

UNIT 0 – INTRODUCTION TO PROGRAMMING THROUGH APP INVENTOR

UNIT 0 – INTRODUCTION TO PROGRAMMING THROUGH APP INVENTOR.....	1
LEARNING THE ESSENTIAL FEATURES OF APP INVENTOR	3
BRIEF DESCRIPTION AND HISTORY OF APP INVENTOR FOR ANDROID	3
OVERVIEW OF APP INVENTOR.....	3
How to Access App Inventor.....	3
How to Work with App Inventor.....	3
The MyProjects Page	4
The Design Page	4
The Blocks Editor.....	4
PAINTPOT: CREATING YOUR FIRST APP.....	5
STEP 1: CREATING THE PAINTPOT APP	5
STEP 2: USING THE COOK/CHEF ANALOGY TO UNDERSTAND THE LOGIC OF THE PAINTPOT APP.....	5
EXERCISE.....	7
CREATING MORE APPS – MORE EXAMPLES	7
WARNING!	7
CREATING MORE APPS – MAKING YOUR OWN!.....	8
INTRODUCTION	8
METHOD 1 – IMPROVE AN EXISTING APP: MOLEMASH EXTREME VERSION	8
METHOD 2 – CREATE YOUR OWN APPS!.....	8
APP INVENTOR MAIN IDEAS – REVIEW #1.....	9
APP INVENTOR MAIN IDEAS – REVIEW #2.....	10
APP INVENTOR MAIN IDEAS – REVIEW #3.....	11
USING THE “FOLLOW-THE-MOLE MASH” GAME TO APPRECIATE THE POWER OF VARIABLES.....	13
INTRODUCTION TO LOOPS: LINE DRAWING PROBLEMS.....	14
INTRODUCTION	14
PROBLEM.....	14
HINTS.....	14
EXPLANATION.....	14
MORE PROBLEMS.....	15
GENERATING THE PICTURES IN APP INVENTOR.....	16
EXPLANATION OF THE MORE EFFICIENT METHOD.....	16
TYPES OF LOOPS	16
Counted Loops (“For Loops”).....	16
Conditional Loops (“While Loops”).....	16
CIRCLE DRAWING PROBLEMS	17
PROBLEM WITH CURRENT VERSION OF APP INVENTOR	17
THE DEFINITION OF THE “DRAWCIRCLEPOINTBYPOINT” PROCEDURE	17
AN EXAMPLE OF A CALL TO THE “DRAWCIRCLEPOINTBYPOINT” PROCEDURE	17
CIRCLE DRAWINGS	18
MORE LINE/CIRCLE DRAWING PRACTICE.....	19
ANALYZING THE MAKEQUIZ APP FROM CHAPTER 10.....	20
EXAMPLE	22
PROGRAMMING PROBLEMS WHOSE SOLUTIONS REQUIRE THE USE OF COUNTED (“FOR”) OR CONDITIONAL (“WHILE”) LOOPS	23
Algorithm.....	23
EUCLID AND THE GCD.....	25
DEFINITION OF GCD	25
Examples	25

BRUTE FORCE (EXHAUSTIVE SEARCH) ALGORITHM FOR COMPUTING THE GCD OF TWO INTEGERS	25
APP INVENTOR CODE FOR SLOW GCD ALGORITHM	26
<i>Questions</i>	27
DESCRIPTION OF EUCLID’S (FAST) METHOD FOR COMPUTING THE GCD OF TWO INTEGERS.....	27
EXAMPLE	27
YOUR TASK	27
SOLUTIONS TO SELECTED PROBLEMS REQUIRING LOOPS	28
QUESTIONS	31
APP INVENTOR REVIEW PROBLEMS #1	33
APP INVENTOR REVIEW PROBLEMS #2	34
APP INVENTOR REVIEW PROBLEMS #3	35
NOTE	35

LEARNING THE ESSENTIAL FEATURES OF APP INVENTOR

Brief Description and History of App Inventor for Android

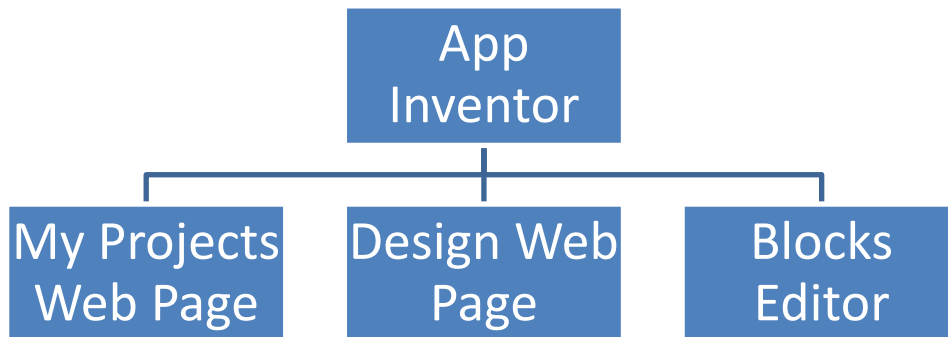
- Allows anyone to create software applications (“apps”) for the Android Operating System
- The Android Operating System is used on several different mobile devices including models made by Samsung, HTC, LG, Motorola, Sony, Alcatel, Archos, Kyocera, Dell, Xperia, Excite, Asus, Sanyo, Acer and others
- 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”

Overview of App Inventor

How to Access App Inventor

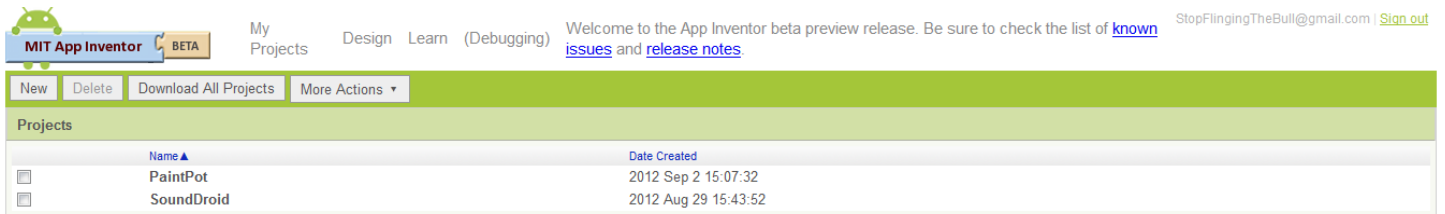
- Requires a Google account
- If you already have a Google account, simply use it to log on to App Inventor
- If you do not have a Google account, create one at <https://accounts.google.com/NewAccount>
- Once you have a Google account, log on to App Inventor at <http://appinventor.mit.edu/>
- In addition to having a Google account, the following must also be installed on your computer:
 - Java 6 or higher (see <http://www.java.com>)
 - The App Inventor Setup Package (see <http://beta.appinventor.mit.edu/learn/setup/>)

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, Download Project to Local Computer, etc
- **Design Web Page**
Tools for Designing the User Interface
Palette, Viewer, Components List, Properties List
- **Blocks Editor**
Java Program that runs in its own Window (i.e. does not run in a Web browser)
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

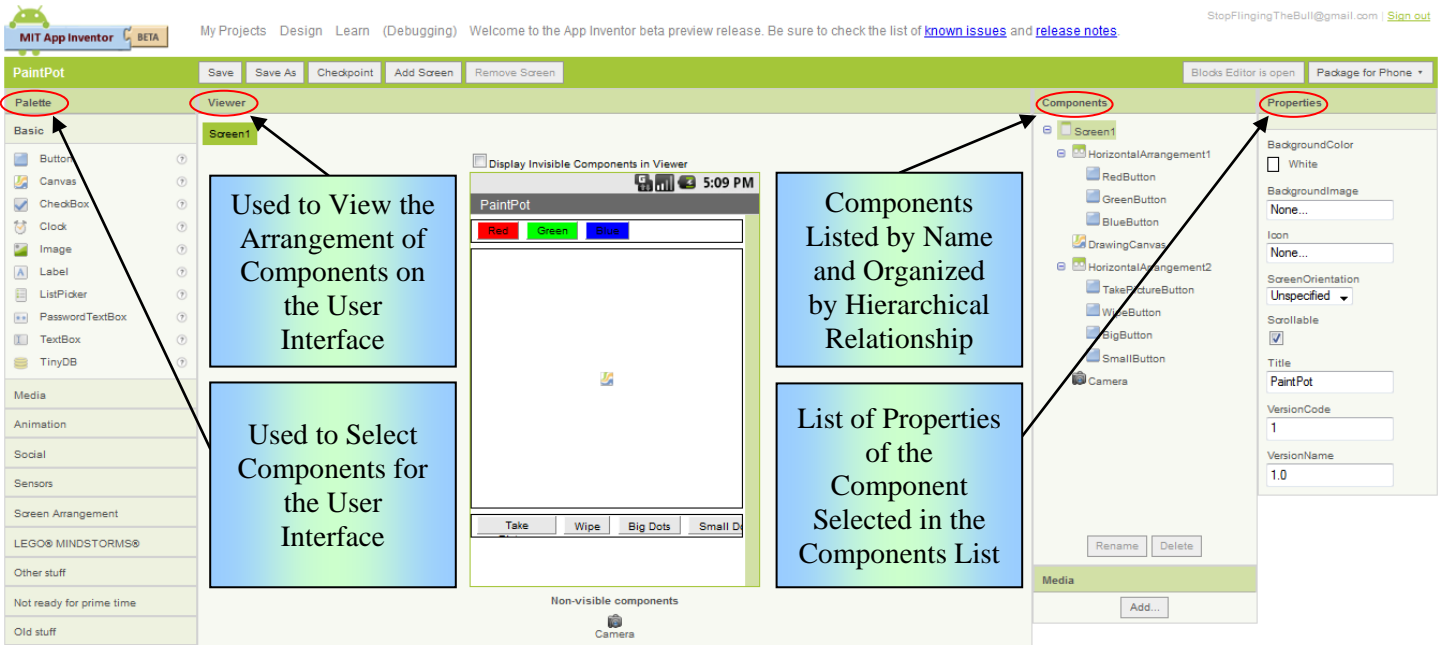
The MyProjects Page



The MyProjects Page shows the MIT App Inventor interface. At the top, there's a header with the MIT App Inventor logo, a 'BETA' badge, and navigation links: 'My Projects', 'Design', 'Learn', and '(Debugging)'. A welcome message states: 'Welcome to the App Inventor beta preview release. Be sure to check the list of [known issues](#) and [release notes](#).' A link to 'StopFlingingTheBull@gmail.com' and a 'Sign out' link are also present. Below the header, there's a toolbar with 'New', 'Delete', 'Download All Projects', and 'More Actions'. The main area is titled 'Projects' and contains a table with two columns: 'Name' and 'Date Created'. The table lists two projects: 'PaintPot' (created on 2012 Sep 2 15:07:32) and 'SoundDroid' (created on 2012 Aug 29 15:43:52).

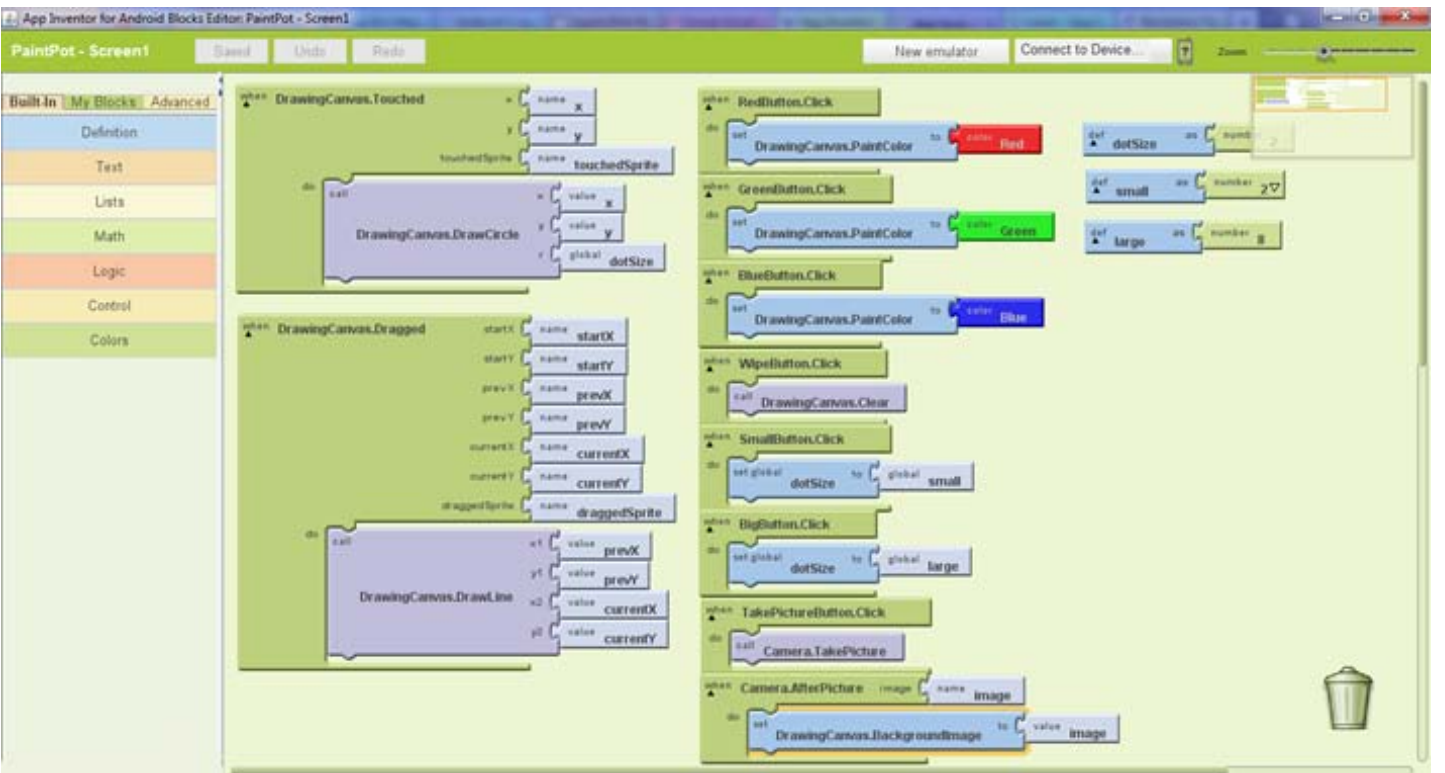
	Name ▲	Date Created
	PaintPot	2012 Sep 2 15:07:32
	SoundDroid	2012 Aug 29 15:43:52

The Design Page



The Design Page shows the MIT App Inventor interface for the 'PaintPot' project. The top header is identical to the MyProjects page. Below the header, there's a toolbar with 'Save', 'Save As', 'Checkpoint', 'Add Screen', and 'Remove Screen'. The main area is divided into four panels: 'Palette', 'Viewer', 'Components', and 'Properties'. The 'Palette' panel on the left lists various components categorized by 'Basic', 'Media', 'Animation', 'Social', 'Sensors', 'Screen Arrangement', 'LEGO MINDSTORMS', 'Other stuff', and 'Old stuff'. The 'Viewer' panel in the center shows a preview of the 'Screen1' with a canvas and buttons. The 'Components' panel on the right lists the components currently on the screen, organized hierarchically. The 'Properties' panel on the far right shows the properties for the selected component, 'PaintPot'. Annotations with arrows point to each panel: 'Used to View the Arrangement of Components on the User Interface' points to the Viewer; 'Used to Select Components for the User Interface' points to the Palette; 'Components Listed by Name and Organized by Hierarchical Relationship' points to the Components panel; and 'List of Properties of the Component Selected in the Components List' points to the Properties panel.

The Blocks Editor



The Blocks Editor shows the MIT App Inventor interface for the 'PaintPot' project. The top header is identical to the previous pages. Below the header, there's a toolbar with 'New emulator', 'Connect to Device...', and a 'Zoom' slider. The main area is divided into two panels: 'Built-In' and 'My Blocks'. The 'Built-In' panel on the left lists various blocks categorized by 'Definition', 'Text', 'Lists', 'Math', 'Logic', 'Control', and 'Colors'. The 'My Blocks' panel on the right shows the blocks created by the user. The central workspace contains several block diagrams. The first diagram is a 'when DrawingCanvas.Touched' block with a 'call DrawingCanvas.DrawCircle' block. The second diagram is a 'when DrawingCanvas.Dragged' block with a 'call DrawingCanvas.DrawLine' block. The third diagram is a 'when RedButton.Click' block with a 'set DrawingCanvas.PaintColor to color Red' block. The fourth diagram is a 'when GreenButton.Click' block with a 'set DrawingCanvas.PaintColor to color Green' block. The fifth diagram is a 'when BlueButton.Click' block with a 'set DrawingCanvas.PaintColor to color Blue' block. The sixth diagram is a 'when WipeButton.Click' block with a 'call DrawingCanvas.Clear' block. The seventh diagram is a 'when SmallButton.Click' block with a 'set global dotSize to global small' block. The eighth diagram is a 'when BigButton.Click' block with a 'set global dotSize to global large' block. The ninth diagram is a 'when TakePictureButton.Click' block with a 'call Camera.TakePicture' block. The tenth diagram is a 'when Camera.AfterPicture' block with a 'set DrawingCanvas.BackgroundImage to value image' block. A trash can icon is located in the bottom right corner.

PAINTPOT: CREATING YOUR FIRST APP

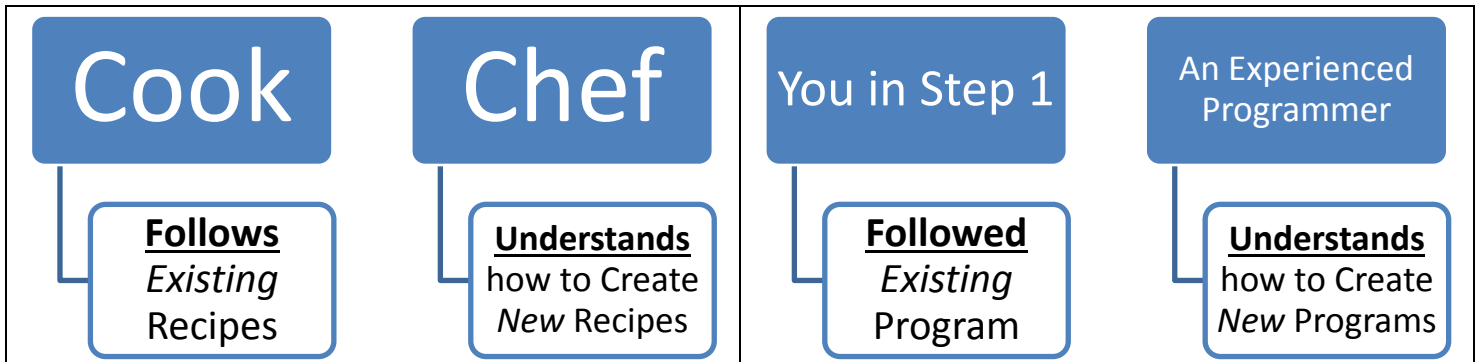
Step 1: Creating the PaintPot App

This part is easy! All you need to do is follow the instructions in the following document:

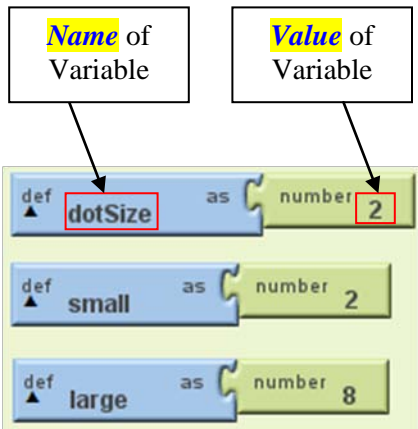
I:\Out\Nolfi\Ics3u0\ch2PaintPot.pdf


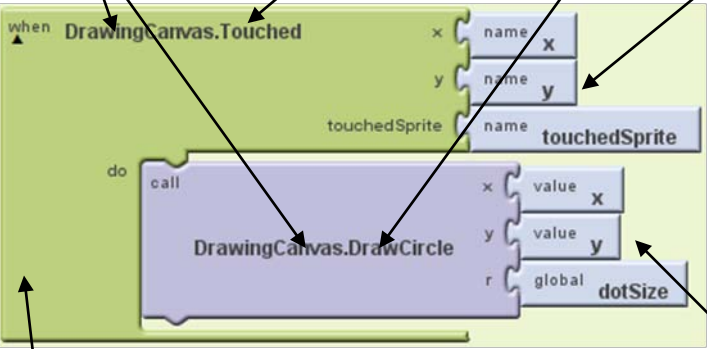
If you follow the instructions very carefully, the app should function correctly. In the event that it does not work as expected, check your blocks carefully to ensure that they are exactly as shown in the above document.

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



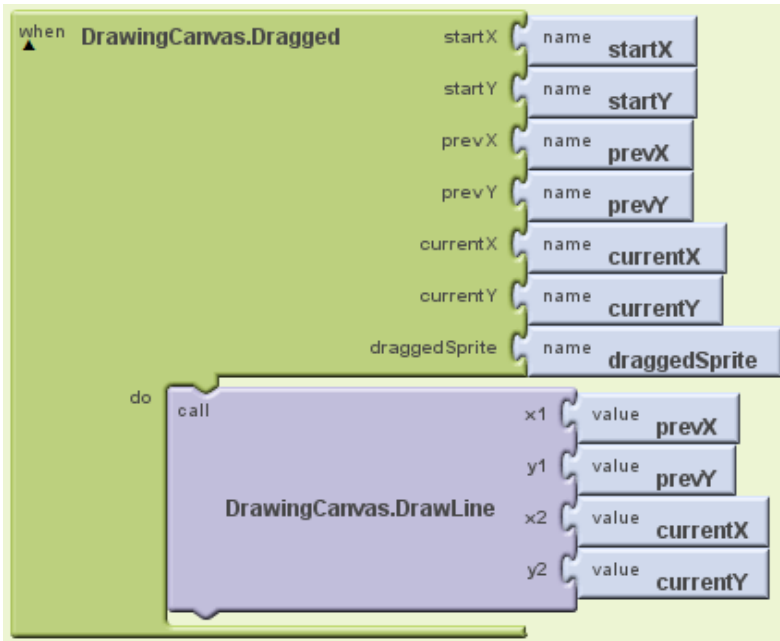
As can easily be appreciated from the above analogy, it is not enough merely to *follow* existing programs. All programmers must also be able to develop new software from scratch. To accomplish this, it is obviously very important to *understand* programming concepts. A detailed description of the programming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

Picture	Programming Concepts
	<p>Variable</p> <ul style="list-style-type: none">• A variable is a <u>name</u> that is used to <u>represent</u> a value that is stored in a computer's main memory (i.e. in the RAM).• Variables are used whenever information needs to be "<u>remembered</u>" (i.e. "<u>memorized</u>") 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. Notice the names "dotSize," "small" and "large." These names are far more meaningful than "d," "s" and "l."

Picture	Programming Concepts
<div data-bbox="102 163 821 741"> <div> Name of Component (aka Object) </div> <div> Name of the Event that causes execution of procedure block </div> <div> Name of a Property of a Component </div>  <div> Procedure. The instructions within the block are executed when the “Click” event occurs on “RedButton.” </div> <div> Value of the “PaintColor” Property </div> </div>	<div data-bbox="849 163 1511 783"> Procedure <ul style="list-style-type: none"> 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 <ul style="list-style-type: none"> 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.” </div>
<div data-bbox="102 821 821 1587"> <div> Name of Component (aka Object) </div> <div> Name of the Event that causes execution of procedure block </div> <div> Name of a Method </div>  <div> Procedure Block with Parameters (aka Arguments). The instructions within the block are executed when the “Touched” event occurs on “DrawingCanvas.” The arguments of this procedure block are the variables <i>x</i>, <i>y</i> and <i>touchedSprite</i>. </div> </div>	<div data-bbox="821 821 1221 1587"> <p>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 <i>x</i> and <i>y</i> store the co-ordinates of the point that is touched on “DrawingCanvas.” The parameter “touchedSprite” is used for animations.</p> <p>The Arguments passed to the “DrawCircle” method. The values of <i>x</i> and <i>y</i> come from the parameters <i>x</i> and <i>y</i> of the procedure block “DrawingCanvas.Touched.” The radius of the circle comes from the value of “dotSize.”</p> </div> <div data-bbox="1252 821 1528 1692"> Property <ul style="list-style-type: none"> Every component has Properties, which store information on <u>characteristics</u> of the component. Examples of properties include “Enabled,” “Height,” “Text” and “Width.” Method <ul style="list-style-type: none"> Every component has Methods, which are <u>actions</u> that are associated with the component. Examples of methods include “Clear,” “DrawCircle,” “DrawPoint” and “DrawLine.” </div>

Exercise

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



1. “DrawingCanvas” is the name of a _____.
2. “Dragged” is the name of a _____.
3. “DrawingCanvas.Dragged” is the name of a _____.
4. “DrawLine” is the name of a _____.
5. “startX” is the name of a _____. Its purpose is _____.

CREATING MORE APPS – MORE EXAMPLES

For step-by-step instructions on how to create more apps, navigate to the following folder:

I:\4Students\OUT\Nolfi\ICS3U0\00-AppInventor

You will also find all the resources (e.g. pictures, sounds, etc) that you need in the following folder:

I:\4Students\OUT\Nolfi\ICS3U0\00-AppInventor\App Inventor Example 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

By now you should have completed the “MoleMash” app.

(See I:\4Students\OUT\Nolfi\ICS3U0\00-AppInventor\ch3MoleMash.pdf or <http://www.cs.usfca.edu/~wolber/appinventor/bookSplits/ch3MoleMash.pdf>).

Add the following features to the MoleMash app:

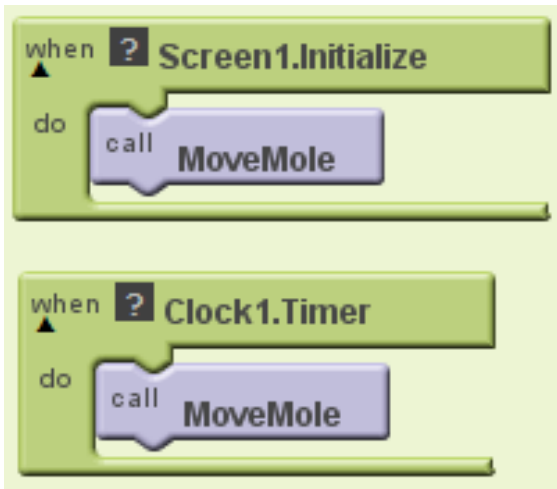
1. Levels of Difficulty: “Easy,” “Medium,” “Difficult”
(e.g. the game can be made more challenging by increasing mole speed, decreasing size of the mole picture, etc)
2. A Pleasant Sound is Played when the Mole is Hit
3. The Mole Picture Changes Briefly when the Mole is Hit
4. A Rude Sound is Played when the Mole is Missed
5. The game ends after a certain number of hits and misses, after which the player is either declared 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!

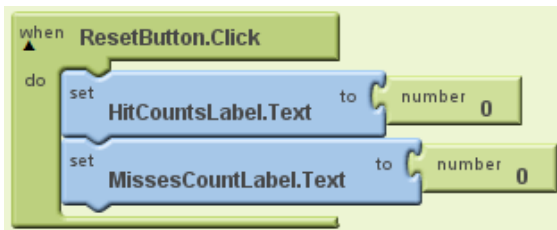
There is no better way to learn about programming than to create your own apps! You are strongly encouraged to unleash your imagination and explore whatever ideas come to mind!

APP INVENTOR MAIN IDEAS – REVIEW #1

1. Identify and Explain Purpose

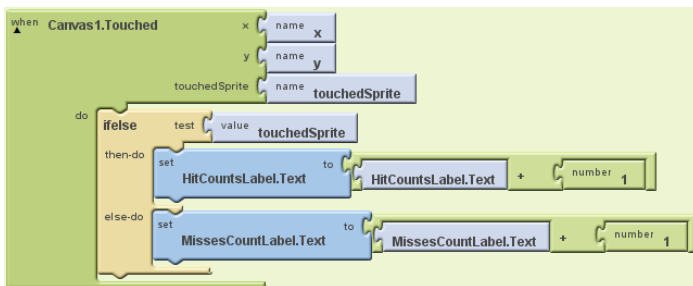


- (a) List all the *procedure names* in the blocks shown at the left.
- (b) List all the *component names* in the blocks shown at the left.
- (c) List all the *event names* in the blocks shown at the left.
- (d) List all the *property names* in the blocks shown at the left.
- (e) Explain the purpose of “Screen1.Initialize.”



- (f) Explain the purpose of “Clock1.Timer.”
- (g) Explain the purpose of “call MoveMole.”
- (h) Explain the purpose of “ResetButton.Click.”
- (i) Explain the purpose of “set HitCountsLabel.Text to 0.”

2. Explain Purpose



- (a) What are “x,” “y” and “touchedSprite?” What is their purpose?
- (b) Explain the purpose of the “if else” block.
- (c) What is the purpose of “set HitCountsLabel.Text to HitCountsLabel.Text + 1?”

3. Explain Concept

In the MoleMash game, the mole picture moves about the canvas in a random fashion. Explain how this is accomplished.

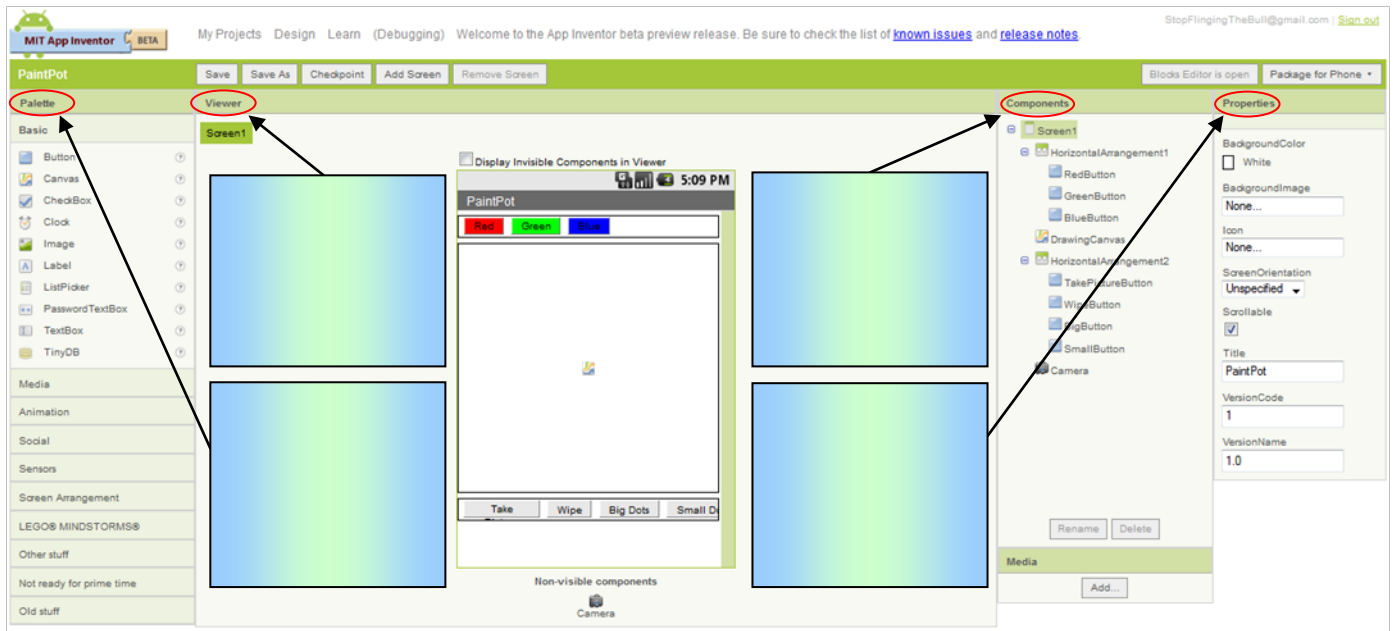
APP INVENTOR MAIN IDEAS – REVIEW #2

Explain each of the following:

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

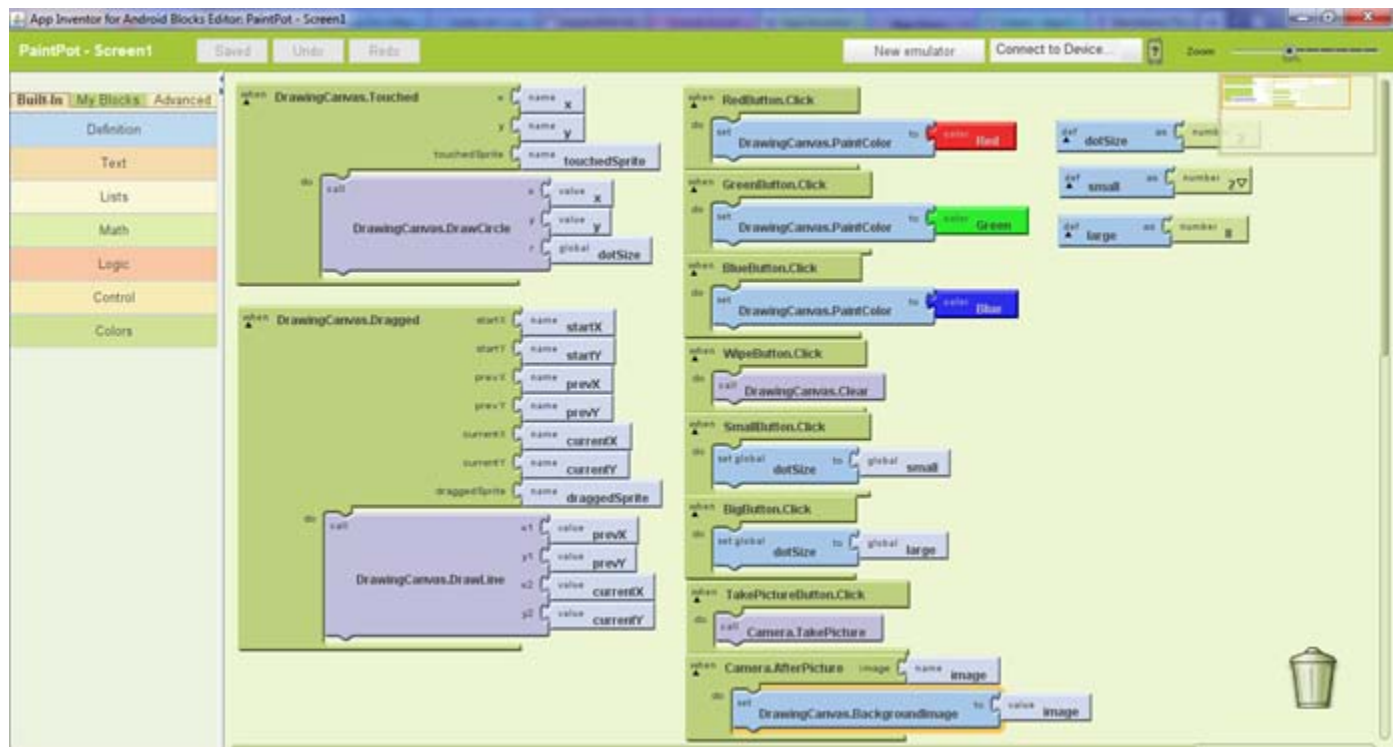
APP INVENTOR MAIN IDEAS – REVIEW #3

1. The purpose of the Design Page shown below is _____



Use the provided text boxes to state the purpose of each of the four main parts of the Design Page.

2. The purpose of the Blocks Editor Java program shown below is _____



3. The purpose of the emulator window shown at the right is _____

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

(a) In the MoleMash app, the mole picture changes briefly when the mole is hit.



(b) In the PaintPot app, straight lines can be drawn as well as curves.

(c) In the MoleMash app, a “bonus image” is occasionally displayed for a brief time. The player receives bonus points for tapping the bonus image.

(d) In the MoleMash app, 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.

USING THE “FOLLOW-THE-MOLE MASH” GAME TO APPRECIATE THE POWER OF VARIABLES

“Follow-the-Mole Mash” is a simple variation of the MoleMash game. As in the original game, the mole’s position changes randomly at regular intervals. Unlike the original, a second “penalty” sprite follows the mole closely, sometimes leaving very little of the mole exposed. If the penalty sprite is touched instead of the mole, the “misses” count increases by two to penalize the player.

These are called *comments*. Their purpose is to help *people understand* the program. Comments are ignored by the computer.

The variable 'moleX' stores the x-co-ordinate of the top-left corner of the mole sprite.

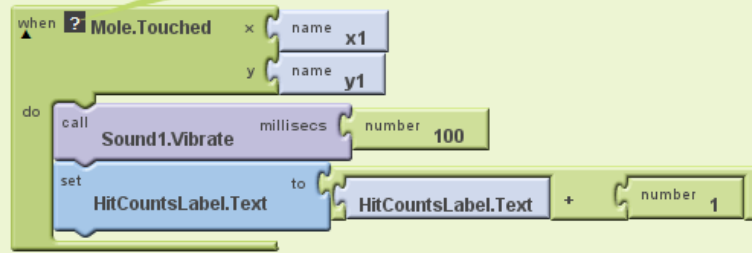
The variable 'moleY' stores the y-co-ordinate of the top-left corner of the mole sprite.

def ? moleX as number 0

def ? moleY as number 0

- To make it possible for the penalty image to “follow” the mole, the mole’s co-ordinates must be known.
- To accomplish this, *variables* are used to store (i.e. “remember”) the mole’s co-ordinates.
- In general, *variables* are used whenever data need to be *saved for later use*.

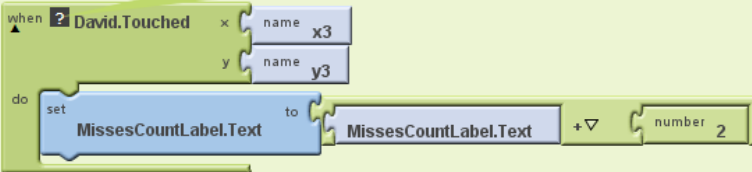
The 'touchedSprite' argument of the 'Canvas1.Touched' event handler has a value of 'true' if ANY sprite on the canvas is touched. Unfortunately, this does not allow us to determine WHICH sprite was touched. Thus, the canvas, the mole and the penalty sprite must each have its own event handler for the 'Touched' event.



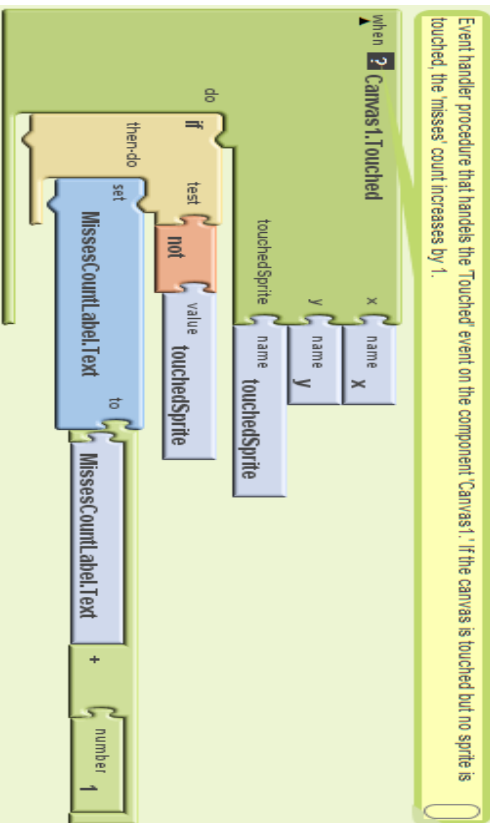
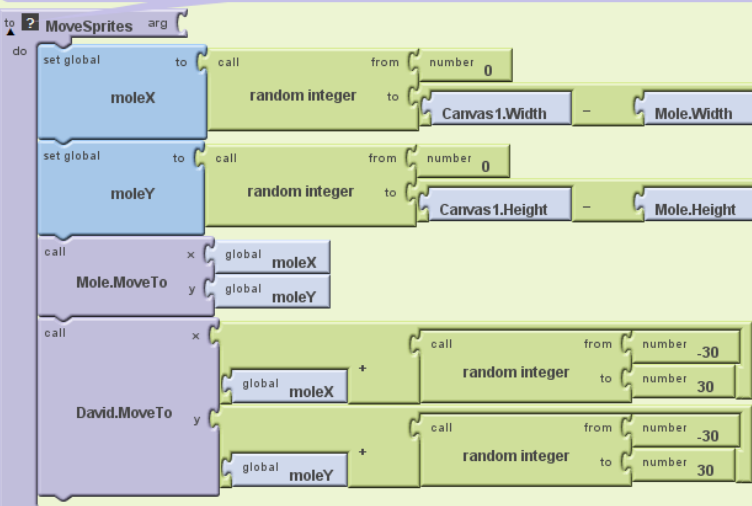
Move the sprites every time the component 'Clock1' fires the 'Timer' event. The 'Timer' event is fired at regular intervals according to the value of the clock's 'TimerInterval' property.



If the penalty sprite is touched, the misses count increases by 2.



The MoveSprites procedure is NOT an event handler procedure because its execution is not directly triggered by an event. This kind of procedure is executed only if it is CALLED. We shall refer to such procedures as “general procedures.”



INTRODUCTION TO LOOPS: LINE DRAWING PROBLEMS

Introduction

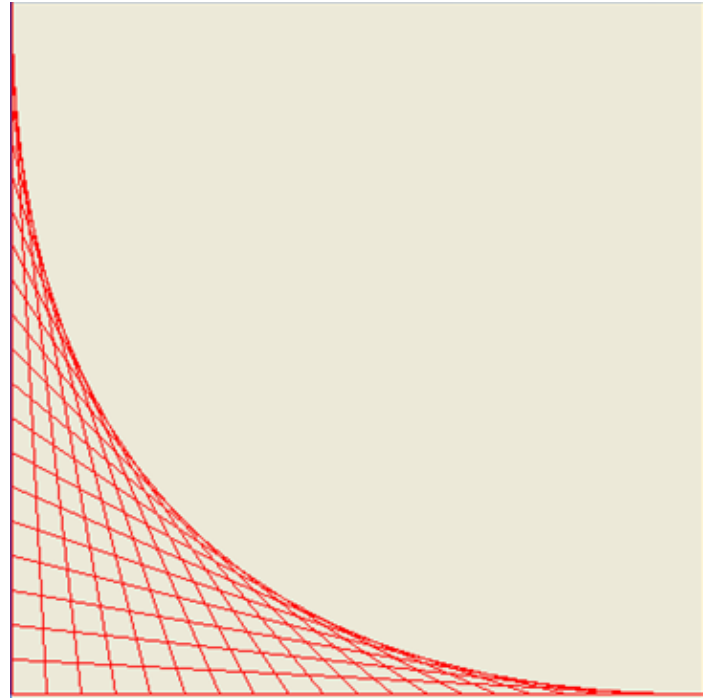
Although we most definitely *perceive* a curve in the picture at the right, the picture itself was created by drawing a series of *straight lines*. No curves were actually drawn! Why then do we seem to see a curve? The answer to this question has everything to do with how our brains *construct* the “reality” 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, in a sense, to create the perception of a curve.

Problem

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

Hints

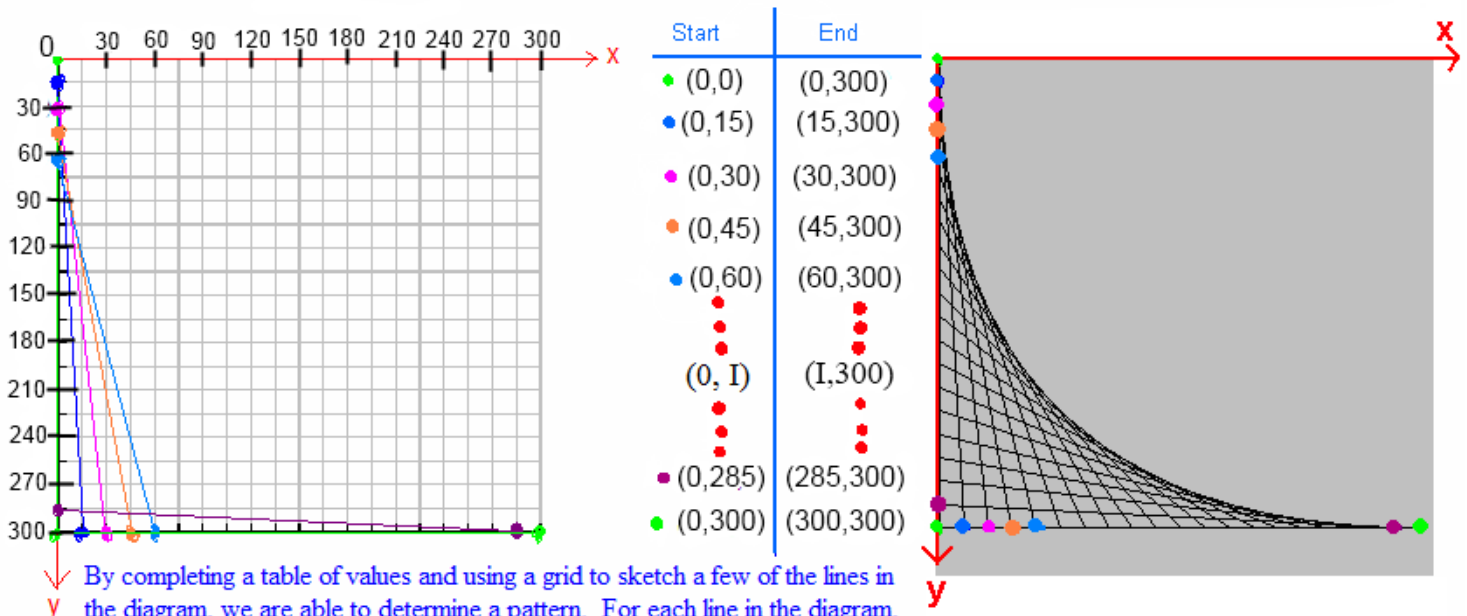
1. Use a “Canvas” component.
2. In addition, your app will need to use the “DrawLine” method of a “Canvas” component.
3. You need to understand the co-ordinate system that is used for “Canvas” components.
4. There is a definite pattern that governs how the lines are drawn. Before attempting to create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

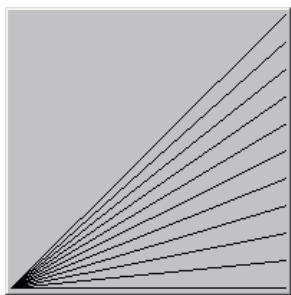


By completing a table of values and using a grid to sketch a few of the lines in the diagram, we are able to determine a pattern. For each line in the diagram, the y-co-ordinate of the “start” point is equal to the x-co-ordinate of the “end” point. In addition, we observe that the x-co-ordinate of the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

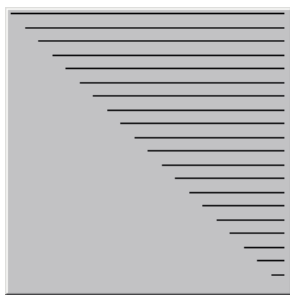
More Problems

Use graph paper and a table of values to determine the pattern that is used to create each picture. Then create an App Inventor app that displays each picture successively at regular intervals. For an extra challenge, create the app in such a way that the picture that is displayed at any given time is selected randomly.

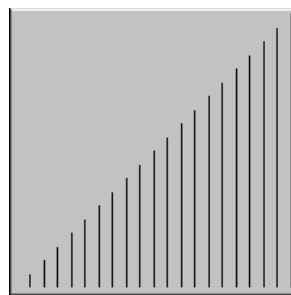
1.



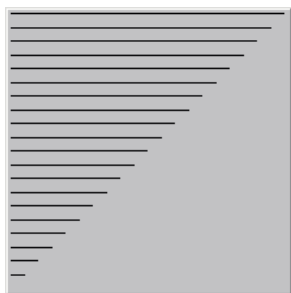
2.



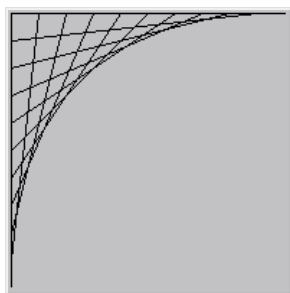
3.



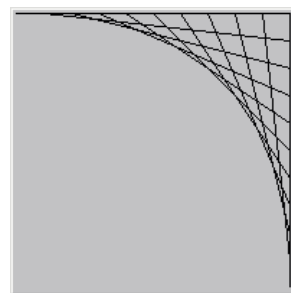
4.



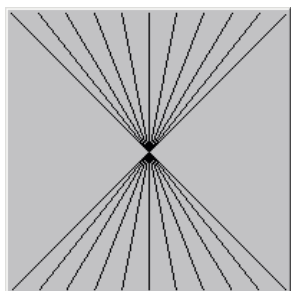
5.



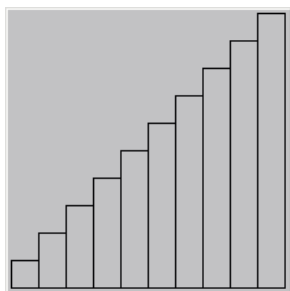
6.



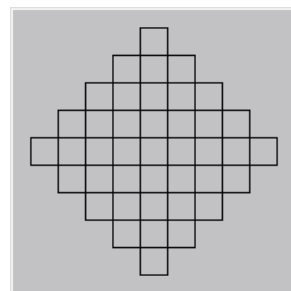
7.



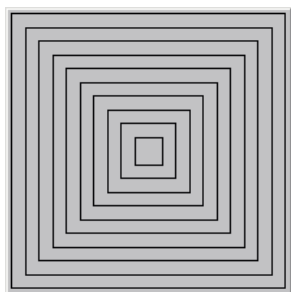
8.



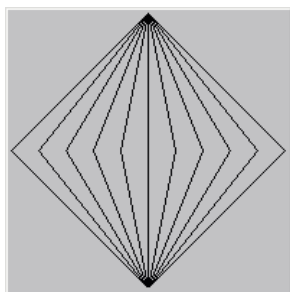
9.



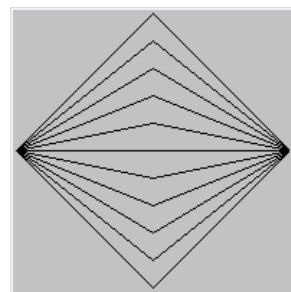
10.



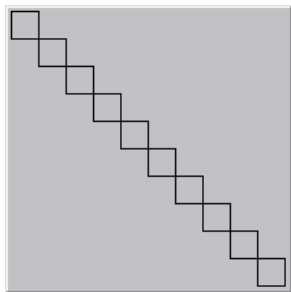
11.



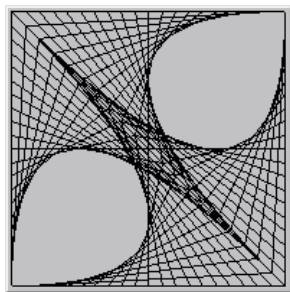
12.



13.

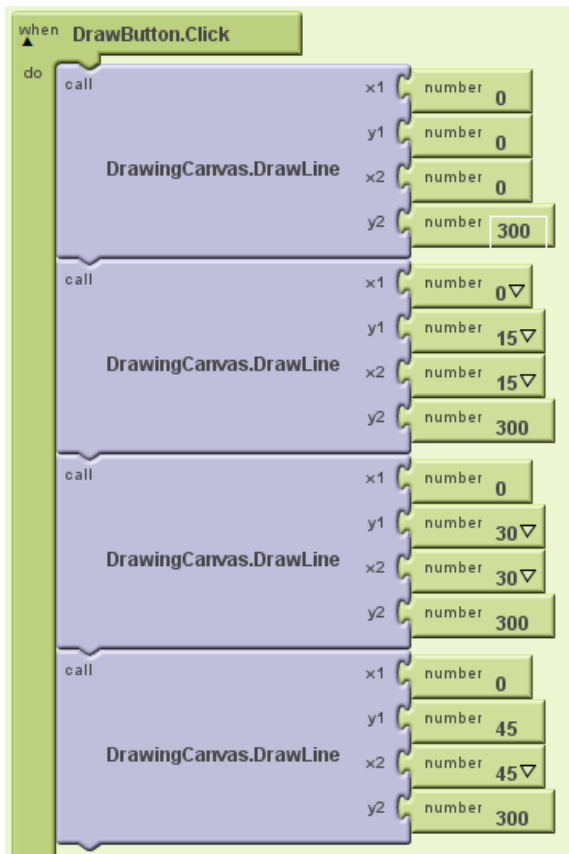


14.



Generating the Pictures in App Inventor

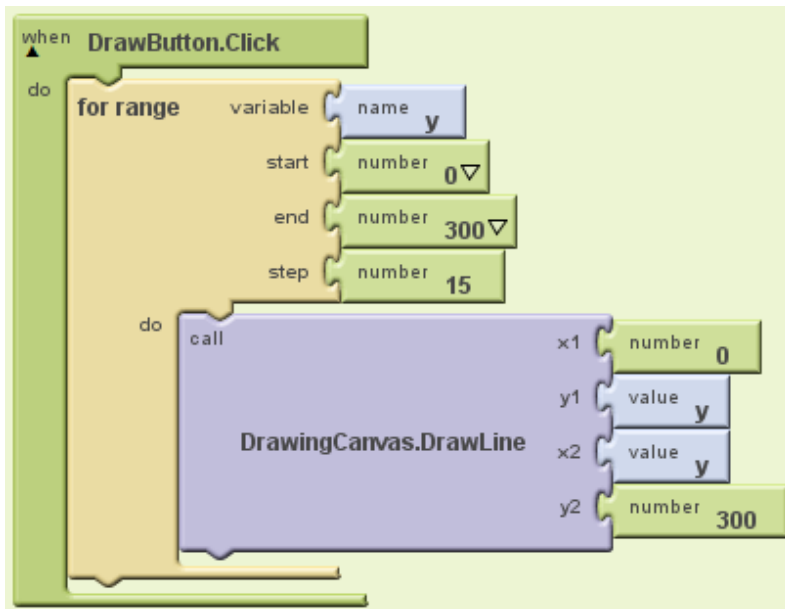
Method 1- Easy to Understand but Tedious



-
-
-

The remaining *seventeen* 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 “for range” loop shown above:

- The call to the “DrawLine” method is shown only *once* BUT it is *repeated* exactly twenty-one times by the “for” loop.
- 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 “step.” In the above example, the value of “y” increases by 15 after each repetition because the value of “step” is 15.
- The value of “y” *ranges* from 0 to 300 because the “start” and “end” values are set respectively to 0 and 300. This explains the name of the block in App Inventor (i.e. “for range”).
- Thus “y” takes on the values 0, 15, 30, 45, ..., 270, 285, 300, after which the loop terminates (i.e. stops repeating).

Types of Loops

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 *three times*.

Conditional Loops (“While Loops”)

These are used when the number of repetitions is *not* known at *design-time* and *cannot* be calculated at *run-time*. Whether looping continues or terminates is based on whether a certain *condition* is true or false. The number of repetitions of such loops is generally *not predictable*.

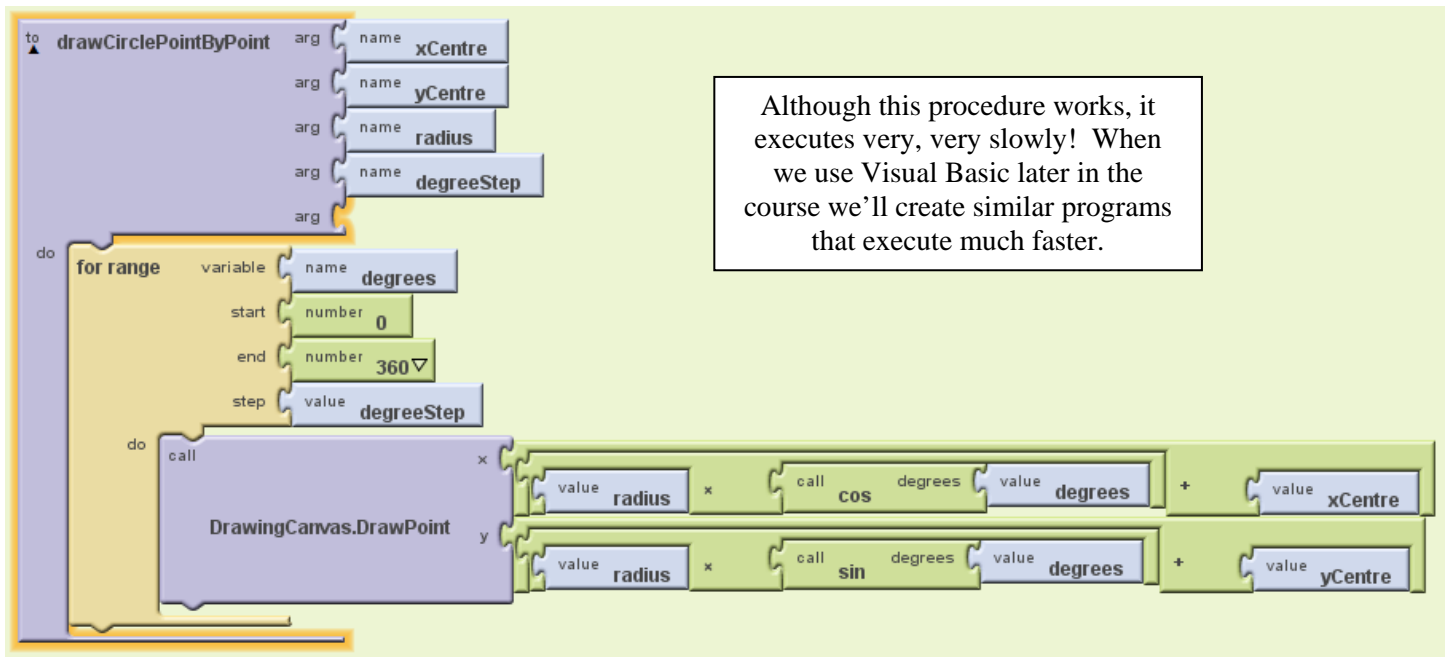
Analogy: Keep stirring the coffee until the sugar dissolves. Repeat the act of stirring once until the sugar dissolves.

CIRCLE DRAWING PROBLEMS

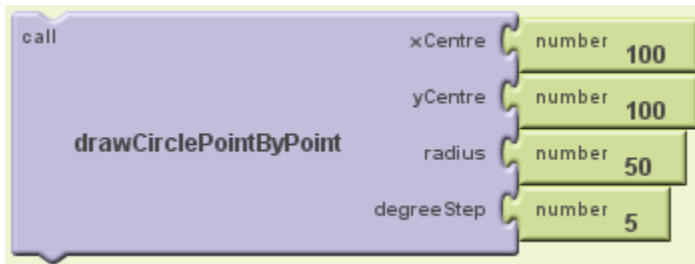
Problem with Current Version of App Inventor

The current version of App Inventor only provides a procedure for drawing *filled* circles. To draw only the outline of a circle without filling its interior, we are forced to *create our own procedure*. The procedure described below can draw unfilled circles but it does so at an excruciatingly slow speed. Nonetheless, it is better than nothing!

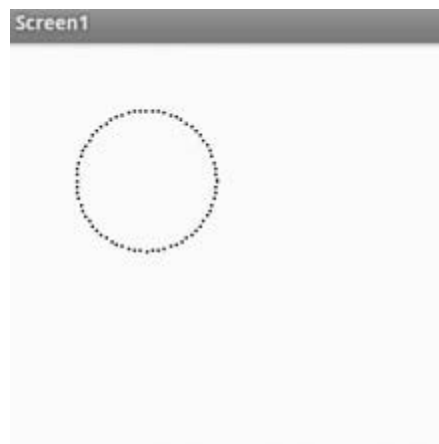
The Definition of the “drawCirclePointByPoint” Procedure



An Example of a Call to the “drawCirclePointByPoint” Procedure



The following is the circle produced by this call. Reduce the value of “degreeStep” to decrease the “gaps” in the circle.

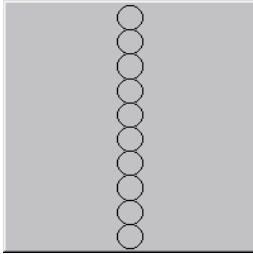


Circle Drawings

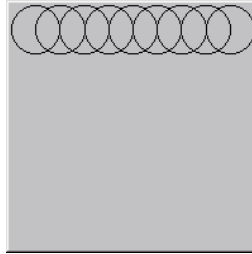
Use App Inventor to create each of the pictures shown below. Keep in mind the following important points:

- Use a table of values to determine the pattern(s) in each picture.
- Distinguish between the information that remains *constant* and the *variable* information.
- If two or more values are variable, determine how the variable values are *related* to each other. Then express each variable value in terms of a *single variable name*. There is no need to use more than one variable name.

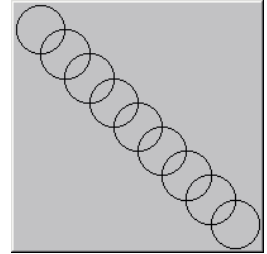
1.



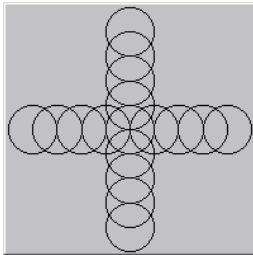
2.



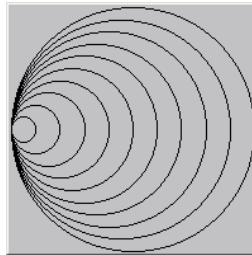
3.



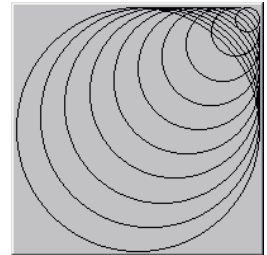
4.



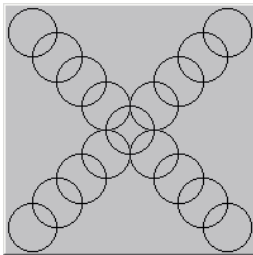
5.



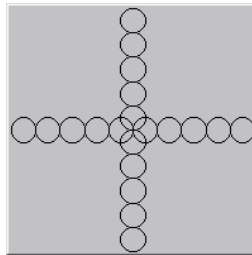
6.



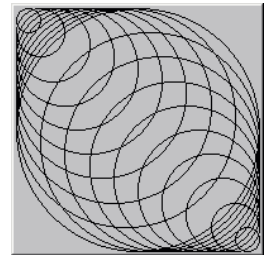
7.



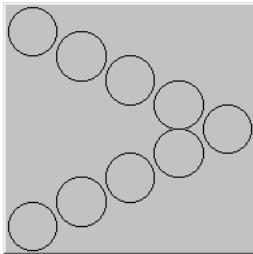
8.



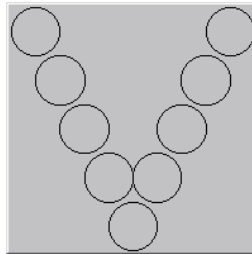
9.



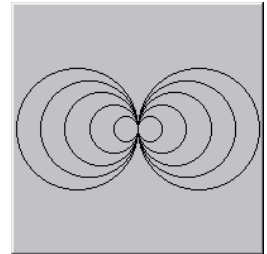
10.



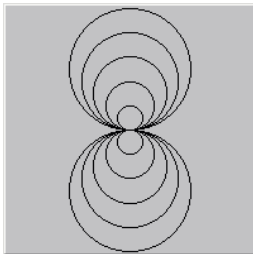
11.



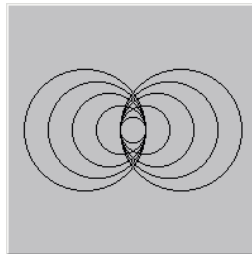
12.



13.

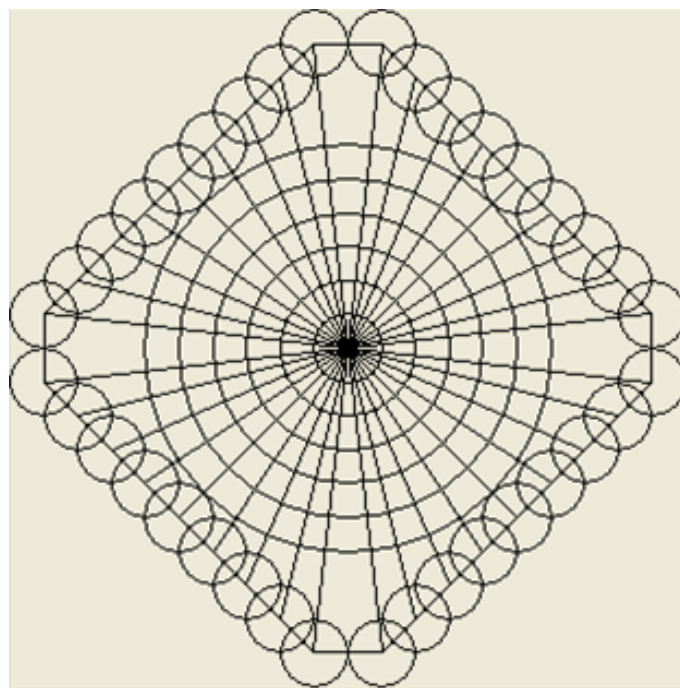
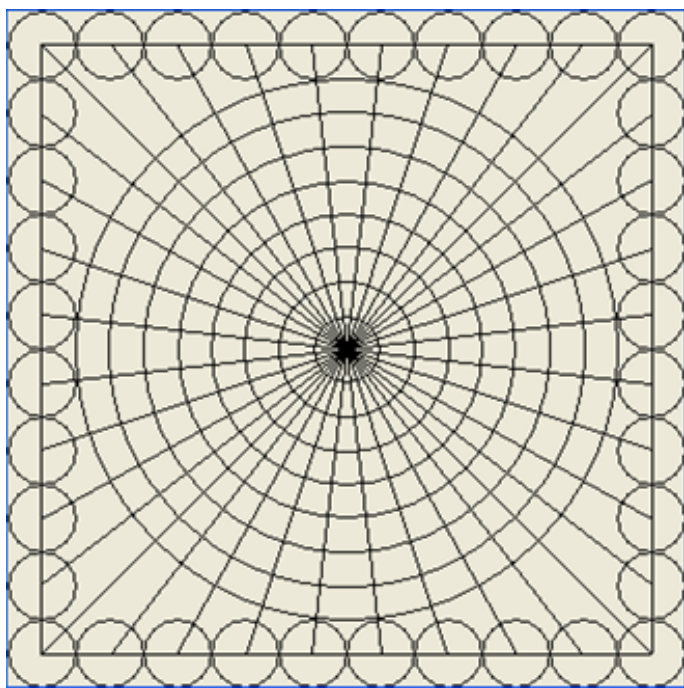
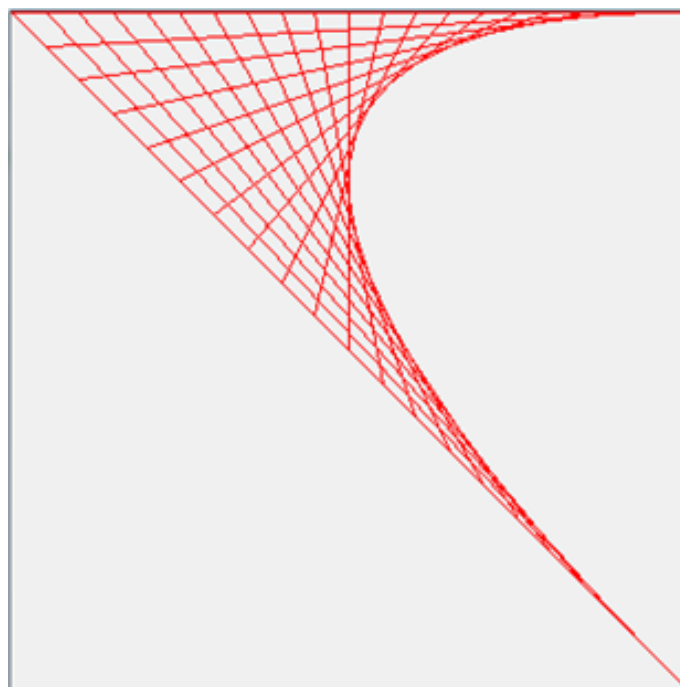
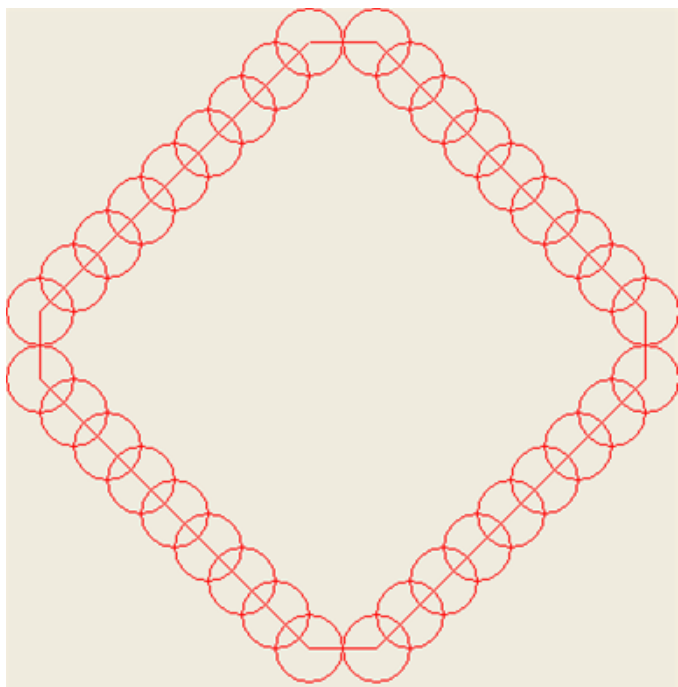


14.



MORE LINE/CIRCLE DRAWING PRACTICE

Use App Inventor to create apps that draw the following pictures.

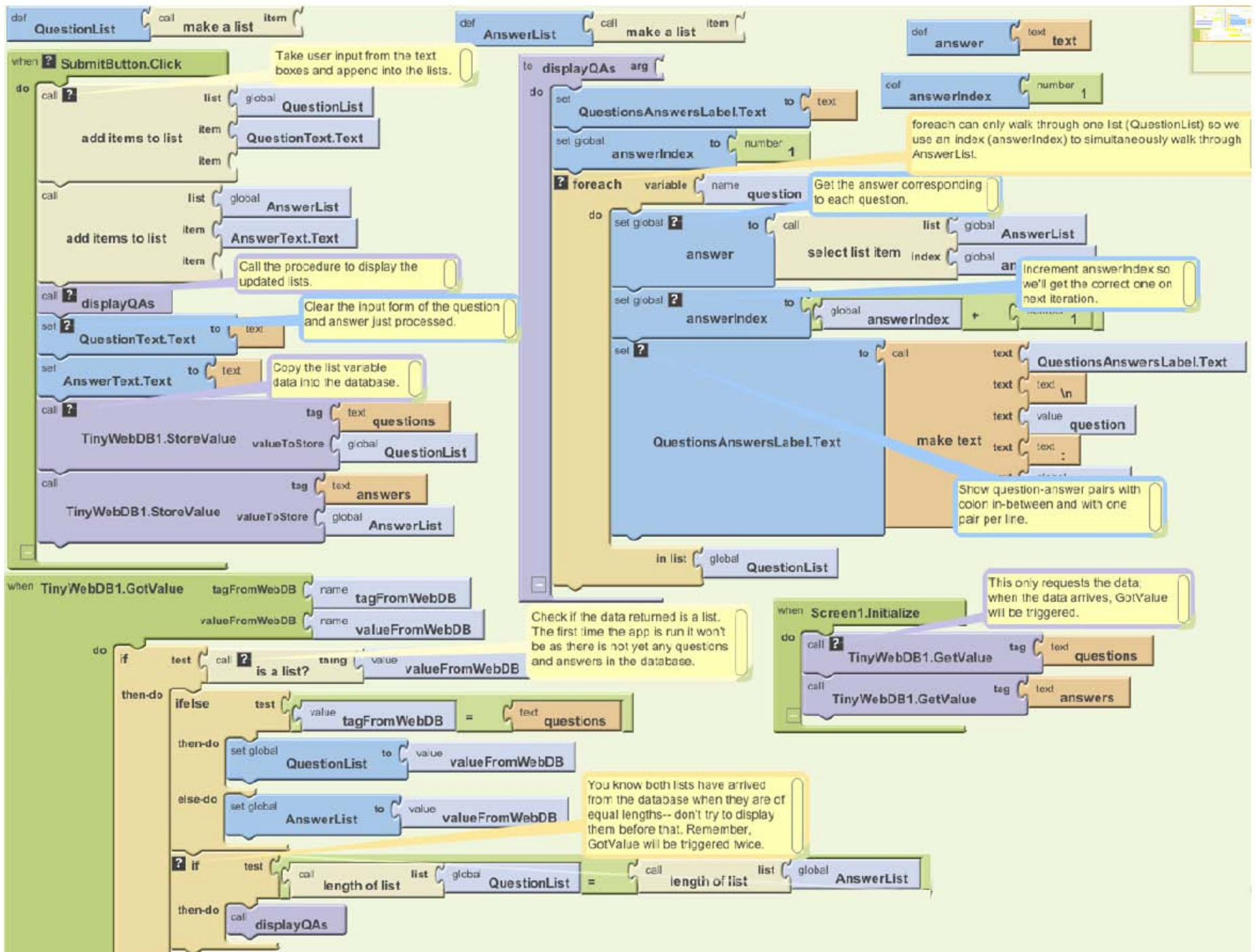


ANALYZING THE MAKEQUIZ APP FROM CHAPTER 10

1. Study the App Inventor blocks on the next page. Then complete the following table.

A. Variable **B.** Component **C.** Property **D.** Method **E.** Event **F.** General Procedure (Built-in)
G. Argument **H.** Event Handler Procedure **I.** General Procedure (Defined by Programmer)

<i>Name</i>	<i>What is it?</i>								
answer	A.	B.	C.	D.	E.	F.	G.	H.	I.
answerIndex	A.	B.	C.	D.	E.	F.	G.	H.	I.
displayQAs	A.	B.	C.	D.	E.	F.	G.	H.	I.
Click	A.	B.	C.	D.	E.	F.	G.	H.	I.
QuestionText	A.	B.	C.	D.	E.	F.	G.	H.	I.
AnswerText	A.	B.	C.	D.	E.	F.	G.	H.	I.
Text	A.	B.	C.	D.	E.	F.	G.	H.	I.
Initialize	A.	B.	C.	D.	E.	F.	G.	H.	I.
GotValue	A.	B.	C.	D.	E.	F.	G.	H.	I.
TinyWebDB1	A.	B.	C.	D.	E.	F.	G.	H.	I.
StoreValue	A.	B.	C.	D.	E.	F.	G.	H.	I.
GetValue	A.	B.	C.	D.	E.	F.	G.	H.	I.
QuestionsAnswersLabel	A.	B.	C.	D.	E.	F.	G.	H.	I.
SubmitButton	A.	B.	C.	D.	E.	F.	G.	H.	I.
Click	A.	B.	C.	D.	E.	F.	G.	H.	I.
SubmitButton.Click	A.	B.	C.	D.	E.	F.	G.	H.	I.
tagFromWebDB	A.	B.	C.	D.	E.	F.	G.	H.	I.
valueFromWebDB	A.	B.	C.	D.	E.	F.	G.	H.	I.
Screen1	A.	B.	C.	D.	E.	F.	G.	H.	I.
Initialize	A.	B.	C.	D.	E.	F.	G.	H.	I.
Screen1.Initialize	A.	B.	C.	D.	E.	F.	G.	H.	I.
make a list	A.	B.	C.	D.	E.	F.	G.	H.	I.
add items to list	A.	B.	C.	D.	E.	F.	G.	H.	I.
TinyWebDB1.GotValue	A.	B.	C.	D.	E.	F.	G.	H.	I.
question	A.	B.	C.	D.	E.	F.	G.	H.	I.
length of list	A.	B.	C.	D.	E.	F.	G.	H.	I.
QuestionList	A.	B.	C.	D.	E.	F.	G.	H.	I.
AnswerList	A.	B.	C.	D.	E.	F.	G.	H.	I.



2. Most universities in North America use a grading system known as the GPA (grade point average) system. It is summarized in the table given below.

<i>Percentage Grade</i>	<i>Grade Point Score</i>
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

<i>Subject</i>	<i>Percentage Mark</i>	<i>Grade Point Score</i>
Math	76%	3.0
Computer Science	84%	3.7
Chemistry	63%	1.7
Physics	45%	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 up to five percentage grades. After the user clicks “Submit,” the app displays the user’s G.P.A. as well as his/her percentage average.

PROGRAMMING PROBLEMS WHOSE SOLUTIONS REQUIRE THE USE OF COUNTED (“FOR”) OR CONDITIONAL (“WHILE”) LOOPS

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/baking/mixing drinks etc, 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** (i.e. too slow and/or require too much memory).
e.g. What are the prime factors of a given number?
- Some problems cannot be solved by a computer (**i.e.** no algorithm exists that can be implemented on a computer).

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

- Please note that the “**For**” looping structure exists only as a convenience! For situations in which the number of repetitions is known beforehand, “**For**” loops allow for easier coding. However, **any** loop logic, including situations in which the number of repetitions is known beforehand, **can** be expressed using a conditional loop!
- Finally, you will solve many of the problems given below using what is known as an **exhaustive search** or a **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.

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 greater than or equal to 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?

<i>Programming Problem</i>	<i>Can you write a solution that only requires a counted loop? Explain.</i>
<p>(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 $28 = 1 + 2 + 4 + 7 + 14$.</p> <p>Write a program that determines whether a given number is perfect. (Exhaustive Search)</p>	<p>Yes / No (Circle One)</p> <p>Why?</p>
<p>(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)</p>	<p>Yes / No (Circle One)</p> <p>Why?</p>
<p>(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)</p>	<p>Yes / No (Circle One)</p> <p>Why?</p>
<p>(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 range of an Integer variable. (Exhaustive Search)</p>	<p>Yes / No (Circle One)</p> <p>Why?</p>
<p>(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)</p>	<p>Yes / No (Circle One)</p> <p>Why?</p>

EUCLID AND THE GCD

Definition of GCD

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

Examples

- $\text{gcd}(8,12) = 4$ because 4 is the largest integer that divides into both 8 and 12
- $\text{gcd}(14,42) = 14$ because 14 is the largest integer that divides into both 14 and 42
- $\text{gcd}(9,28) = 1$ 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 repeatedly and systematically to divide both integers by possible divisors until the greatest common divisor is found. This is illustrated below.

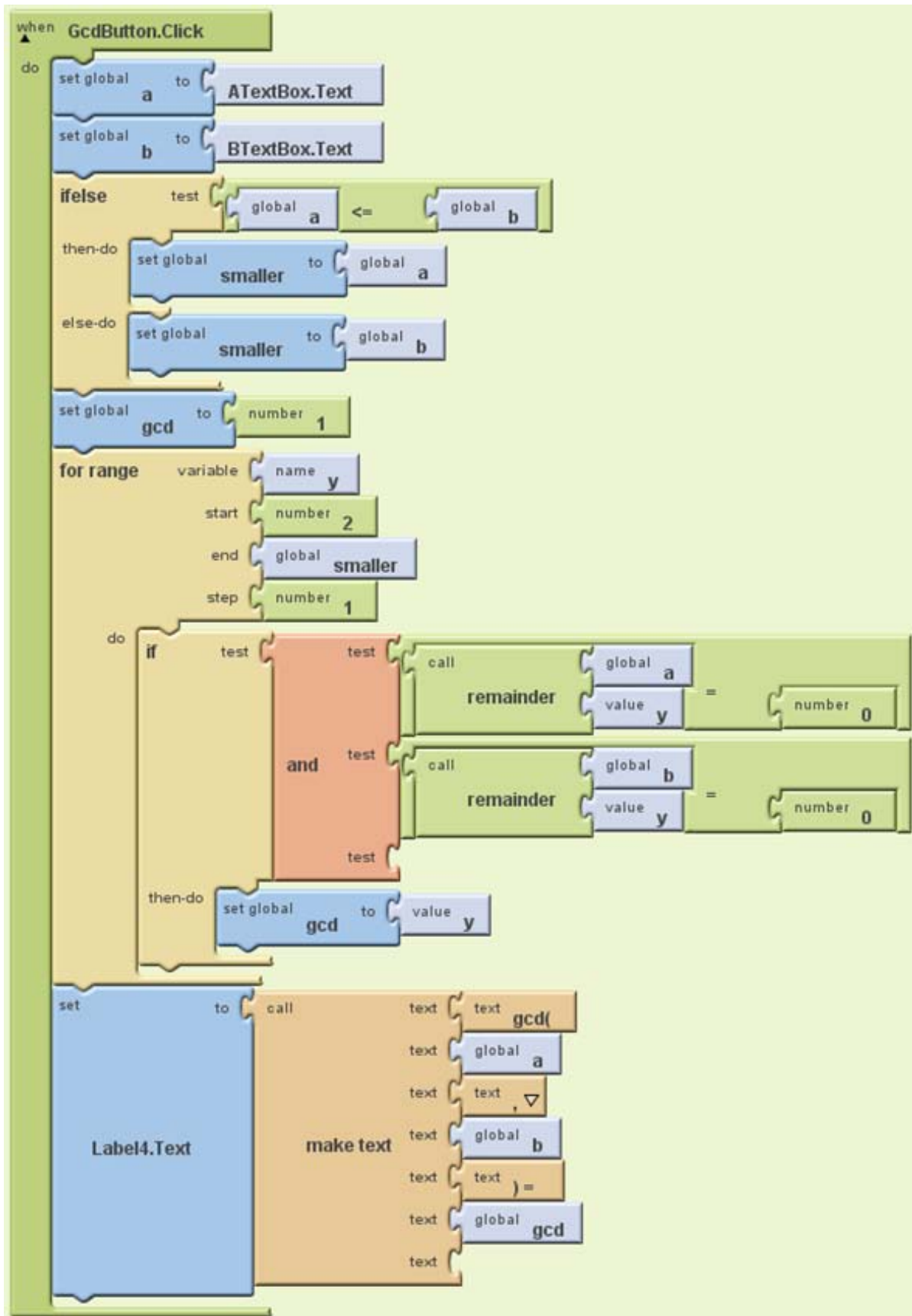
- **a, b:** These variables store the two integers for which the GCD must be found. The values of these two variables remain the same throughout the execution of the code.
- **y:** This variable stores the values of all the integers that we try to divide into both **a** and **b**. The value of this variable is controlled by a “**For**” 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 code in the “**For**” loop, the value of ‘gcd’ remains at 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 Slow GCD Algorithm

The following is a Sub that calculates and displays the GCD of two integers entered by a user. Study the code and then answer the questions on the next page.



Questions

1. Explain the purpose of the “If” statement that immediately precedes the “For” loop.
2. Why does the search for common divisors end at the smaller of “a” and “b?”

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:

$$\text{gcd}(a, b) = \text{gcd}(b, a \bmod b)$$

Note: $a \bmod b$ 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.

$\text{gcd}(2322, 654) = \text{gcd}(654, 2322 \bmod 654) = \text{gcd}(654, 360)$
 $\text{gcd}(654, 360) = \text{gcd}(360, 654 \bmod 360) = \text{gcd}(360, 294)$
 $\text{gcd}(360, 294) = \text{gcd}(294, 360 \bmod 294) = \text{gcd}(294, 66)$
 $\text{gcd}(294, 66) = \text{gcd}(66, 294 \bmod 66) = \text{gcd}(66, 30)$
 $\text{gcd}(66, 30) = \text{gcd}(30, 66 \bmod 30) = \text{gcd}(30, 6)$
 $\text{gcd}(30, 6) = \text{gcd}(6, 30 \bmod 6) = \text{gcd}(6, 0)$
 $\text{gcd}(6, 0) = 6$

Therefore, $\text{gcd}(2322, 654) = 6$.

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

Essentially, the Euclid 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).

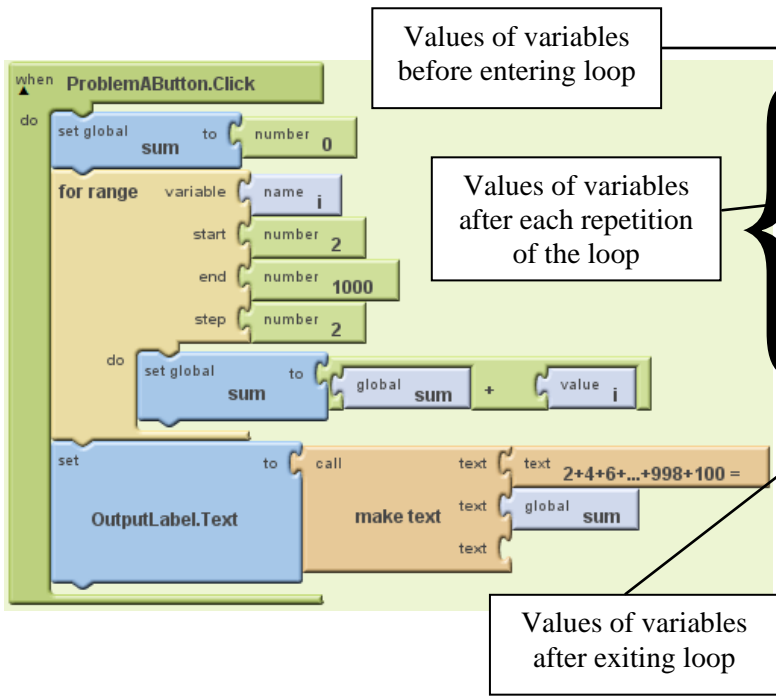
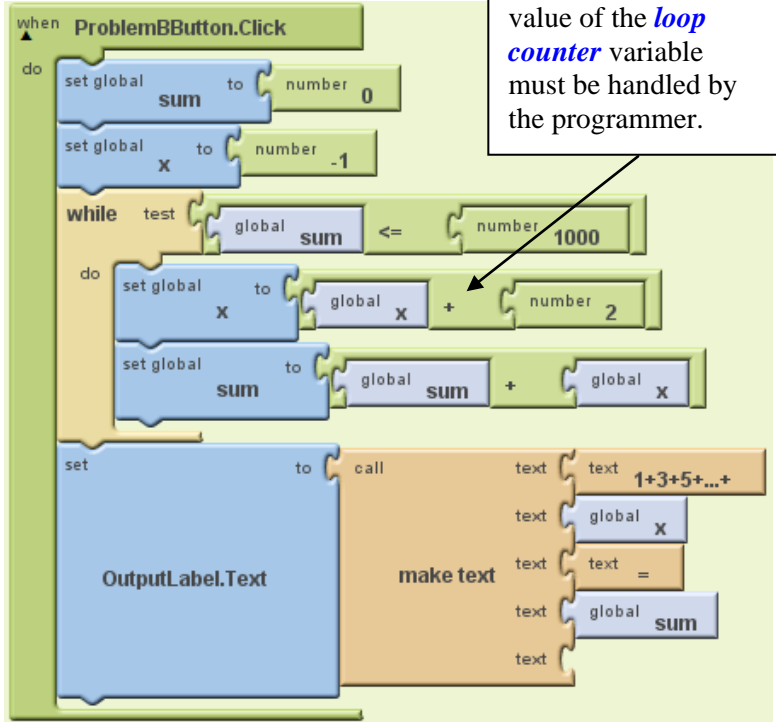
This process continues until the value of ‘b’ is zero.

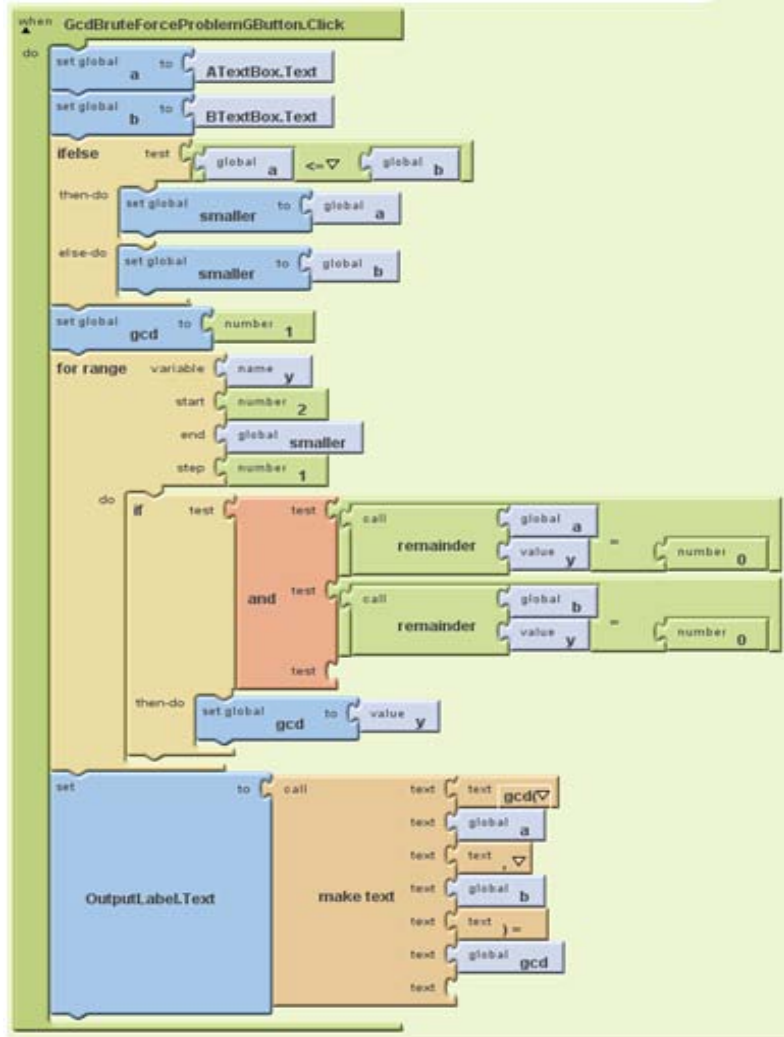
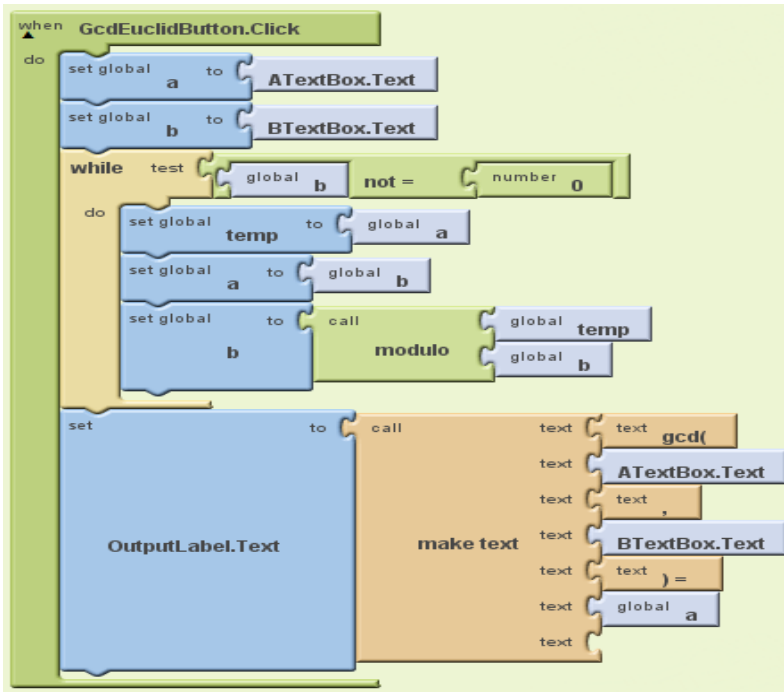
Your Task

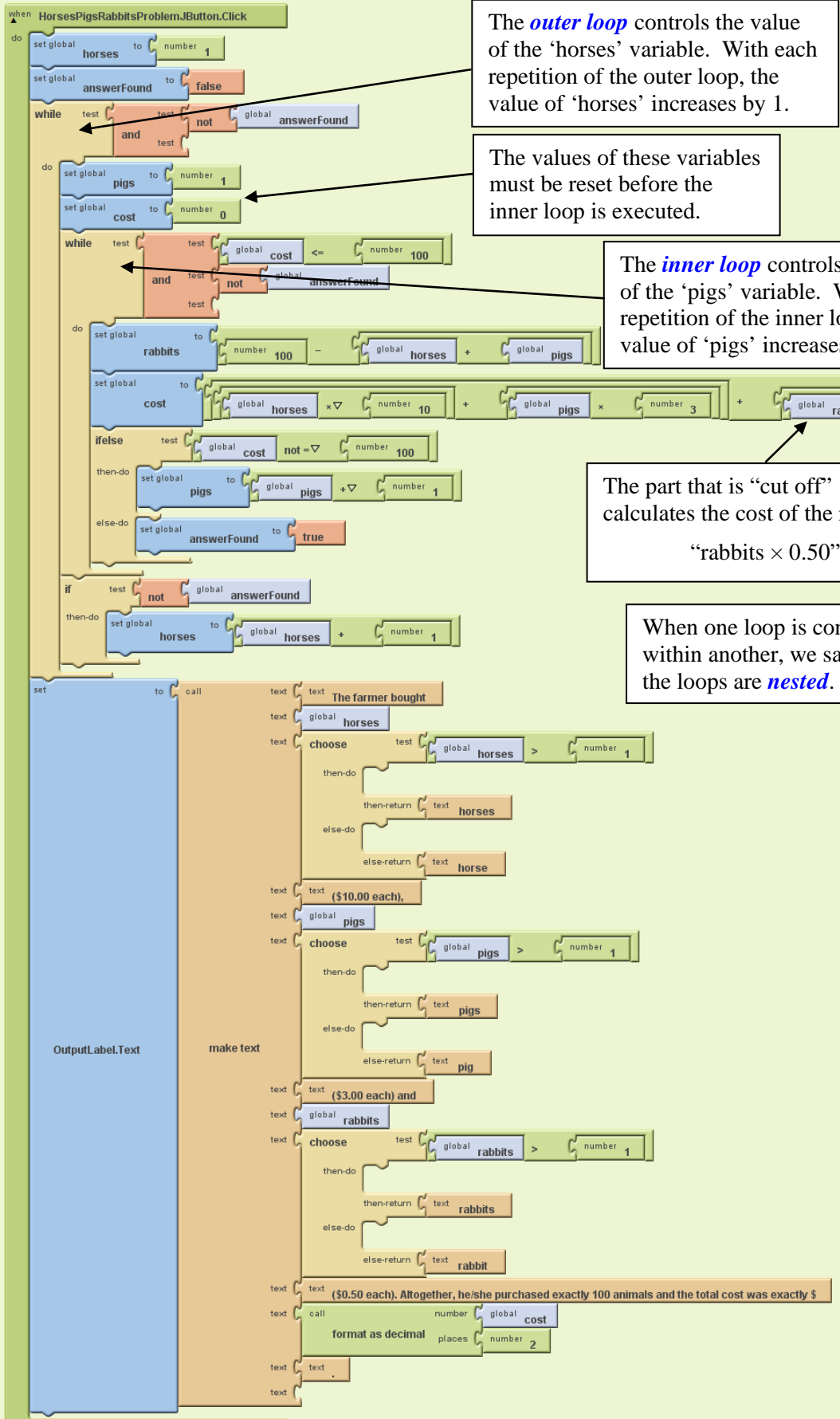
1. Use Euclid’s method to calculate $\text{gcd}(4896, 830)$.
2. How many repetitions would be required by the “slow GCD” algorithm to compute $\text{gcd}(4896, 830)$?
3. Try to write App Inventor code to implement the Euclid GCD algorithm. Test your code thoroughly and debug if necessary.

a	b

SOLUTIONS TO SELECTED PROBLEMS REQUIRING LOOPS

Problem	App Inventor Solution	Notes																						
<p>(a) Write a program to calculate the sum of all positive even integers less than or equal to 1000.</p>	<div><p>Values of variables before entering loop</p><p>Values of variables after each repetition of the loop</p><p>Values of variables after exiting loop</p></div>	<table><thead><tr><th>i</th><th>sum</th></tr></thead><tbody><tr><td>–</td><td>0</td></tr><tr><td>2</td><td>2</td></tr><tr><td>4</td><td>6</td></tr><tr><td>6</td><td>12</td></tr><tr><td>8</td><td>20</td></tr><tr><td>⋮</td><td>⋮</td></tr><tr><td>⋮</td><td>⋮</td></tr><tr><td>998</td><td>249500</td></tr><tr><td>1000</td><td>250500</td></tr><tr><td>–</td><td>250500</td></tr></tbody></table> <p>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.</p>	i	sum	–	0	2	2	4	6	6	12	8	20	⋮	⋮	⋮	⋮	998	249500	1000	250500	–	250500
i	sum																							
–	0																							
2	2																							
4	6																							
6	12																							
8	20																							
⋮	⋮																							
⋮	⋮																							
998	249500																							
1000	250500																							
–	250500																							
<p>(b) Write a program to calculate the sum of all positive odd integers until the sum exceeds 1000.</p>	<div><p>For a “while” loop, the changes in the value of the loop counter variable must be handled by the programmer.</p></div>	<p>This problem is very similar to the previous one. It also makes use of the cash register algorithm.</p> <table><thead><tr><th>x</th><th>sum</th></tr></thead><tbody><tr><td>–1</td><td>0</td></tr><tr><td>1</td><td>1</td></tr><tr><td>3</td><td>4</td></tr><tr><td>5</td><td>9</td></tr><tr><td>7</td><td>16</td></tr><tr><td>⋮</td><td>⋮</td></tr><tr><td>⋮</td><td>⋮</td></tr><tr><td>61</td><td>961</td></tr><tr><td>63</td><td>1024</td></tr><tr><td>–</td><td>1024</td></tr></tbody></table> <p>However, this time a “for” loop cannot be used because the number of repetitions is not known at design-time. For this reason, a “while” loop must be used. The instructions within the body of the loop are repeated as long as the sum remains smaller than or equal to 1000.</p>	x	sum	–1	0	1	1	3	4	5	9	7	16	⋮	⋮	⋮	⋮	61	961	63	1024	–	1024
x	sum																							
–1	0																							
1	1																							
3	4																							
5	9																							
7	16																							
⋮	⋮																							
⋮	⋮																							
61	961																							
63	1024																							
–	1024																							

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

Problem	App Inventor Solution
<p>(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 <i>search</i> for the solution to this problem. (Exhaustive Search)</p>	 <p>The outer loop controls the value of the 'horses' variable. With each repetition of the outer loop, the value of 'horses' increases by 1.</p> <p>The values of these variables must be reset before the inner loop is executed.</p> <p>The inner loop controls the value of the 'pigs' variable. With each repetition of the inner loop, the value of 'pigs' increases by 1.</p> <p>The part that is "cut off" calculates the cost of the rabbits: $\text{"rabbits"} \times 0.50$</p> <p>When one loop is contained within another, we say that the loops are nested.</p> <p>OutputLabel.Text: The farmer bought global horses choose test global horses > number 1 then-do then-return text horses else-do then-return text horse text (\$10.00 each), global pigs choose test global pigs > number 1 then-do then-return text pigs else-do then-return text pig text (\$3.00 each) and global rabbits choose test global rabbits > number 1 then-do then-return text rabbits else-do then-return text rabbit text (\$0.50 each). Altogether, he/she purchased exactly 100 animals and the total cost was exactly \$ call number global cost format as decimal places number 2 text .</p>

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

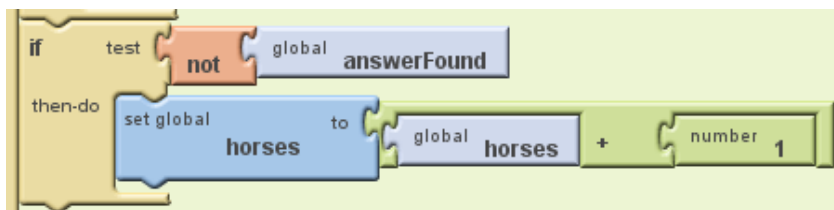
Fourth Repetition of Outer Loop Inner Loop Repeats 5 Times			
horses	pigs	rabbits	cost
4	1	95	\$90.50
4	2	94	\$93.00
4	3	93	\$95.50
4	4	92	\$98.00
4	5	91	\$100.50

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 'temp' 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 loop counters are handled in both "for" and "while" loops.
4. In the "Horses, Pigs, Rabbits" program, what will go wrong if the values of the variables 'pigs' and 'cost' are not reset just before the inner loop is executed?
5. What is the purpose of the "format as decimal" procedure? (See the "Horses, Pigs, Rabbits" program.)

6. In the “Horses, Pigs, Rabbits” program, a “Choose” block is used. Explain the general purpose of this block. What is the specific purpose of this block in the “Horses, Pigs, Rabbits” program?
7. What is the purpose of the ‘AnswerFound’ variable in the “Horses, Pigs, Rabbits” program? How does this variable differ from variables that store numeric values?
8. Who was George Boole? What contributions did he make to mathematics? Given what you have learned about George Boole, explain why it is appropriate to call the variable ‘AnswerFound’ (see question 7) a *Boolean variable*.
9. The following small portion of the “Horses, Pigs, Rabbits” program contains the instruction that increases the value of the variable ‘horses’ by 1. Why is this instruction placed within an “if” block? What would go wrong if it were not placed within an “if” block?



10. 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.

The Stupendous SPLIT Personalities!
Come to Room 224, Period 4 Day 1, to
Bust a Rhyme with the Splitters!



2. Create an App Inventor program that calculates the sum of the squares of the positive integers from 1 to 100. (i.e. $1^2 + 2^2 + 3^2 + \dots + 100^2$)
3. Create an App Inventor program that calculates the sum of the squares of the positive odd integers from 1 to 999. (i.e. $1^2 + 3^2 + 5^2 + \dots + 999^2$)
4. Create an App Inventor program that calculates the sum of the squares of the positive integers until the sum exceeds 100,000,000. (i.e. $1^2 + 2^2 + 3^2 + \dots$, until the sum exceeds 100 million.)
5. Create an App Inventor program that can add, subtract multiply or divide 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.

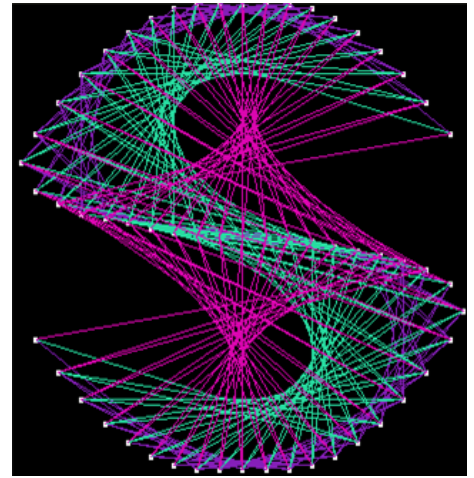
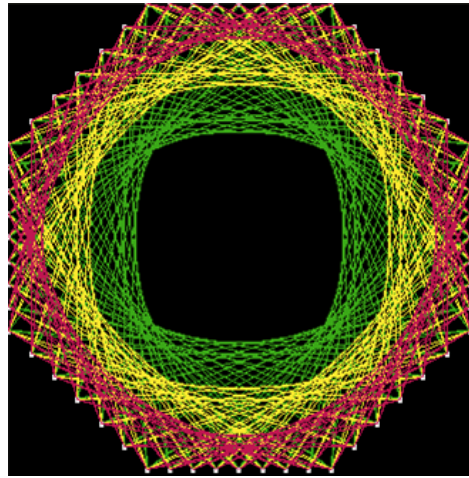
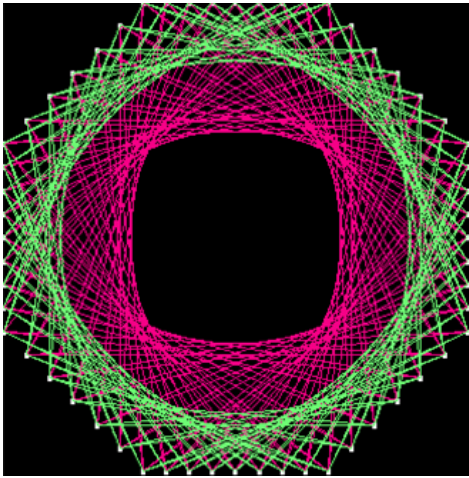
The Stupendous SPLIT Personalities!
Come to Room 224, Period 4 Day 1, to
Bust a Rhyme with the Splitters!



2. Create an App Inventor program that calculates the sum of the cubes of the positive integers from 1 to 100. (i.e. $1^3 + 2^3 + 3^3 + \dots + 100^3$)
3. Create an App Inventor program that calculates the sum of the cubes of the positive odd integers from 1 to 99. (i.e. $1^3 + 3^3 + 5^3 + \dots + 99^3$)
4. Create an App Inventor program that calculates the sum of the cubes of the positive integers until the sum exceeds 100,000,000. (i.e. $1^3 + 2^3 + 3^3 + \dots$, until the sum exceeds 100 million.)
5. Create an App Inventor program that can convert a percentage mark to the equivalent grade point score. (See page 22 for details.)

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

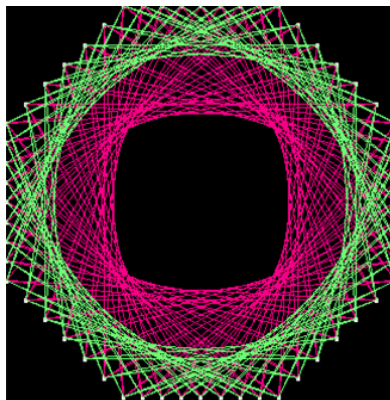
while A < N

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