integers divisible by 5 while the product is less than or equal to 1000000.	Yes / No (Circle One) Why?
(e) An integer is called ***prime*** if it has exactly two divisors, one and itself. The following is a list of the first 10 prime numbers: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29 Write a program that determines whether a given number is prime. **(Exhaustive Search)**	Yes / No (Circle One) Why?

Programming Problem	***Can you write a solution that only requires a counted loop? Explain.***
(f) A ***proper divisor*** of an integer is any integer that divides evenly into the integer, except for the number itself. For example, the proper divisors of 12 are 1, 2, 3, 4 and 6. A number is called ***perfect*** if the sum of its ***proper divisors*** is equal to the number itself. Two examples of perfect numbers are 6 and 28 because $6 = 1 + 2 + 3$ and $28 = 1 + 2 + 4 + 7 + 14$. Write a program that determines whether a given number is perfect. **(Exhaustive Search)**	Yes / No (Circle One) Why?
(g) Write a program that finds the ***greatest common divisor*** of any two integers. For example, the greatest common divisor (GCD) of 24 and 40 is 8. **(Exhaustive Search)**	Yes / No (Circle One) Why?
(h) Write a program that finds the ***least common multiple*** of any two integers. For example, the least common multiple (LCM) of 24 and 40 is 120. **(Exhaustive Search)**	Yes / No (Circle One) Why?
(i) The numbers 220 and 284 are called an ***amicable pair*** because the sum of the proper divisors of 220 is 284 and the sum of the proper divisors of 284 is 220. Write a program that finds ***all*** amicable pairs within the search range $1 \leq x \leq 1000000$. **(Exhaustive Search)**	Yes / No (Circle One) Why?
(j) Horses cost \$10, pigs cost \$3 and rabbits cost only \$0.50. A farmer buys 100 animals for \$100. How many of each animal did he buy? Write a program to ***search*** for the solution to this problem. **(Exhaustive Search)**	Yes / No (Circle One) Why?

Euclid and the GCD

Definition of gcd

The *Greatest Common Divisor* (gcd) of two positive integers is the largest integer that divides both integers exactly.

Examples

- $\gcd(8,12)=4$ because 4 is the largest integer that divides into both 8 and 12
- $\gcd(14,42)=14$ because 14 is the largest integer that divides into both 14 and 42
- $\gcd(9,28)=1$ because 1 is the largest integer that divides into both 9 and 28

Brute Force (Exhaustive Search) Algorithm for Computing the GCD of Two Integers

The most obvious method for computing the gcd of two integers is to perform an exhaustive search of the set of all possible divisors. The search may not find any divisors other than one or it may find several divisors. In either case, the gcd must be the largest divisor found. This is illustrated below.

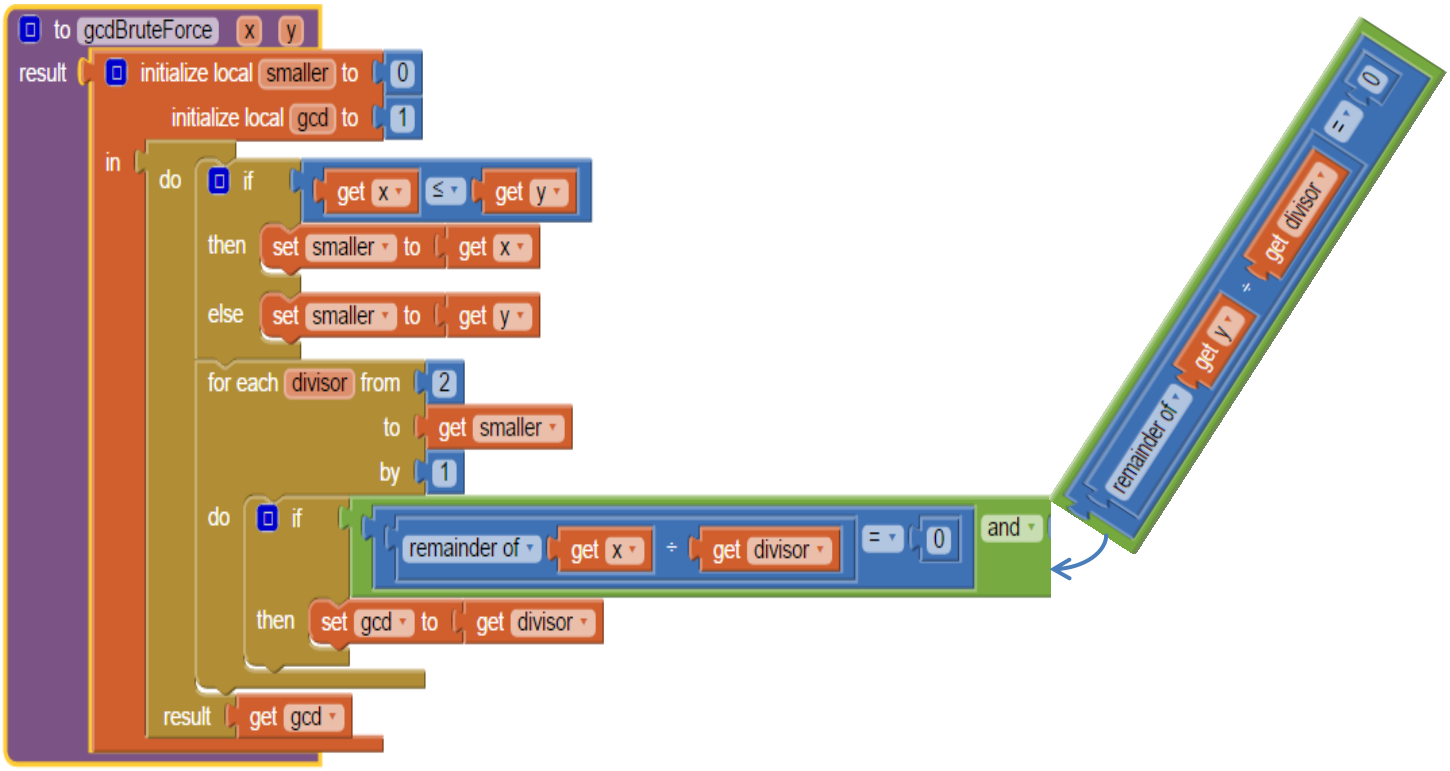
- **`a, b:`** These variables store the two integers for which the gcd must be found. The values of these two variables do not change throughout the execution of the code.
- **`y:`** This variable stores the values of all the integers that we try to divide into both **`a`** and **`b`**. The value of this variable is controlled by a "**for each**" loop.
- **`gcd:`** This variable stores the greatest common divisor found so far. Before entering the loop, it is initialized to 1 because 1 divides into every number. If no other common divisor is found by the "**for each**" loop, the value of 'gcd' never changes, meaning that its final value will be 1 (see the second table).

	a	b	y	Remainder obtained when 'a' is divided by 'y'	Remainder obtained when 'b' is divided by 'y'	gcd
Values Before Entering Loop →	8	12	?	?	?	1
	8	12	2	0	0	2
	8	12	3	2	0	2
	8	12	4	0	0	4
	8	12	5	3	2	4
	8	12	6	2	0	4
	8	12	7	1	5	4
	8	12	8	0	4	4
Values After Exiting Loop →	8	12	9	1	3	4

	a	b	y	Remainder obtained when 'a' is divided by 'y'	Remainder obtained when 'b' is divided by 'y'	gcd
Values Before Entering Loop →	9	28	?	?	?	1
	9	28	2	1	0	1
	9	28	3	0	1	1
	9	28	4	1	0	1
	9	28	5	4	3	1
	9	28	6	3	4	1
	9	28	7	2	0	1
	9	28	8	1	4	1
	9	28	9	0	1	1
Values After Exiting Loop →	9	28	10	1	3	1

App Inventor Code for "Brute Force" GCD Algorithm

The following is a procedure that calculates and displays the GCD of the integers "x" and "y." Study the code and then answer the questions found below.



Questions

1. Explain the purpose of the "if" statement that immediately precedes the "**for each**" loop.

2. Why does the search for common divisors end at the smaller of "x" and "y?"

3. The greatest common divisor of 1,000,000,000 and 500,000,000 is 500,000,000. How many times do the instructions within the "**for each**" loop need to be repeated before the procedure finds this answer? Comment on the efficiency of using an exhaustive search to compute the gcd of a pair of very large numbers. Can you think of any ways of improving the efficiency of the process?

Description of Euclid's (Fast) Method for Computing the GCD of Two Integers

Background

More than 2000 years ago, Euclid published an algorithm for finding the GCD of two numbers. His version was strictly geometric since algebra had not been invented yet, but the algebraic version is described below.

Summary

The Euclid algorithm can be expressed concisely by the following recursive formula:

gcd(*a*, *b*) = gcd(*b*, *a* mod *b*)

Note: ***a* mod *b*** means the remainder obtained when *a* is divided by *b*.

Example

Here is an example of Euclid's algorithm in action.

Find the GCD of 2322 and 654.

gcd(2322, 654) = gcd(654, 2322 mod 654) = gcd(654, 360)
gcd(654, 360) = gcd(360, 654 mod 360) = gcd(360, 294)
gcd(360, 294) = gcd(294, 360 mod 294) = gcd(294, 66)
gcd(294, 66) = gcd(66, 294 mod 66) = gcd(66, 30)
gcd(66, 30) = gcd(30, 66 mod 30) = gcd(30, 6)
gcd(30, 6) = gcd(6, 30 mod 6) = gcd(6, 0)
gcd(6, 0) = 6

Therefore, gcd(2322,654) = 6.

a	b
2322	654
654	360
360	294
294	66
66	30
30	6
6	0

Essentially, the Euclidean algorithm performs the following two steps:

1. The value of 'b' is copied to 'a.'
2. The value of 'b' changes to the value of 'a mod b" (the original value of 'a' must be used, i.e. the value of 'a' before step 1 was carried out).

This process continues until the value of 'b' is zero.

Your Task

1. Use Euclid's method to calculate gcd(4896, 830).

a	b

2. How many repetitions would be required by the "slow GCD" algorithm to compute gcd(4896, 830)?

3. Try to write App Inventor code to implement the Euclid GCD algorithm. Test your code thoroughly and debug if necessary.

Enrichment Questions

1. Who was Euclid? Why do historians of mathematics consider him an extremely important figure?

2. Prove that gcd(*a*, *b*) = gcd(*b*, *a* mod *b*).

Solutions to Selected Problems Requiring Loops

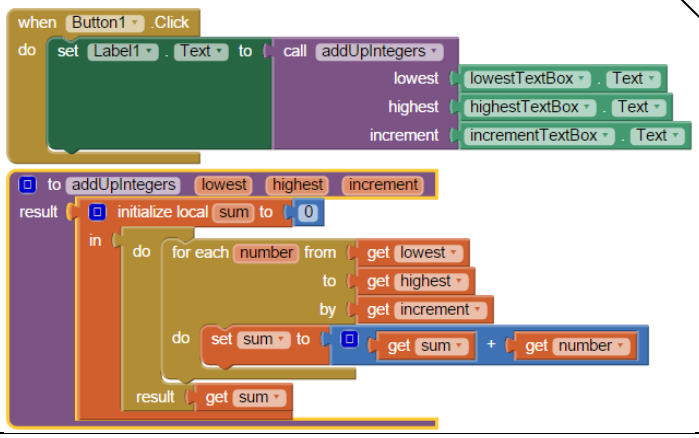
Problem	App Inventor Solution	Notes
(a) Write a program to calculate the ***sum*** of all positive even integers less than or equal to 1000.	The procedure "addUpIntegers" can find the sum of any *arithmetic series* (within the numeric range available in App Inventor). Arithmetic series take the following form: $a+(a+d)+(a+2d)+\cdots+(a+nd)$ The series 2+4+6+…+1000 is an example of an arithmetic series.	Think of this algorithm as the "**cash register algorithm**." Values are successively added to the total until the final total is obtained. For this problem, a "for" loop ***can*** be used because the number of repetitions is known at design-time.
(b) Write a program to calculate the sum of all positive odd integers until the sum exceeds 1000.	Unlike "for each" loops, "while" loops ***do not*** have a built-in loop counter variable. However, it's a very simple matter to include a variable that behaves just like a loop counter.	The solution to this problem also makes use of the **cash register algorithm**. However, this time a "for" loop ***cannot*** be used because the number of repetitions is ***not known*** at design-time. The instructions within the body of the loop are repeated as long as the sum remains smaller than or equal to 1000.

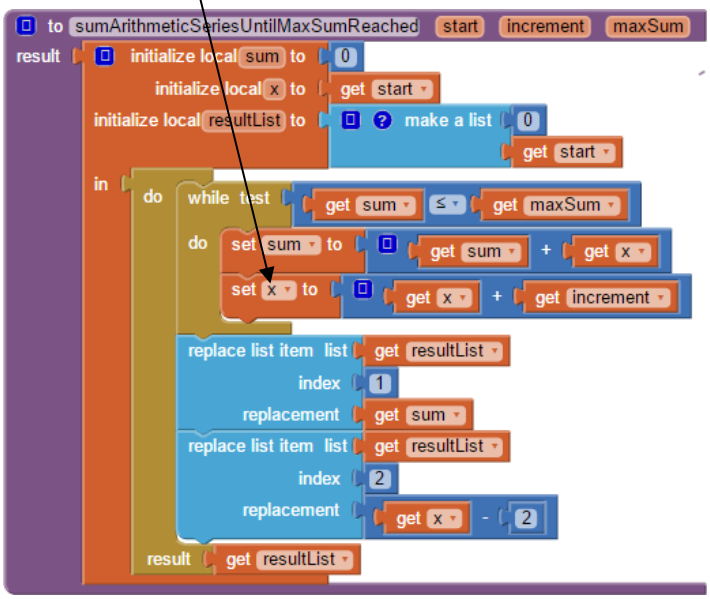
Values of variables before entering loop

Values of variables after each repetition of the loop

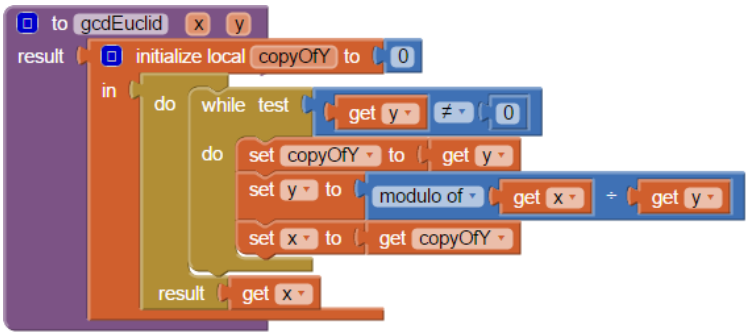
Values of variables after exiting loop

number	sum
–	0
2	2
4	6
6	12
8	20
.	.
.	.
.	.
998	249500
1000	250500
–	250500





x	sum
−1	0
1	1
3	4
5	9
7	16
.	.
.	.
.	.
61	961
63	1024
–	1024

<table>
<tr><th>Problem</th><th>App Inventor Solution</th><th>Notes</th></tr>
<tr>
<td>(g) Write a program that finds the greatest common divisor of any two integers. For example, the greatest common divisor (gcd) of 24 and 40 is 8. (Exhaustive Search)</td>
<td>See page 36.</td>
<td>For this example, suppose that x=12 and y=20. Then ‘smaller’ has a value of 12.
<table>
<tr><th>divisor</th><th>gcd</th></tr>
<tr><td>–</td><td>1</td></tr>
<tr><td>2</td><td>2</td></tr>
<tr><td>3</td><td>2</td></tr>
<tr><td>4</td><td>4</td></tr>
<tr><td>5</td><td>4</td></tr>
<tr><td>6</td><td>4</td></tr>
<tr><td>7</td><td>4</td></tr>
<tr><td>8</td><td>4</td></tr>
<tr><td>9</td><td>4</td></tr>
<tr><td>10</td><td>4</td></tr>
<tr><td>11</td><td>4</td></tr>
<tr><td>12</td><td>4</td></tr>
<tr><td>–</td><td>4</td></tr>
</table>
The final value of the variable “gcd” turns out to be 4. Therefore, gcd(12, 20) = 4.</td>
</tr>
<tr>
<td>The Euclidean GCD algorithm is much more efficient than the brute force algorithm given above.</td>
<td> 
</td>
<td>The following table shows how gcd(2322, 654) is computed by the Euclidean algorithm. Notice that the number of steps required to calculate the gcd is significantly smaller than for the brute force algorithm.
<table>
<tr><th>a</th><th>b</th></tr>
<tr><td>2322</td><td>654</td></tr>
<tr><td>654</td><td>360</td></tr>
<tr><td>360</td><td>294</td></tr>
<tr><td>294</td><td>66</td></tr>
<tr><td>66</td><td>30</td></tr>
<tr><td>30</td><td>6</td></tr>
<tr><td>6</td><td>0</td></tr>
<tr><td>6</td><td>0</td></tr>
</table>
The search ends when b=0. The value of ‘a’ is the gcd. In this example, gcd(2322,654)=6.</td>
</tr>
</table>

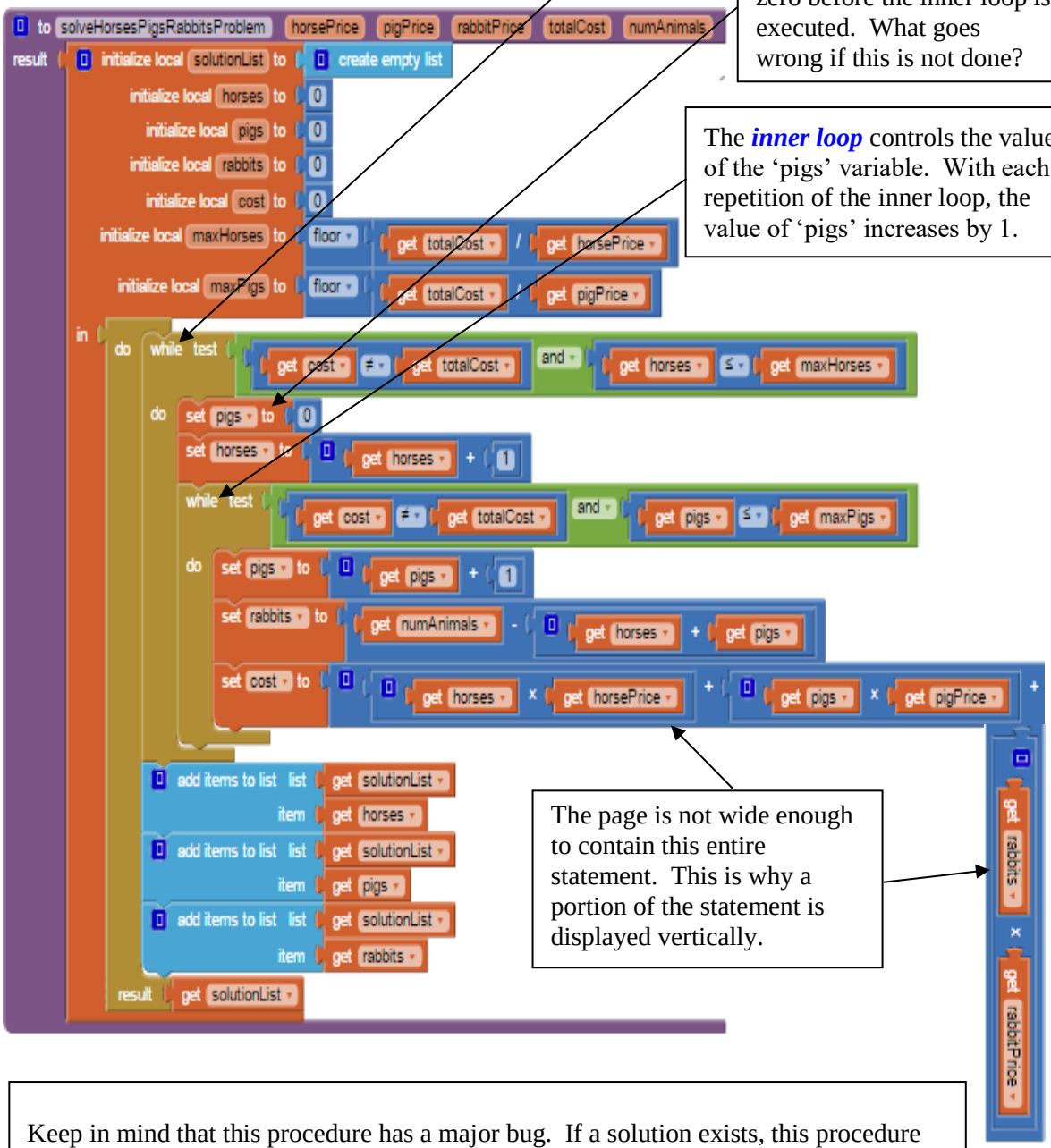
Problem	***App Inventor Solution***
(h) Horses cost $10, pigs cost $3 and rabbits cost only $0.50. A farmer buys 100 animals for $100. How many of each animal did he buy? Write a program to ***search*** for the solution to this problem. **(Exhaustive Search)**	

This solution involves a "while" loop contained within another "while" loop. When one loop is contained within another, we say that the loops are ***nested***.

The ***outer loop*** controls the value of the 'horses' variable. With each repetition of the outer loop, the value of 'horses' increases by 1.

The values of the 'pigs' variable must be reset to zero before the inner loop is executed. What goes wrong if this is not done?

The ***inner loop*** controls the value of the 'pigs' variable. With each repetition of the inner loop, the value of 'pigs' increases by 1.



The page is not wide enough to contain this entire statement. This is why a portion of the statement is displayed vertically.

Keep in mind that this procedure has a major bug. If a solution exists, this procedure will find it. However, if there is no solution, an erroneous result will be produced.

In class, you were asked to correct this bug.

First Repetition of Outer Loop Inner Loop Repeats 17 Times			
horses	pigs	rabbits	cost
1	1	98	$62.00
1	2	97	$64.50
1	3	96	$67.00
1	4	95	$69.50
1	5	94	$72.00
1	6	93	$74.50
1	7	92	$77.00
1	8	91	$79.50
1	9	90	$82.00
1	10	89	$84.50
1	11	88	$87.00
1	12	87	$89.50
1	13	86	$92.00
1	14	85	$94.50
1	15	84	$97.00
1	16	83	$99.50
1	17	82	$102.00

Second Repetition of Outer Loop Inner Loop Repeats 13 Times			
horses	pigs	rabbits	cost
2	1	97	$71.50
2	2	96	$74.00
2	3	95	$76.50
2	4	94	$79.00
2	5	93	$81.50
2	6	92	$84.00
2	7	91	$86.50
2	8	90	$89.00
2	9	89	$91.50
2	10	88	$94.00
2	11	87	$96.50
2	12	86	$99.00
2	13	85	$101.50

Third Repetition of Outer Loop Inner Loop Repeats 9 Times			
horses	pigs	rabbits	cost
3	1	96	$81.00
3	2	95	$83.50
3	3	94	$86.00
3	4	93	$88.50
3	5	92	$91.00
3	6	91	$93.50
3	7	90	$96.00
3	8	89	$98.50
3	9	88	$101.00

Fourth Repetition of Outer Loop Inner Loop Repeats 5 Times			
horses	pigs	rabbits	cost
4	1	95	$90.50
4	2	94	$93.00
4	3	93	$95.50
4	4	92	$98.00
4	5	91	$100.50

Fifth Repetition of Outer Loop Inner Loop Repeats 1 Time			
horses	pigs	rabbits	cost
5	1	94	$100.00

Questions

1. What is the purpose of the variable 'copyOfY' in the Euclidean GCD program? What would go wrong without this variable?

2. Explain how the brute force GCD program could be made more efficient. Would these gains of efficiency make a significant difference when computing the GCD of very large numbers?

3. What is a loop counter variable? Explain how to implement a loop counter in a "while" loop.

4. In the "Horses, Pigs, Rabbits" program, what will go wrong if the value of the variable 'pigs' is not reset to zero just before the inner loop is executed?

5. Write an App Inventor program that uses the Sieve of Eratosthenes algorithm to generate a list of all prime numbers less than 400.

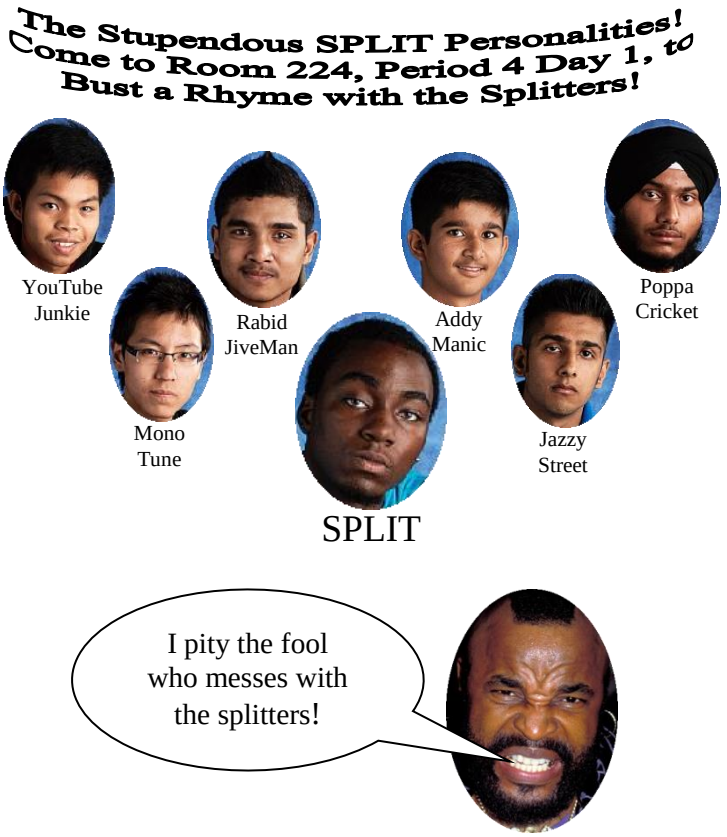
Please note! You will need to do some research to solve this problem! For starters, visit the following Web page:

http://www.hbmeyer.de/eratosiv.htm

App Inventor Review Problems #1

1. Give a step-by-step explanation of how the following could be accomplished:

A variation of the MoleMash game replaces the picture of the mole with pictures of members of the *Split Personalities*. Each time a picture of one of the splitters is tapped, the member's favourite rap line is heard over the speakers and displayed in a label. Otherwise, if the user taps the screen without hitting any of the moving images, Mr. T's picture appears, and the line "I pity the fool" is played over the speakers.



2. Create an App Inventor program that calculates the sum shown below, where the values of *x*, *d*, *k* and *n* are chosen by the user.

$$x^k + (x+d)^k + (x+2d)^k + \cdots + (x+(n-1)d)^k = \sum_{i=1}^{n} (x+(i-1)d)$$

For example, if the user chooses $x = 3$, $d = 2$, $k = 4$ and $n = 10$, then the program would compute the sum $3^4 + 5^4 + 7^4 + 9^4 + \cdots + 21^4$. You should create a procedure with a result that is devoted to calculating the sum.

3. Create an App Inventor program just like the one in question 2 except that the user specifies the maximum (or minimum) sum instead of specifying the number of terms (i.e. instead of *n*).

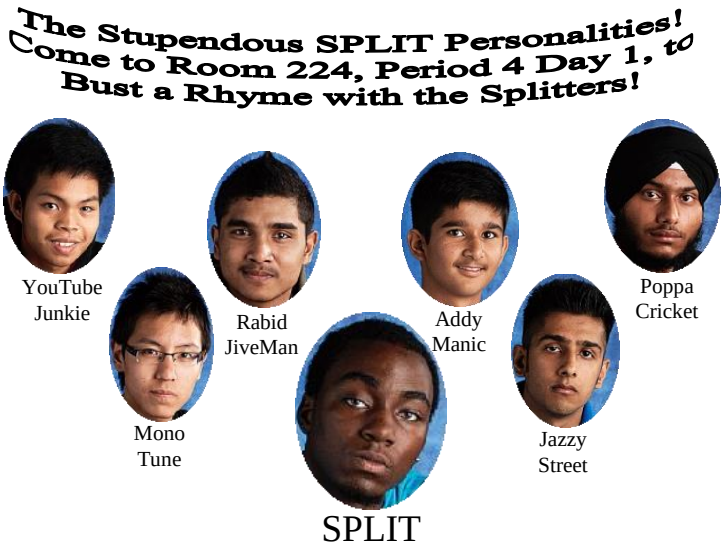
4. Create an App Inventor program that can add, subtract multiply or divide any two fractions.

App Inventor Review Problems #2

1. Give a step-by-step explanation of how the following could be accomplished:

A variation of the MoleMash game replaces the picture of the mole with pictures of members of the *Split Personalities*.

- Each time a picture of one of the splitters is tapped, the picture disappears but reappears exactly one minute later.
- If the user is fast enough to make all the pictures disappear before any of them reappear, the game ends and the user wins.
- If any of the pictures are still in motion after five minutes of play, the game ends and the user loses the game. In this case, Mr. T's picture appears and the line, "I told you not to mess with the splitters!" is played over the speakers.

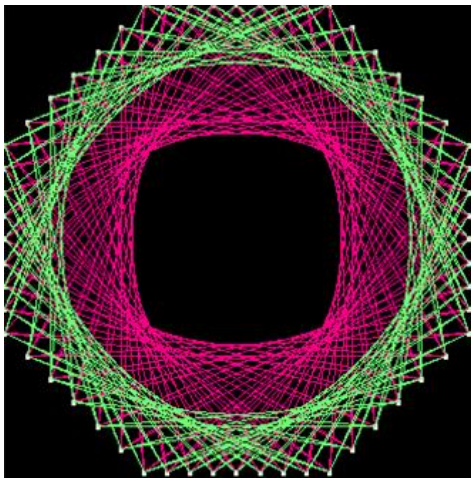
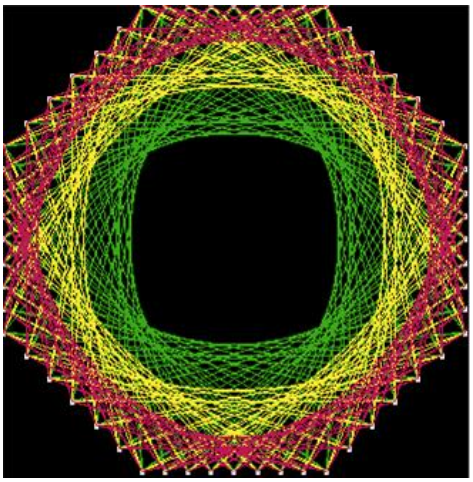
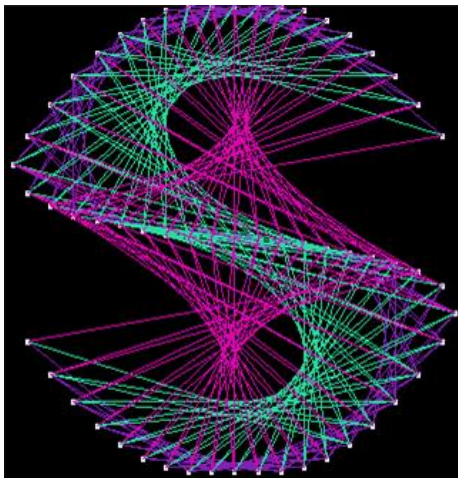


2. Create an App Inventor procedure with a result that takes a percentage mark as input and returns the equivalent grade point score. (See page 32 for details.)

3. Create an App Inventor program that factors quadratic expressions. The app takes as input the coefficients *a*, *b* and *c* of the quadratic in standard form (i.e. $ax^2 + bx + c$) and then displays the factored form of the quadratic.

App Inventor Review Problems #3

Write an App Inventor program that can produce string art. Examples of string art are shown below:

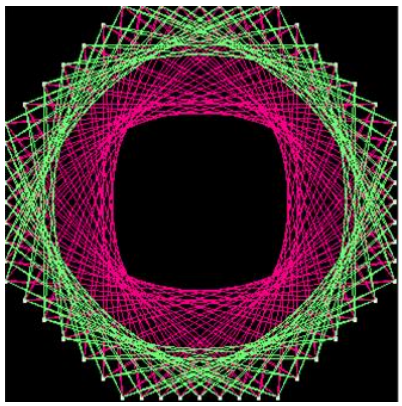
The following is a ***pseudocode*** description of an algorithm for producing string art:

```
Initialize the values of A and B
Set A=1
Set B=some value between 1 and N
loop
   join point A to point B
   add 1 to A
   join point B to point A
   add 1 to B
   if B > N
      set B=1
while A < N
```

Pseudocode
Statements outlining the operation of a computer program, written in something similar to computer language but in a more understandable format.

"Points" Referred to in Pseudocode

- The points are equally spaced along the perimeter of a shape such as an octagon.
- The points are numbered 1, 2, 3, …, N where N represents the total number of points



In this picture, there are 64 equally spaced points along the perimeter of an octagon. The picture is formed by joining points to other points.

Note

- The prefix "pseudo" means "false."
- Other words beginning with this prefix:
  pseudonym, pseudoscience, pseudohistorical
- The co-ordinates of the points spaced uniformly around the perimeter of the polygon (e.g. octagon) can be generated mathematically. Once generated, the co-ordinates can be stored in lists.