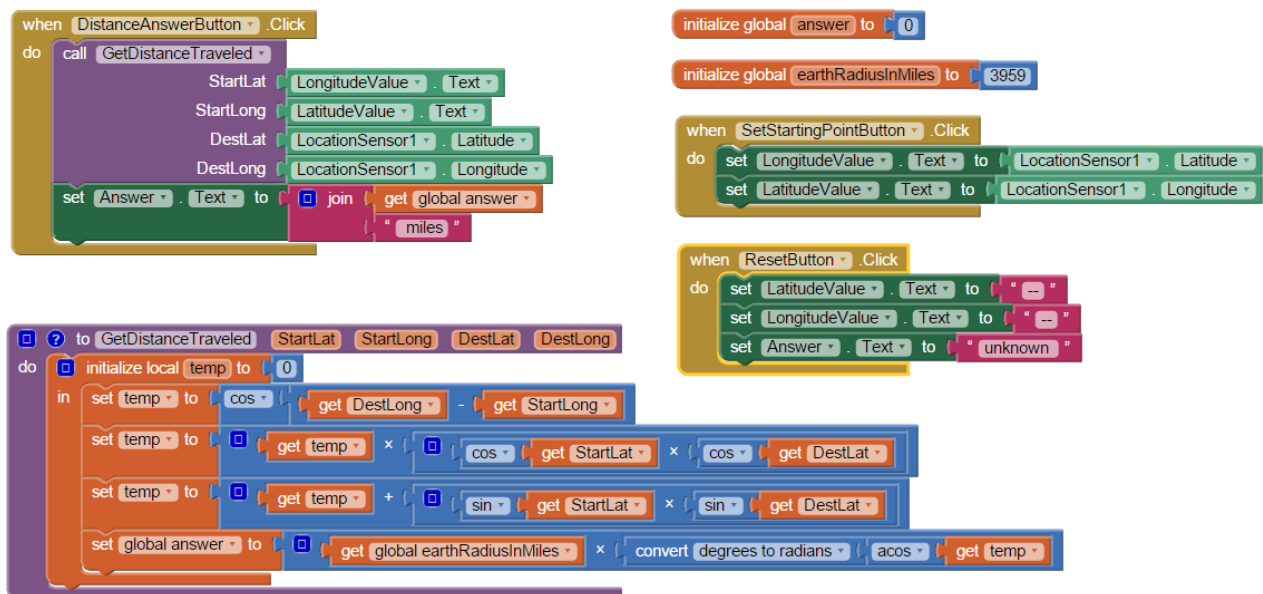


Make This

Chapter 6 - Using the Location Sensor

Your smartphone (and many Android tablets) are equipped with a number of built-in sensors that allow you to gather information. In this tutorial, you'll learn about the App Inventor **LocationSensor** component, which gives you the capability to gather location data from your device's GPS sensor and utilize this information in your apps. This tutorial teaches the following skills:

- **Creating global and local variables**
- **Creating a procedure (function)**
- **Using the LocationSensor component**



How Does LocationSensor Help Me Create More Powerful Apps?

Think about the times you've used a navigation app (like Google Maps) to find a destination. Your smartphone is using the navigation app, GPS sensors, and the GPS satellite system to provide directions so you don't lose your way. Think of some applications where using location information would be helpful to guide users or to provide information within an app.

Note: Before attempting this exercise, complete the Chapter 2 and 3 exercises to familiarize yourself with the App Inventor interface, getting your Android device connected, and starting a new project.

Programming Concepts

To understand the construction of this app, you first need to familiarize yourself with two basic programming concepts: variables and procedures (functions).

What is a variable?

Variables are used in almost every computer programming language. Think of variables as containers that hold values. Variables are handy for storing values that you will need to use in some part of an app, often repeatedly. A variable might be hours worked in a week or an address. In App Inventor, there are two types of variables you can use: **global** and **local** variables.

Global Variables

Global variables provide maximum flexibility. As the name implies, you can access a global variable from anywhere within your app. You can obtain the current value of the variable (to use it in a calculation for instance) or set its value to something else (perhaps a calculated value generated by your app). In **Blocks** view, from the **Variable** drawer, you drag out a



block to create a global variable.

Local Variables

A **local variable** is a variable that is only used in a specified part of a program. Usually, local variables are defined (declared) inside of a procedure (function). Local variables are also used to pass information to functions, in which case they are referred to as arguments. As opposed to global variables, which can be accessed from anywhere in the app, local variables can only be accessed within the specific function where they are defined or passed in as an argument.



In **Blocks** view, from the **Variable** drawer, you drag out a block to create a local variable.

You can assign a name to each variable you create. In the **initialize** block, just click **name** and enter the name you want to assign to the variable. Variable names should be unique within an app. You cannot have two global variables with identical names within the same app. You also cannot use identical names for two local variables contained within the same procedure.

What is a procedure?

A **procedure** (which is also called a **function** or **method**) is a set of instructions designed to perform a calculation or a task. A recipe for baking a cake is an example of a procedure.

Procedures are used when a task is going to be done repeatedly. Although you could create the same block multiple times to perform a repetitive calculation, this would be a waste of time, energy, and space in the app's programming. It is easier to create one procedure that is reused.

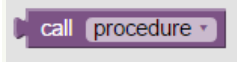


There are two types of procedure blocks you can use: . The

“do” block allows you to gather up a series of blocks to perform a set of steps. The result block is used when you need the procedure to return a result.

For example, you might be making an app for a carpet installer that computes the total floor area that needs to be carpeted. If there are four rooms, the length and width of each room would be measured, and the area of each room would be calculated. Then the four areas would be added together to obtain the total area being carpeted. There is no need to create a separate block for each room. Since the calculation is the same (width x length), it is more efficient to create a procedure that calculates room area and use it over and over again.

Once a procedure block is dragged out onto the Viewer, App Inventor makes a call block



available in the Procedure drawer. A call block is used for invoking the procedure.

Now that you understand more about variables and procedures, let's proceed to build an app that uses the **LocationSensor** component.

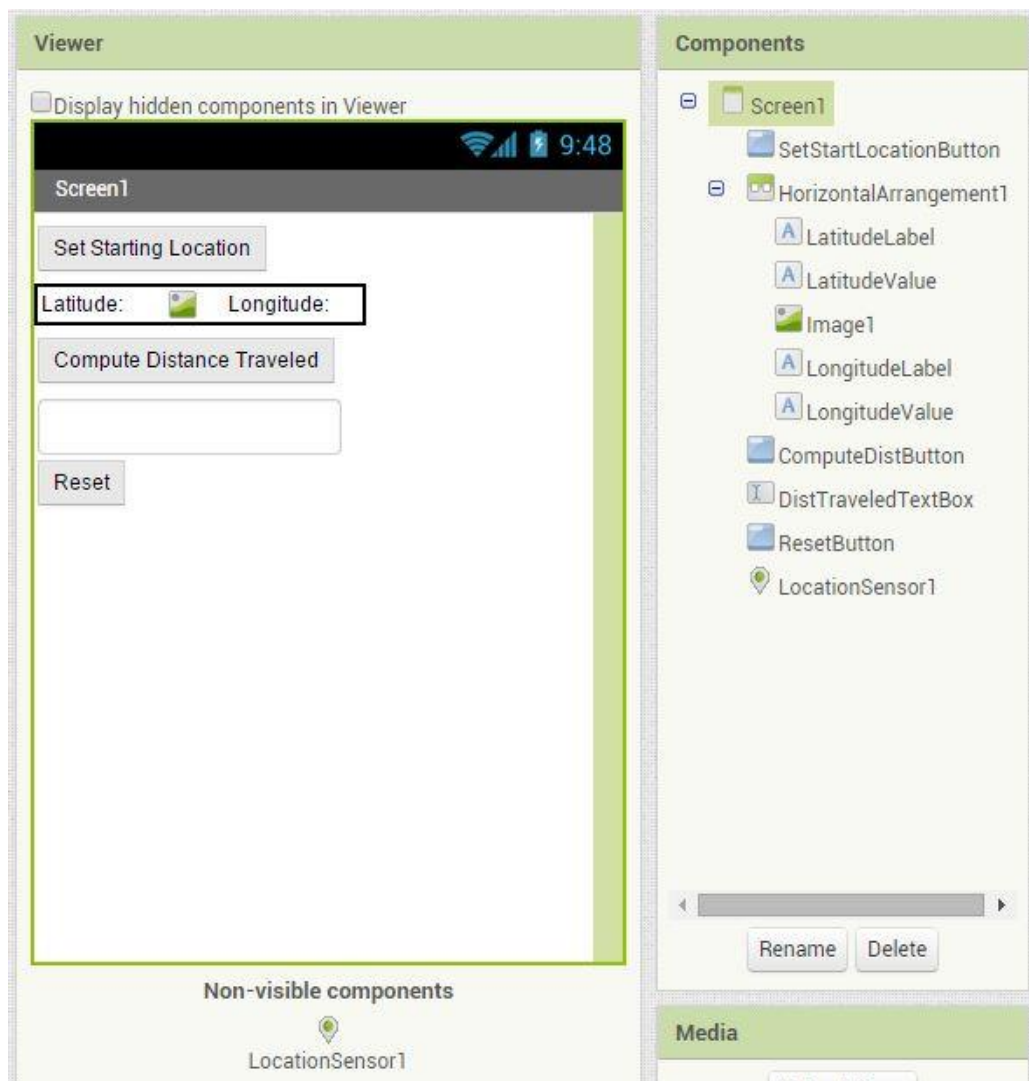
Building the LocationSensor Example App

This app will record a starting location with your device (using latitude and longitude). Then, after traveling some distance, the app will calculate how far you have traveled from your starting point.

1. Navigate to <http://appinventor.mit.edu/explore/>. If necessary, sign in with your Google Account.
2. Start a new project named *LocationSensorExample*.
3. Connect App Inventor to your Android device.

Part 1 - Constructing the Interface

From previous chapter exercises, you should be familiar with constructing an interface. Your relatively simple layout should look like this:



Tip: When working in Blocks view, it can sometimes be difficult to remember which component is supposed to be used for what function when using default names (like **Button1**). Therefore, it is a good idea for you to change the names of your components to something more descriptive of its function.

You can build the interface shown above by dragging out the components shown in the following table:


Component	Palette Group	Component Name	Function
Button	User Interface	SetStartLocationButton	To record your starting location
Horizontal Arrangement	Layout	HorizontalArrangement1	To hold the location labels
Label	User Interface	LatitudeLabel	Let the user know what is displayed
Label	User Interface	LatitudeValue	Display the latitude value
Image	User Interface	Image1	A placeholder to space out the labels to make them more readable
Label	User Interface	LongitudeLabel	Let the user know what is displayed
Label	User Interface	LongitudeValue	Display the longitude value
Button	User Interface	ComputeDistButton	User clicks this to compute distance traveled
TextBox	User Interface	DistTraveledTextBox	Text box to display the distance traveled
Button	User Interface	ResetButton	User clicks this to reset starting location
LocationSensor	Sensors	LocationSensor1	Sense where the device is

Set the properties of each component in the following ways:

- Change the Text for **SetStartLocationButton** to *Set Starting Location*.
- Change the Text for **LatitudeLabel** to *Latitude:* (adding a space after the colon).

Tip: When entering label text, you can include spaces which helps readability.

- Change the Text for **LongitudeLabel** to *Longitude:* (adding a space after the colon).
- Delete the default Text for **LatitudeValue** and **LongitudeValue**.

Note: When you delete the default text for a label component, the label will show up as a green dash  in the viewer only when that label component is selected in the Components window. If any other element is selected, the label will be blank on the screen and may look as if it disappeared. Rest assured, it is still there and listed in the Components section!

- Set the Width of **Image1** to *25 pixels*.

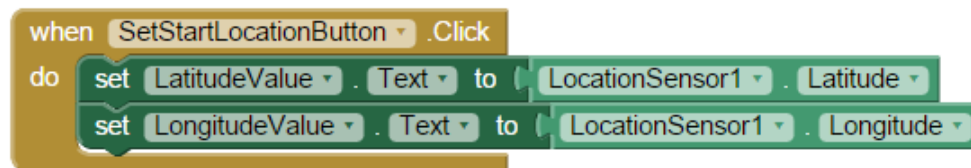
Tip: Image components do not have to have an image associated with them. If they have no image associated with them, they appear as blank space on the interface. They are often used in this manner to improve readability on an interface.

- Change the **Text** for **ComputeDistButton** to *Compute Distance Traveled*.
- Change the **Text** for **ResetButton** to *Reset*.
- Delete the default text in **Hint** for **DistTraveledTextBox**.

Part 2 - Programming the Starting Location and Reset Buttons

1. Switch to the **Blocks** view.

Starting Location Button



2. In the **Blocks palette**, select **SetStartLocationButton**, and select the **when SetStartLocationButton is clicked** block from the drawer.
3. In the **Blocks palette**, select **LatitudeValue**, and select the **set LatitudeValue.Text to** block from the drawer. Drag it inside the **when SetStartLocationButton is clicked** block.
4. In the **Blocks palette**, select **LongitudeValue**, and select the **set LongitudeValue.Text to** block from the drawer. Drag it inside the **when SetStartLocationButton is clicked** block.
5. In the **Blocks palette**, select **LocationSensor1**, and select the **LocationSensor1.Latitude** block and insert it into the open socket on the **set LatitudeValue.Text to** block.
6. Select **LocationSensor1** again and select the **LocationSensor1.Longitude** block and insert it into the open socket on the **set LongitudeValue.Text to** block.

Now test the button and see if your current latitude and longitude displays. Here are some reasons why your device may not display a latitude and longitude:

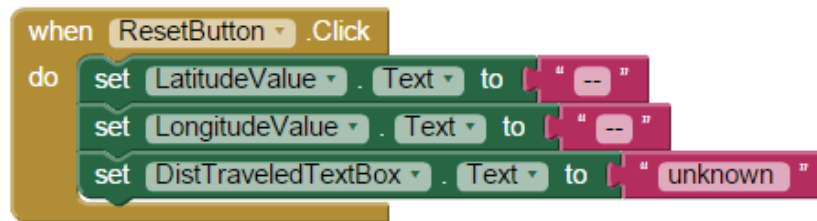
- **Location services are turned off on your device** – On most devices, tap **Settings**, then **Connection** and ensure Location services are turned on. (Note: Google the instructions for turning on location services for your particular device).
- **The Location Locating Method on your device is set to use Wi-Fi** – While this saves power, it doesn't provide the most accurate information (and on some WiFi networks it

fails to provide location information at all). Try changing the locating method on your device to **GPS only** for testing this app.

Alert: In the emulator, the latitude and longitude values will always return as zero.

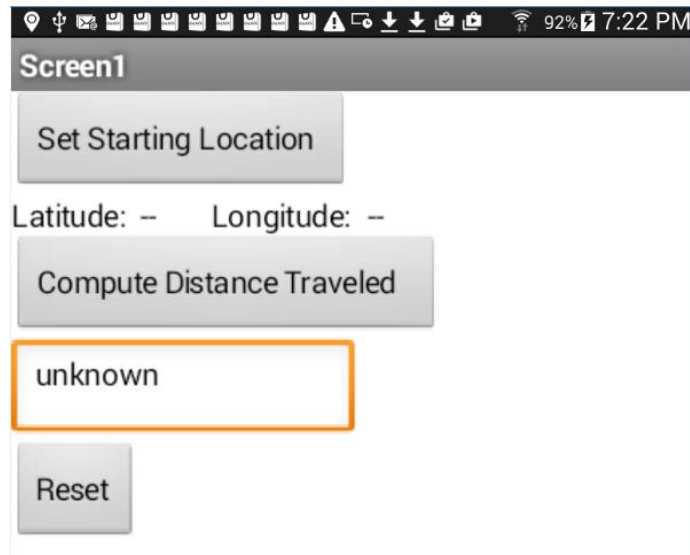
How do you know if the latitude and longitude values are correct? Go to the [Find Latitude and Longitude](#) website and enter the postal code of your location. The values shown on your device should be close to the values shown on this website.

Reset Button



1. In the **Blocks palette**, select **ResetButton**, and select the **when ResetButton .Click** block from the drawer.
2. In the **Blocks palette**, select **LatitudeValue**, and select the **set LatitudeValue . Text to** block. Drag it inside the **when ResetButton .Click** block.
3. In the **Blocks palette**, select **LongitudeValue**, and select the **set LongitudeValue . Text to** block. Drag it inside the **when ResetButton .Click** block.
4. In the **Blocks palette**, select **DistTraveledTextBox**, and select the **set DistTraveledTextBox . Text to** block. Drag it inside the **when ResetButton .Click** block.
5. In the **Blocks palette**, select **Text**, and click and drag three **" "** blocks onto the **Viewer** and insert one in each open socket. Insert two hyphens (--) – in the first two blocks and the text *unknown* in the third block.

Now test the Reset button. Your screen should look like this after the Reset button is pressed:



Part 3 -The Compute Distance Traveled Procedure

When the Compute Distance Traveled button is pressed, you want your app to calculate the distance between the starting location the app stored and the present location per the location sensor. This requires creating a procedure (make sure you have reviewed the section at the beginning of this document explaining procedures) for calculating the distance between two latitudes and longitudes.

For the calculation, we need to use the spherical law of cosines:

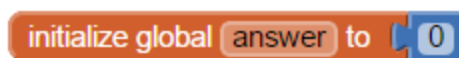
Distance = $\text{acos}(\text{SIN}(\text{lat1}) * \text{SIN}(\text{lat2}) + \text{COS}(\text{lat1}) * \text{COS}(\text{lat2}) * \text{COS}(\text{lon2} - \text{lon1})) * \text{Earth Radius in Miles}$





Although this looks complicated, it is a relatively simple formula using basic trigonometry functions...all of which are available in App Inventor. The earth's radius in miles is 3,959. Your app only needs four pieces of information to compute this formula: the starting and ending latitude and longitude.

Note: For a detailed explanation of the spherical law of cosines and computing distances between to geographic points, try one of these links: <http://www.movable-type.co.uk/scripts/latlong.html> or <http://gis.stackexchange.com/questions/4906/why-is-law-of-cosines-more-preferable-than-haversine-when-calculating-distance-b>

Create Global Variables

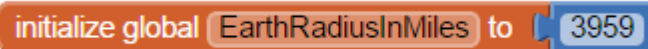
First you need a place to store your calculation (the distance traveled). You will create a global variable (see the discussion of variables at the beginning of this document) to hold the results of your calculation.





1. In the **Blocks palette**, under **Built-in**, select **Variables**, and select the  block.
2. In the **Blocks palette**, under **Built-in**, select **Math**, and select the  block. Insert it into the open socket on the  block. Click  and change the name of the variable to *answer*.

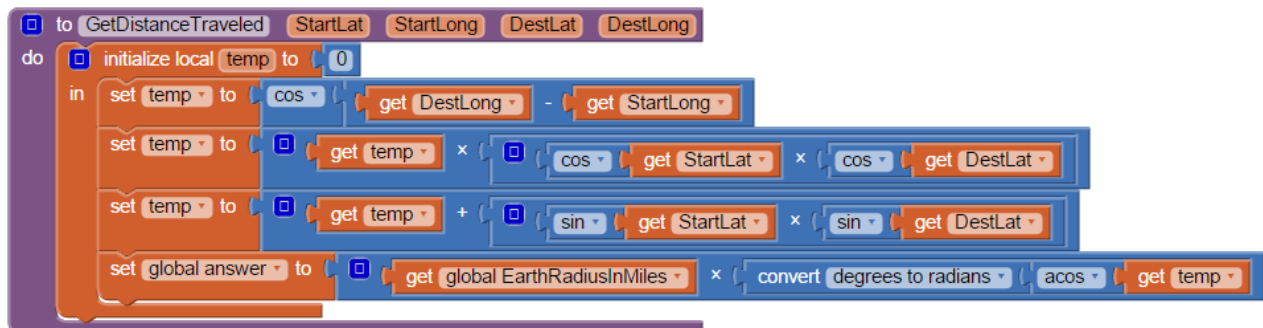
This creates the global variable called *answer* and sets its initial value to zero. We now have a variable in which we can store the result of our distance calculation.

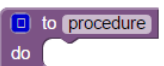
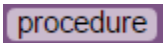
Another reason for creating variables is to store values in them (especially constants) that we can use in other calculations. Now create a global variable for the Earth's radius.

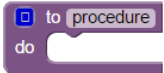

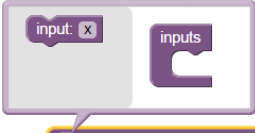
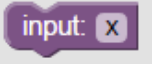



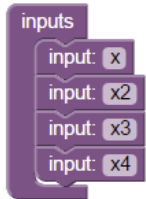
3. Select another  block from the **variables drawer**. Change the name of this variable to *EarthRadiusInMiles*.
4. From the math drawer, select another  block and insert it in the open socket. Change its value to 3959.

Build the Procedure to Calculate Distance Traveled

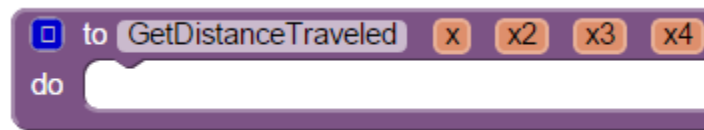


5. In the **Blocks palette**, under **Procedures**, select the  block. Click on  and change the name of the procedure to *GetDistanceTraveled*.

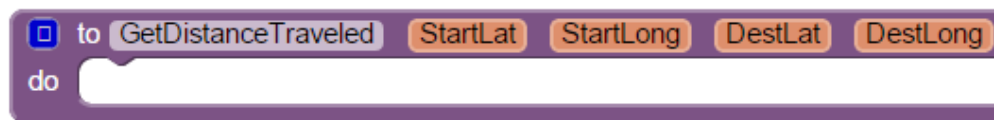
6. On the  block, click the mutator button  to open the dialog box . Drag four  blocks into the  block as shown here:



. Your procedure block should now look like this:



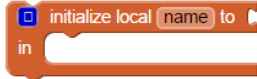
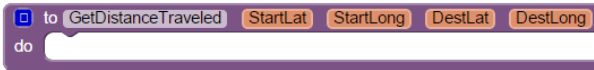
7. Click on each of the orange arguments and change their names as shown below:

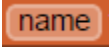



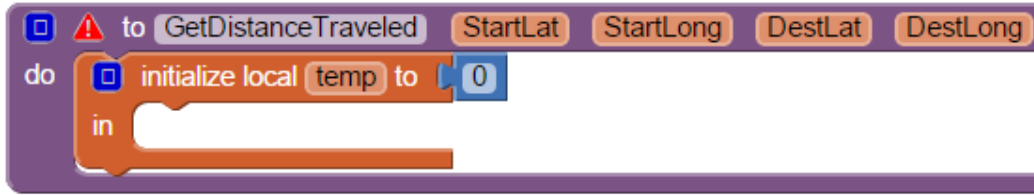
These arguments represent the four pieces of data that will be used by this procedure to perform the distance calculation. Now let's consider our formula:

$\text{acos}(\text{SIN}(\text{lat1}) * \text{SIN}(\text{lat2}) + \text{COS}(\text{lat1}) * \text{COS}(\text{lat2}) * \text{COS}(\text{lon2} - \text{lon1})) * \text{EarthRadiusInMiles}$

Lat1 and Lon1 are the starting latitude and longitude, while Lat2 and Lon2 are the destination coordinates. The formula has been color coded to indicate how it can be broken into pieces for the calculation. You will use a local variable called *temp* to temporarily hold the values of your calculations within the procedure.

8. In the **Blocks palette**, under **Variables**, select the  block. Drag it inside the  block.

9. Click on  and change the local variable name to *temp*.
10. From the **Math drawer**, select a  block and insert it in the open socket. This sets the value of *temp* to zero. Your block should look like this:



Alert: Don't worry about the warning symbol. It is only there because you are not yet finished building the procedure. It will disappear later.

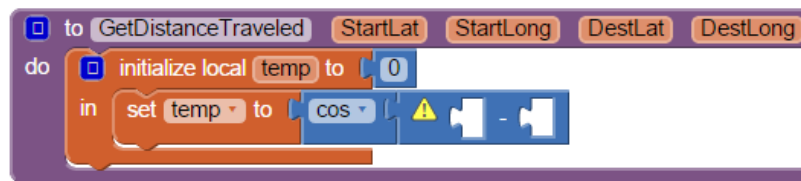
11. From the **Variables drawer**, select a block. Drag it inside the local variable block. Click the down arrow in the block and select *temp* from the drop-down menu.

Now you are ready to define the first part of the calculation as shown above in red:

$\text{COS}(\text{lon2} - \text{lon1})$

12. From the **Math drawer**, select a block and insert it in the open socket.

13. From the **Math drawer**, select a subtraction block and insert it in the open socket on the block. Your block should look like this:



14. With your cursor, hover over the **DestLong** argument. From the dialog box that displays, select the block and insert it in the first open socket on the block.

15. Now hover over the **StartLong** argument. Select the block from the dialog box and insert it in the open socket on the block. Your block should look like this:



So far what your block does is calculate $\text{COS}(\text{lon2}-\text{lon1})$ and store the value in the local variable *temp*.

You will now calculate the next part of the formula $[\text{COS}(\text{lat1})*\text{COS}(\text{lat2})]$ and multiply it by the value stored in *temp*. This will provide you with a value representing this part of the formula $\text{COS}(\text{lat1})*\text{COS}(\text{lat2})*\text{COS}(\text{lon2}-\text{lon1})$, which you will then store in the *temp* variable for the next piece of the calculation.

16. Right-click on the **set temp to** block and select duplicate from the drop-down menu.

Insert the new block into the procedure block below the first **set temp to** block.

Duplicate it again and insert the new block under the others. You'll have three set temp to boxes.



17. Drag out two **multiplication** blocks from the math drawer. Insert one of these blocks into the right-hand socket of the other like this:



. The result should look like this:



. Now insert this block into the open socket on the

second **set temp to** block.

18. With your cursor, hover over the **temp** variable name (at the top of the *initialize local temp to* block) and select **get temp** from the dialog box. Insert this in the first open socket of the math block.



19. Select two more **COS** blocks from the **Math drawer**. Drop them into the remaining



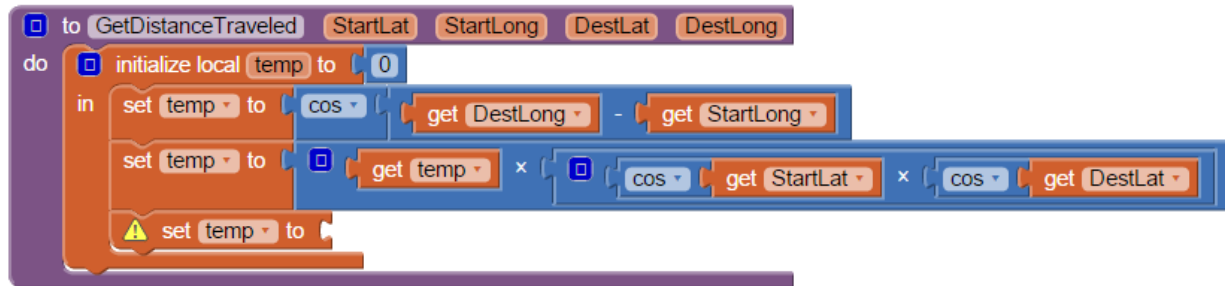
two slots on the block.

20. Hover over the **StartLat** argument and drag a **get StartLat** block to the middle socket of the multiplication block.



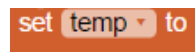
21. Hover over the **DestLat** argument and drag a **get DestLat** block to the remaining open socket in the multiplication block. Your block should now look like this:




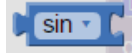


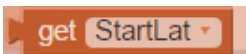

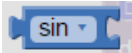
22. Select an **addition** block  and a **multiplication** block  from the **Math drawer**. Insert the multiplication block into the second socket of the

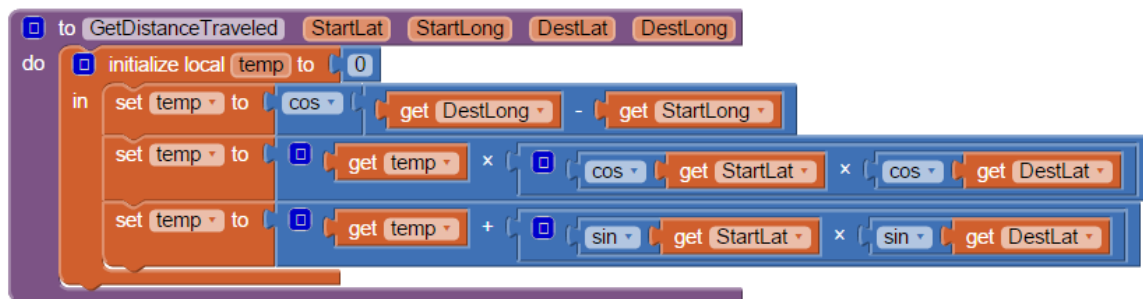
addition block like this:

 block.

23. Duplicate the  block and insert it into the open socket in the addition block.

24. From the **Math drawer**, select two  blocks and insert them in the two open sockets in the multiplication block.

25. Duplicate the  and the  blocks. Insert them into the  blocks. Your block should now look like this:




The resulting value stored in *temp* will now comprise these parts of the formula:

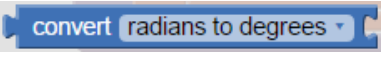
$\text{SIN}(\text{lat1}) * \text{SIN}(\text{lat2}) + \text{COS}(\text{lat1}) * \text{COS}(\text{lat2}) * \text{COS}(\text{lon2} - \text{lon1})$. Since your next calculation will finish the formula, you will store the answer in the global variable *answer*.


26. From the **Variables drawer**, select a  block. Insert it into the local variable block underneath the last  block. Click the drop-down arrow in the



 block and select *global answer* from the drop-down list.

27. From the **Math drawer**, select a **multiplication** block  and attach it to the **set global answer to** block.

28. From the **Variables drawer**, select a **get** block . Click the down arrow and select **global EarthRadiusInMiles** from the drop-down list. Insert this block into the first open socket of the **multiplication** block.

29. From the **Math drawer**, select a **convert radians to degrees** block . Click the down arrow and select **degrees to radians** from the drop-down menu. Insert this block into the open socket of the multiplication block.

30. From the **Math drawer**, select a **cos** block . Click the down arrow and select **acos** from the drop-down menu. Insert this block into the open socket of the **convert degrees to radians** block.

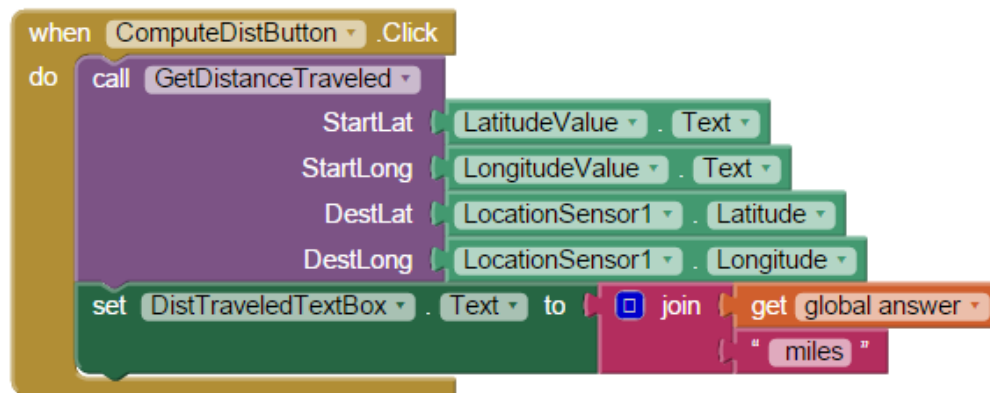
31. Duplicate the **get temp** block  and insert it into the open socket in the **acos** block . Here's the final block:



Alert: The **acos** function in App Inventor returns a value in degrees. For this formula, you need a value in radians. This is why you must use the *convert degrees to radians* math block.

Although the procedure block is now built, the procedure won't actually execute until it is called. Your last step in this app is to make the **ComputeDistButton** call the **GetDistanceTraveled** procedure.

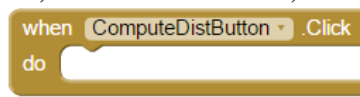
Part 4 -The Compute Distance Traveled Button



1. In the **Blocks palette**, select **ComputeDistButton**, and select the



2. In the **Blocks palette**, under **Procedures**, select the

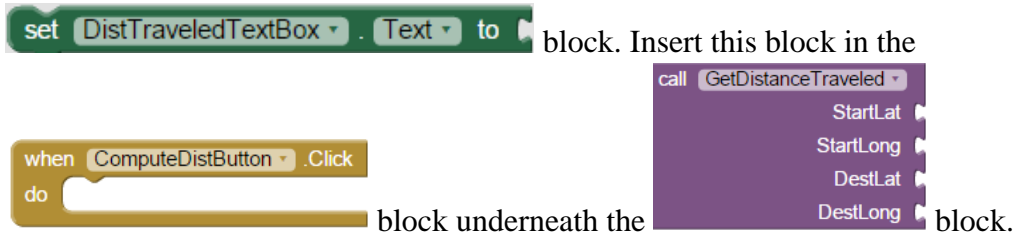


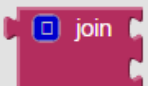
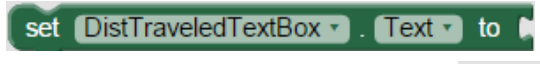
this block into the


Tip: Whenever you create a procedure, App Inventor adds a call block for that procedure to the procedure drawer. The call block will have a list of all of the arguments defined in the procedure. These arguments need to be filled with values so that they can be passed to the procedure. The procedure needs argument values to make the proper computations.

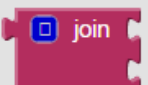
3. In the **Blocks palette**, select **LatitudeValue**, and select the block from the drawer. Insert this block into the **StartLat** socket.
4. In the **Blocks palette**, select **LongitudeValue**, and select the block from the drawer. Insert this block into the **StartLong** socket.
5. Duplicate the and blocks from the **when SetStartLocationButton.Click** block you previously created. Insert these blocks in the **DestLat** and **DestLong** sockets, respectively.

6. In the **Blocks palette**, select **DistTraveledTextBox**, and select the

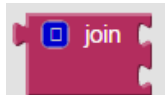



7. In the **Blocks palette**, select **Text**, and select the  block and insert it into the open socket on the  block.

8. In the **Blocks palette**, select **Variables**, and select the  block and insert it

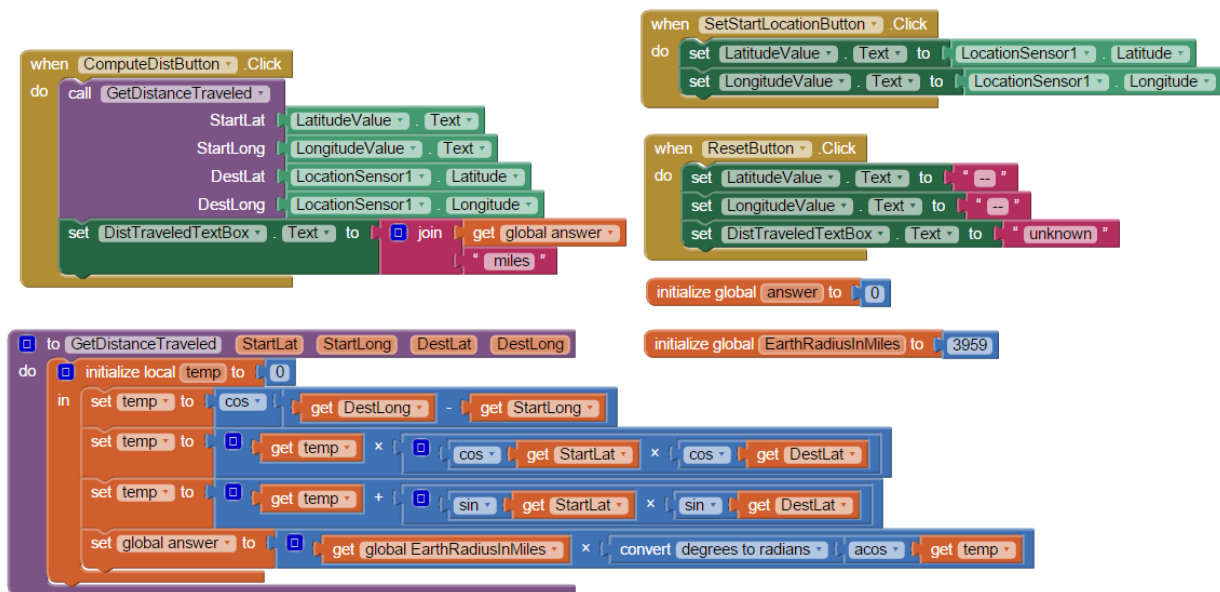
into the top open socket on the  block. Click the down arrow and select *global answer* from the drop-down menu.

9. In the **Blocks palette**, select **Text**, and select the  block and insert it into the

open socket on the  block. Insert *miles* (with a space in front of it) into the  block.

The entire app should resemble this:

, vur



Now test the app. You will need to capture a starting point then move to a different location before calculating the distance.

Alert: As explained previously, using GPS only location settings on your device to test this app is a good idea. Say you are using Wi-Fi to measure location. If you walk to the next classroom and then try to measure the distance traveled, it might show up as zero. This is because you are connected to the same Wi-Fi access point in both classrooms and your location sensors think you are actually in the same place since they are reporting the location of the access point, not your physical location!

Alert: In the emulator, the latitude and longitude values will always return as zero. Therefore, distance will not computer in the emulator.

Now that you’ve worked through the project once, go back and modify the components to better suit your own preferences, or try one of the *Extensions to This Project* below.

Extensions to This Project

1. Modify the app so that it stores, and displays, the ending latitude and longitude.
2. Modify the app to make it a “Hey Dude, Where’s My Car App”. Have the app store the location of where you parked your car. Then modify the app to launch Google Maps and show you how to get back to your car from your current location. Hint: There is a tutorial for an app similar to this here: <http://www.appinventor.org/bookChapters/chapter7.pdf>

Resources

- [Sensor Components: LocationSensor](#)
- [AI2 Procedures](#)
- [AI2 Variables](#)
- [User Guide for App Inventor 2](#)
- [Guide to Understanding Blocks](#)

MIT App Inventor is a blocks-based programming tool that allows everyone, even novices, to start programming and build fully functional apps for Android devices. Google's Mark Friedman and MIT Professor Hal Abelson co-led the development of App Inventor while Hal was on sabbatical at Google. App Inventor runs as a Web service administered by staff at MIT’s Center for Mobile Learning - a collaboration of MIT’s Computer Science and Artificial Intelligence Laboratory (CSAIL) and the MIT Media Lab. MIT App Inventor supports a worldwide community of nearly 3 million users representing 195 countries worldwide. App Inventor is an open-source tool that seeks to make both programming and app creation accessible to a wide range of audiences. App Inventor is the property of the Massachusetts Institute of Technology (MIT) and the work licensed under a [Creative Commons Attribution-ShareAlike 3.0 Unported License](#). For more information on App Inventor, go to [MIT App Inventor About Us page](#).