

# App Inventor Beginner Tutorials



MIT Center for Mobile Learning

## **1 Four Simple Tutorials for Getting Started with App Inventor**

1.1	TalkToMe: Your first App Inventor app	4
1.2	TalkToMe Part 2: Shaking and User Input	23
1.3	BallBounce: A simple game app	33
1.4	DigitalDoodle: Drawing App	47



MIT App Inventor  
appinventor.mit.edu

# Four Simple Tutorials for Getting Started with App Inventor



## TalkToMe: Your first App Inventor app

This step-by-step picture tutorial will guide you through making a talking app.

To get started, go to App Inventor on the web.

Go directly to [ai2.appinventor.mit.edu](http://ai2.appinventor.mit.edu), or click the orange "Create" button from the App Inventor website.

The screenshot shows the MIT App Inventor website. At the top, there's a navigation bar with the logo, Home, Blog, Support, and a prominent orange "Create" button. An orange arrow points to the "Create" button. Below the navigation is a social media follow section with links to Facebook, Twitter, YouTube, and Email. To the right is a search bar with "Google Custom Search". The main content area features a large smartphone icon displaying a simple app interface with a text box containing "hello". Above the phone, a script editor shows two blocks of code: one for a button click and another for an accelerometer event. To the left of the phone is a sidebar with component categories like Controls, Logic, Math, Text, Lists, Colors, Variables, Procedures, and a list for "Screen1" which includes TextBox1, Button1, TextToSpeech1, and AccelerometerSensor1. Below the sidebar are "Rename" and "Delete" buttons. To the right of the phone, a dark banner with white text reads "Your ideas. Your designs. Your apps." and "Invent Now". At the bottom, there are three sections: "Get Started" with a green flag icon and "Get Started" button; "Create" with an orange smartphone icon and "Create" button; and "Tutorials" with a purple lightbulb icon and "Tutorials" button.



## Log in to App Inventor with a gmail (or google) user name and password.

Use an existing gmail account or school-based google account to log in to ai2.appinventor.mit.edu  
To set up a brand new gmail account, go to accounts.google.com/SignUp

Google

One account. All of Google.

Sign in with your Google Account

The image shows a Google sign-in interface. It features a large circular profile placeholder. Below it is a text input field containing "appinventorskilz@gmail.com". Underneath is another input field with several dots representing a password. A blue "Sign in" button is positioned below these fields. To the left of the button is a checked checkbox labeled "Stay signed in". To the right is a link "Need help?". At the bottom of the form is a "Create an account" link.

appinventorskilz@gmail.com

.....

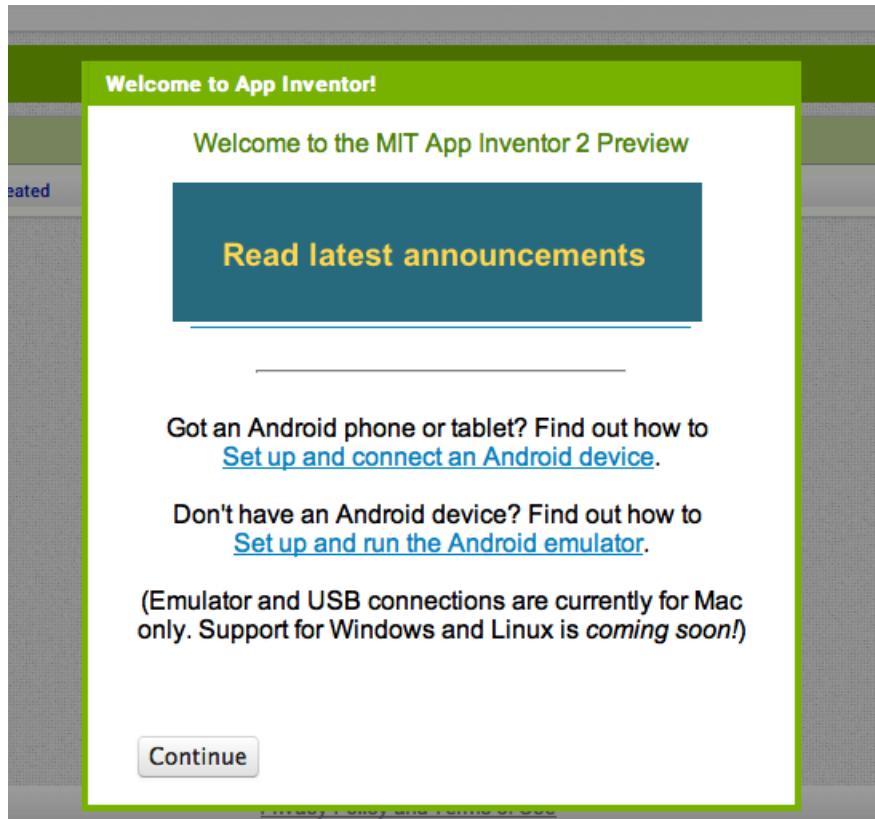
Sign in

Stay signed in      Need help?

Create an account



**Click "Continue" to dismiss the splash screen.**



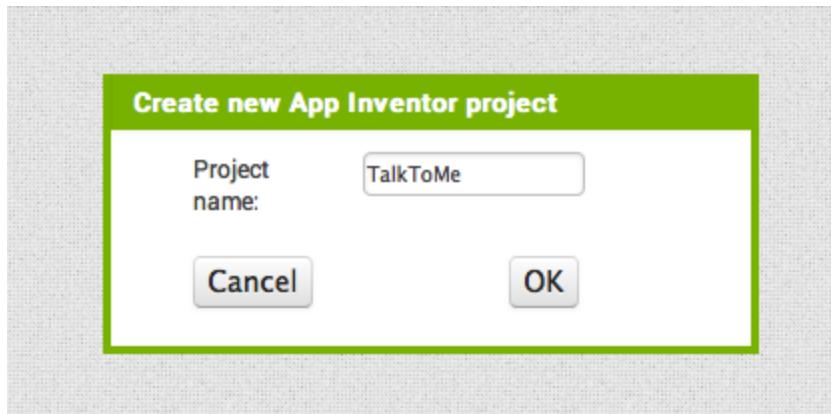


## Start a new project.

The screenshot shows the MIT App Inventor 2 web interface. At the top, there's a toolbar with various icons and the URL 'ai2.appinventor.mit.edu'. Below the toolbar, the main menu includes 'Project', 'Connect', 'Build', 'Help', 'My Projects', 'Guide', 'Report an Issue', and an email link. A red arrow points to the 'New Project ...' button in the 'My Projects' section. The 'Projects' table has columns for 'Name', 'Date Created', and 'Date Modified'. Below the table is a 'Welcome to App Inventor!' message box containing text and an Android icon. At the bottom of the page are links for 'Privacy Policy and Terms of Use'.

## Name the project "TalkToMe" (no spaces!)

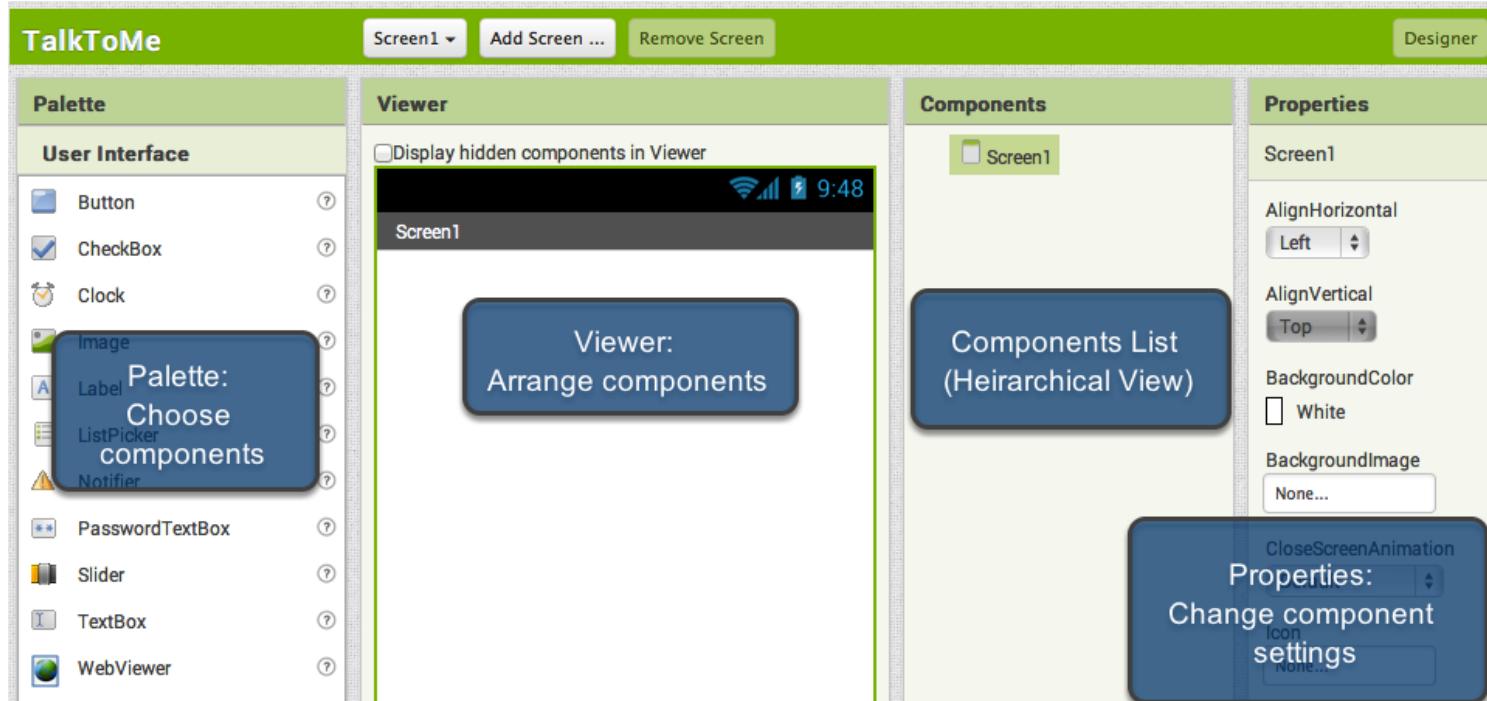
Type in the project name (underscores are allowed, spaces are not) and click OK.





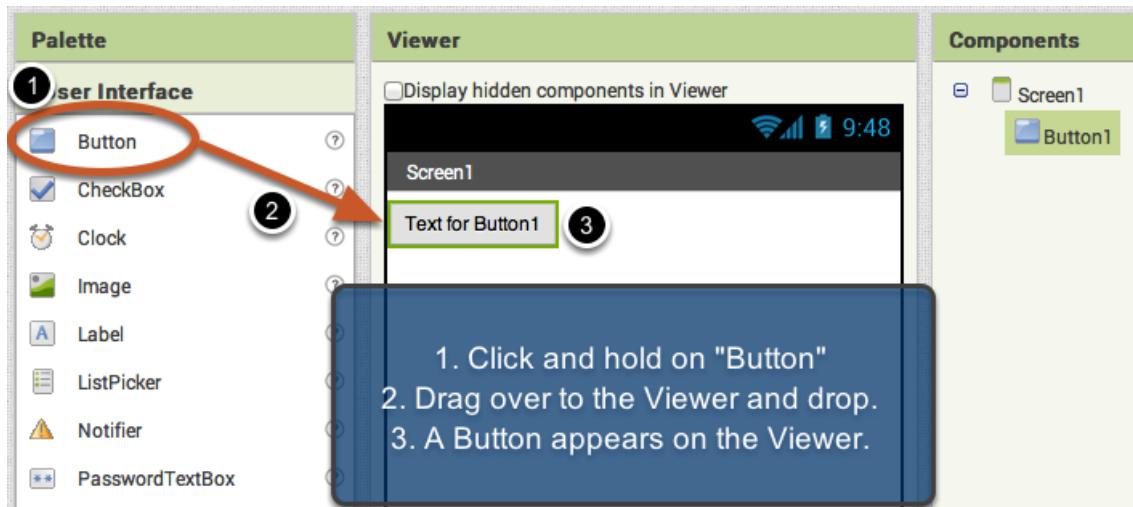
## You are now in the Designer, where you lay out the "user interface" of your app.

The Design Window, or simply "Designer" is where you lay out the look and feel of your app, and specify what functionalities it should have. You choose things for the user interface things like Buttons, Images, and Text boxes, and functionalities like Text-to-Speech, Sensors, and GPS.



## Add a Button

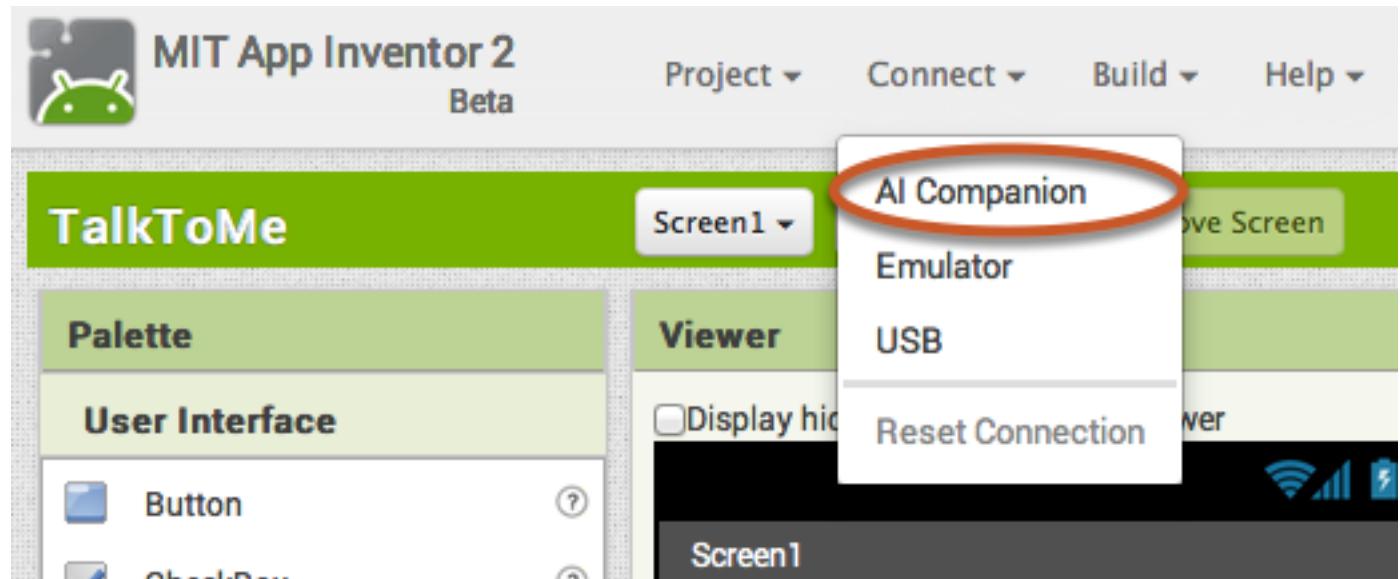
Our project needs a button. **Click and hold** on the word "Button" in the palette. **Drag** your mouse over to the Viewer. **Drop** the button and a new button will appear on the Viewer.





## Connect App Inventor to your phone for live testing

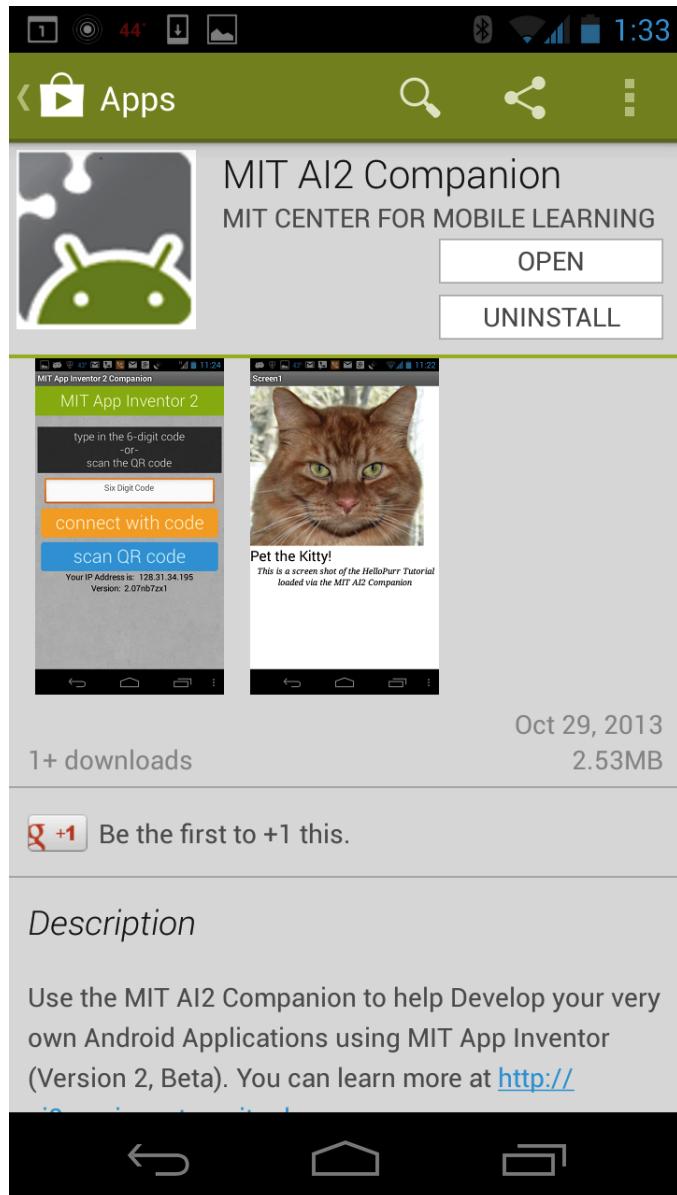
One of the neatest things about App Inventor is that you can see and test your app while you're building it, on a connected device. If you have an **Android phone or tablet**, **follow the steps below**. *If you do not have a device, then follow the instructions for [setting up the on-screen emulator](#) (opens a new page) and then come back to this tutorial once you've gotten the emulator connected to App Inventor.*





**Get the MIT AI2 Companion from the Play Store and install it on your phone or tablet.**

The preferred method for getting the AI2 Companion App is to **download the app from the Play Store by searching for "MIT AI2 Companion".**





## To download the AI2 Companion App to your device directly (SKIP THIS STEP IF YOU already got the app from Play Store)

If for some reason you can not connect to the Google Play store, you can download the AI2 Companion as described here.

First, you will need to go into your phone's settings (#1), choose "Security", then scroll down to allow "Unknown Sources", which allows apps that are not from the Play Store to be installed on the phone.

Second, do one of the following:

A) Scan the QR code above (#2)

or

B) Click the "Need help finding..." link and you'll be taken to the download page. From there you can download the MITAI2Companion.apk file to your computer and then move it over to your device to install it.

SKIP THIS STEP if you already got the AI2 Companion from the Play Store

1

1. Open your phone's settings and click "Security".
2. CHECK the box for "Unknown sources"

2

Scan to download MIT AI2 Companion directly to phone

If you need help...

Connect to Companion

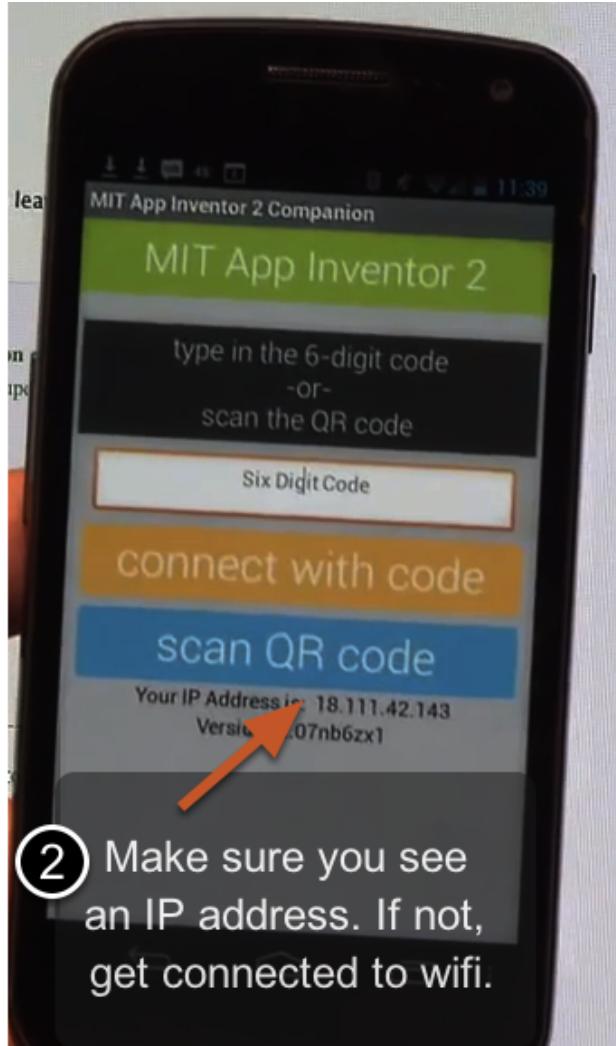
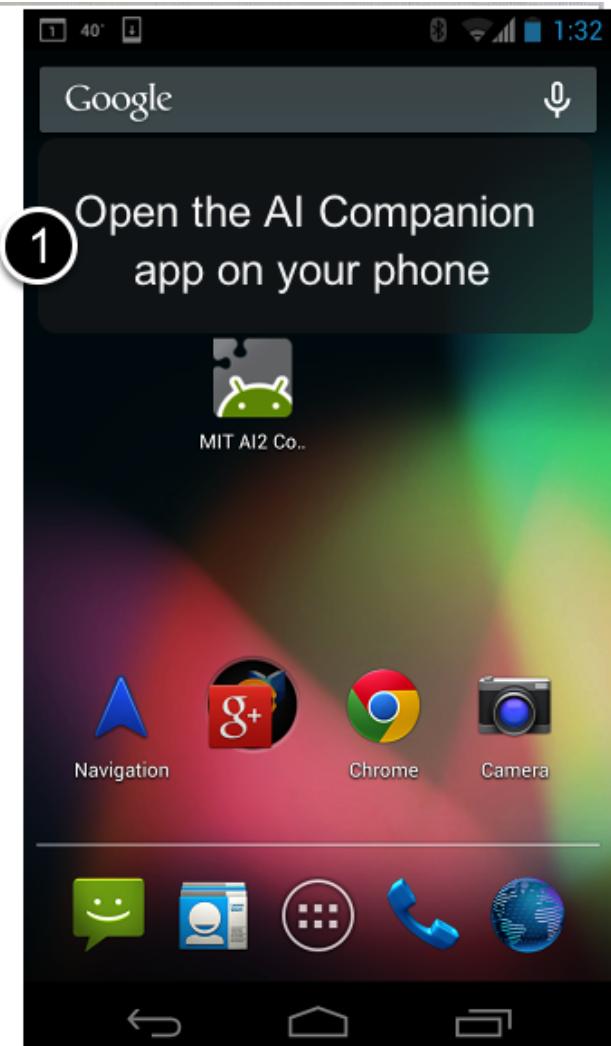
Launch the MIT App Inventor Companion on your device and then scan the barcode or type in the code to connect for live testing of your app.  
[Need help finding the Companion App?](#)

Your code is:



## Start the AICompanion on your device

On your phone or tablet, click the icon for the MIT AI Companion to start the app. **NOTE: Your phone and computer must both be on the same wireless network.** Make sure your phone's wifi is on and that you are connected to the local wireless network. If you can not connect over wifi, go to the Setup Instructions on the App Inventor Website to find out how to connect with a USB cable.





## Get the Connection Code from App Inventor and scan or type it into your Companion app

On the Connect menu, choose "AI Companion". You can connect by:

1 - Scanning the QR code by clicking "Scan QR code" (#1).

or

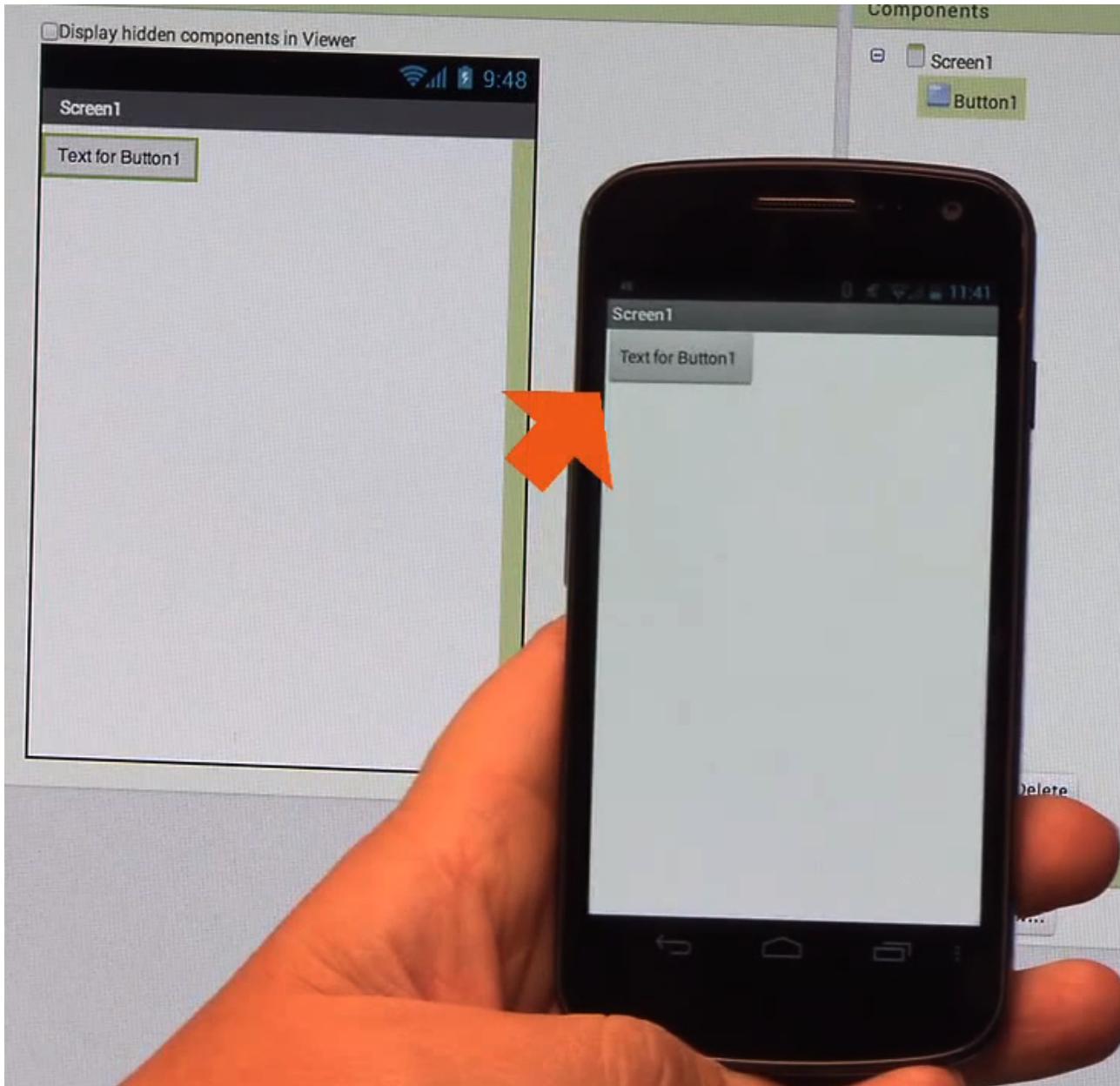
2 - Typing the code into the text window and click "Connect with code" (#2).





## See your app on the connected device

You will know that your connection is successful when you see your app on the connected device. So far our app only has a button, so that is what you will see. As you add more to the project, you will see your app change on your phone.





## Change the Text on the Button

On the properties pane, change the text for the Button. Select the text "Text for Button 1", delete it and type in "Talk To Me". Notice that the text on your app's button changes right away.

The screenshot shows the MIT App Inventor interface with three main panes: Viewer, Components, and Properties.

- Viewer:** Shows a preview of the app's screen titled "Screen1". It contains a green button labeled "Talk To Me".
- Components:** Shows a tree structure with "Screen1" expanded, revealing "Button1".
- Properties:** A detailed pane for "Button1" with the following settings:
  - BackgroundColor: Default (checkbox checked)
  - Enabled: Enabled (checkbox checked)
  - FontBold: (checkbox unchecked)
  - FontItalic: (checkbox unchecked)
  - FontSize: 14.0
  - FontTypeface: default
  - Image: None...
  - Shape: default
  - ShowFeedback: (checkbox checked)
  - Text: Talk To Me
  - TextAlignment: center
  - TextColor: Default

A callout box highlights the "Text" property, which is currently set to "Talk To Me". Another callout box highlights the text input field for the "Text" property, where "Talk To Me" is being typed.



## Add a Text-to-Speech component to your app

Go to the Media drawer and drag out a TextToSpeech component. Drop it onto the Viewer. Notice that it drops down under "Non-visible components" because it is not something that will show up on the app's user interface. It's more like a tool that is available to the app.

The screenshot shows the MIT App Inventor workspace with the following components:

- Palette (left):** Shows the "Media" category selected, with "TextToSpeech" highlighted and circled in red. Other components listed include Camcorder, Camera, ImagePicker, Player, Sound, SoundRecorder, SpeechRecognizer, and VideoPlayer.
- Viewer (center):** Displays a mobile phone screen with "Screen1" and a "Talk To Me" button. A large callout box points to the "Non-visible components" area below the screen, containing the text: "Drop here. Component will automatically show up in Non-visible components area below".
- Components (right):** Shows the component tree with "Screen1", "Button1", and "TextToSpeech1" (which is highlighted and circled in green).
- Properties (far right):** Shows properties for "TextToSpeech1" including "Country" and "Language". Buttons for "Rename" and "Delete" are also present.



## Switch over to the Blocks Editor

It's time to tell your app what to do! Click "Blocks" to move over to the Blocks Editor. Think of the Designer and Blocks buttons like tabs -- you use them to move back and forth between the two areas of App Inventor.

The screenshot shows the MIT App Inventor web-based interface. At the top, there is a navigation bar with links for "My Projects", "Guide", "Report an Issue", and an email address "appinventorskilz@gmail.com". Below the navigation bar is a toolbar with two tabs: "Designer" and "Blocks". The "Blocks" tab is circled with a red oval. To the left of the main workspace is a preview window showing a smartphone screen with a battery icon, signal strength, and the time "9:48". The main workspace is divided into three sections: "Components" (containing "Screen1" and "Button1"), "Properties" (listing "Button1", "BackgroundColor" set to "Default" (checked), "Enabled" (checked), "FontBold" (unchecked), and "FontItalic" (unchecked)), and a large empty area for drawing blocks.



## The Blocks Editor

The Blocks Editor is where you program the behavior of your app. There are Built-in blocks that handle things like math, logic, and text. Below that are the blocks that go with each of the components in your app. *In order to get the blocks for a certain component to show up in the Blocks Editor, you first have to add that component to your app through the Designer.*

The screenshot shows the MIT App Inventor 2 Blocks Editor interface. On the left, there's a sidebar titled "Blocks" with categories like "Built-in" (Control, Logic, Math, Text, Lists, Colors, Variables, Procedures) and "Any component" (Screen1, Button1, TextToSpeech1). Two curly braces on the left side group these categories. In the center, there's a "Viewer" area divided into two sections: "Built-in Blocks" (describing blocks for math, text, logic, and control) and "Component Blocks" (describing blocks for chosen components like Screen1, Button1, and TextToSpeech1). A third brace groups the "Component Blocks" section and the "Viewer" title. On the right, the main workspace is shown with a large callout box containing the text: "Workspace where you assemble the blocks into a program." A trash can icon with an arrow points to a "Trash" section below it, which says "Trash for deleting unneeded blocks." At the bottom of the workspace, there are "Show Warnings" buttons with 0 warnings each, and "Rename" and "Delete" buttons.



## Make a button click event

Click on the Button1 drawer. Click and hold the **when Button1.Click do** block. Drag it over to the workspace and drop it there. This is the block that will handle what happens when the button on your app is clicked. It is called an "Event Handler".

The screenshot shows the MIT App Inventor 2 Beta interface. The top bar includes the logo, title, and navigation links: Project, Connect, Build, Help, My Projects, Guide, and Report.

The main area is titled "TalkToMe" and shows "Screen1" selected. There are tabs for "Add Screen ..." and "Remove Screen".

The left sidebar is the "Blocks" palette, divided into sections:

- Built-in:
  - Control (highlighted with a red oval)
  - Logic
  - Math
  - Text
  - Lists
  - Colors
  - Variables
  - Procedures
- Screen1:
  - Button1 (highlighted with a red oval)
  - Text1
  - Speech1
- Any component

Numbered callouts indicate the steps:

- 1 Circles the "Button1" block in the Screen1 drawer.
- 2 Circles the "when Button1.Click do" block in the Built-in Control section.
- 3 Shows the "when Button1.Click do" block being dragged from the palette to the workspace.

The workspace contains several other blocks, including:

- when Button1.GotFocus do
- when Button1.LongClick do
- when Button1.LostFocus do
- Button1.BackgroundColor
- set Button1.BackgroundColor to
- Button1.Enabled
- set Button1.Enabled to



## Program the TextToSpeech action

Click on the TextToSpeech drawer. Click and hold the **call TextToSpeech1.Speak** block. Drag it over to the workspace and drop it there. This is the block that will make the phone speak. Because it is inside the Button.Click, it will run when the button on your app is clicked.

The screenshot shows the MIT App Inventor 2 interface. The top navigation bar includes 'Project', 'Connect', 'Build', 'Help', 'My Projects', 'Guide', and 'Report an Issue'. The main area is titled 'TalkToMe' with tabs for 'Screen1', 'Add Screen ...', and 'Remove Screen'. On the left, the 'Blocks' panel is open, showing categories like 'Built-in' (Control, Logic, Math, Text, Lists, Colors, Variables, Procedures), 'Screen1' (Button1, TextToSpeech1), and 'Any component'. A red circle labeled '1' highlights the 'TextToSpeech1' block in the 'Screen1' section. In the center, the 'Viewer' pane displays a script with three numbered steps: 1) A 'when TextToSpeech1.AfterSpeaking' event with a 'do' loop containing a 'call TextToSpeech1.Speak' block (circled in orange). 2) A 'when TextToSpeech1.BeforeSpeaking' event with a 'do' loop containing a 'call TextToSpeech1.Speak' block. 3) A 'when Button1.Click' event with a 'do' loop containing a 'call TextToSpeech1.Speak' block. An orange arrow points from step 2 to step 3, indicating the movement of the 'Speak' block from the event loop to the button click event.



## Fill in the message socket on TextToSpeech.Speak Block

Almost done! Now you just need to tell the TextToSpeech.Speak block what to say. To do that, click on the Text drawer, drag out a **text** block and plug it into the socket labeled "message".

The screenshot shows the MIT App Inventor interface. At the top, there's a toolbar with 'Screen1 ▾', 'Add Screen ...', and 'Remove Screen'. Below that is a navigation bar with 'TalkToMe' on the left and 'Blocks' and 'Viewer' tabs on the right. The 'Blocks' tab is active, showing a tree view of categories: Built-in (Control, Logic, Math, Text, Lists, Colors, Variables, Procedures), Screen1 (Button1, TextToSpeech1), and a local list with TextToSpeech1. A red circle highlights the 'Text' category under 'Built-in'. In the 'Viewer' tab, several text blocks are listed: a purple block with a speech bubble icon containing "Hello", a pink block with a person icon containing "join", a pink block with a ruler icon containing "length", a pink block with a question mark icon containing "is empty", a pink block with a comparison operator icon containing "compare texts", and a pink block with a trimmer icon containing "trim". An orange arrow points from the 'Text' category in the blocks tree to the 'Text' block in the viewer. Another orange arrow points from the 'Text' block in the viewer to the 'message' socket of the TextToSpeech1.Speak block in the main workspace.

## Specify what the app should say when the button is clicked

Click on the text block and type in "Congratulations! You've made your first app." (Feel free to use any phrase you like, this is just a suggestion.)

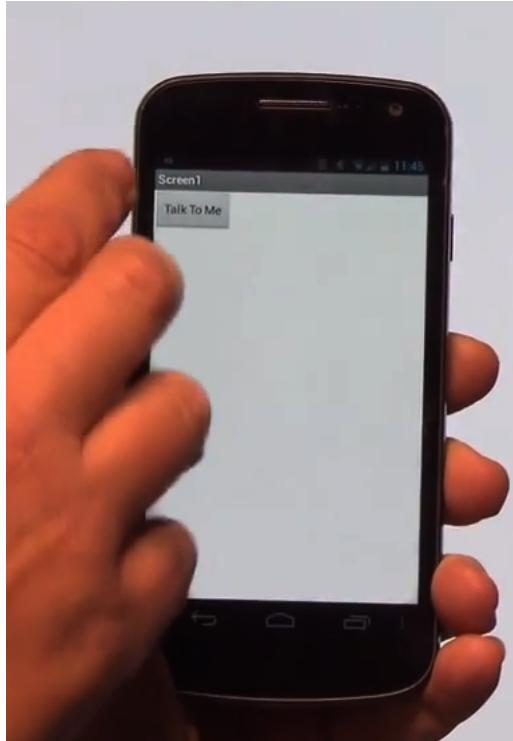
The screenshot shows the completed block code in the workspace:

```
when Button1.Click
do call TextToSpeech1.Speak
    message "Congratulations! You've made your first app."
```



## Now test it out!

Go to your connected device and click the button. Make sure your volume is up! You should hear the phone speak the phrase out loud. (This works even with the emulator.)



## Great job!

Now move on to TalkToMe Part 2 to make the app respond to shaking and to let users put in whatever phrase they want.



## TalkToMe Part 2: Shaking and User Input

This tutorial shows you how to extend the basic TalkToMe app so that it responds to shaking, and so that the user can make the phone say any phrase s/he types in.

### Go to App Inventor on the web and log in.

Go to appinventor.mit.edu and click "Create" or log in directly at ai2.appinventor.mit.edu.

The screenshot shows the MIT App Inventor website's 'Create' interface. At the top, there is a navigation bar with the MIT App Inventor logo, Home, Blog, Support, and a prominent orange 'Create' button. Below the navigation bar, there are social media links for Facebook, Twitter, YouTube, and Email, along with a Google Custom Search bar and a search icon. The main area features a large smartphone icon displaying a simple application interface with a text box containing 'Talk To Me'. To the left of the phone, a sidebar lists various component categories like Controls, Logic, Math, Text, Lists, Colors, Variables, and Procedures, with 'Screen1' expanded to show its components: TextBox1, Button1, TextToSpeech1, and AccelerometerSensor1. Below the sidebar are 'Rename' and 'Delete' buttons. To the right of the phone, a dark overlay contains the text 'Your ideas. Your designs. Your apps.' and a large white 'Invent Now' button. At the bottom, there are three sections: 'Get Started' (with a flag icon and 'Get Started' button), 'Create' (with a smartphone icon and 'Create' button), and 'Tutorials' (with a lightbulb icon and 'Tutorials' button).



## Open the "TalkToMe" project that you worked on in the last tutorial.

App Inventor will always open the last project you worked on, so you may automatically be taken into your TalkToMe app.

The screenshot shows the MIT App Inventor 2 Beta interface. At the top, there's a navigation bar with icons for file operations (New Project, Delete Project), user account (My Projects), help (Guide), and reporting issues (Report an Issue). Below the navigation bar is a green header bar with the text 'New Project ...' and 'Delete Project'. The main area is titled 'Projects' and contains a table with three columns: 'Name', 'Date Created', and 'Date Modified'. There is one project listed: 'TalkToMe' (checkbox checked), created on '2013 Nov 18 14:58:13', and modified on '2013 Nov 19 15:16:37'. A large black cursor arrow points to the 'TalkToMe' project name.

Name	Date Created	Date Modified▼
<input type="checkbox"/> TalkToMe	2013 Nov 18 14:58:13	2013 Nov 19 15:16:37

## Go to the Designer Tab

Your project may open in the Designer. If it does not, click "Designer" in the upper right.

The screenshot shows the top navigation bar of the MIT App Inventor interface. It features two tabs: 'Designer' (which is highlighted in blue) and 'Blocks' (which is greyed out). A large black cursor arrow points to the 'Designer' tab.



## Add an Accelerometer Sensor

In the **Sensors** drawer, drag out an AccelerometerSensor component and drop it onto the Viewer. (It's a non-visible component, so it drops to the bottom of the screen.) NOTE: emulator users should skip this part and proceed to the next section of this tutorial called "Say Anything". (The emulator can not respond to shaking!)

The screenshot shows the MIT App Inventor development environment. On the left is the **Palette**, which contains several categories of components: User Interface, Layout, Media, Drawing and Animation, Sensors, Social, Storage, Connectivity, and LEGO® MINDSTORMS®. The **Sensors** category is highlighted, and the **AccelerometerSensor** component is selected and circled with a red oval. A large orange arrow points from this oval to the **Viewer** window on the right. The **Viewer** window displays a mobile phone screen with the title "Screen1" and a button labeled "Talk To Me". At the bottom of the screen, there is a status bar showing signal strength, battery level, and the time 9:48. The **Components** palette on the far right lists the components currently in the project: Screen1, TextToSpeech1, and AccelerometerSensor1. The AccelerometerSensor1 component is highlighted with a green border and circled with a black oval. A large number "3" is positioned at the bottom right of this highlighted area.



## Go to the Blocks Editor

Click "Blocks" to program the new Accelerometer Sensor that you just added.



## Program the Accelerometer Shaking event

Click the AccelerometerSensor1 drawer to see its blocks. Drag out the **when AccelerometerSensor1.Shaking do** block and drop it on the workspace.

The screenshot shows the MIT App Inventor Blocks Editor. On the left, the 'Blocks' sidebar lists categories like Built-in, Screen1, and AccelerometerSensor1. The AccelerometerSensor1 drawer is circled in red. In the center workspace, there are several blocks:

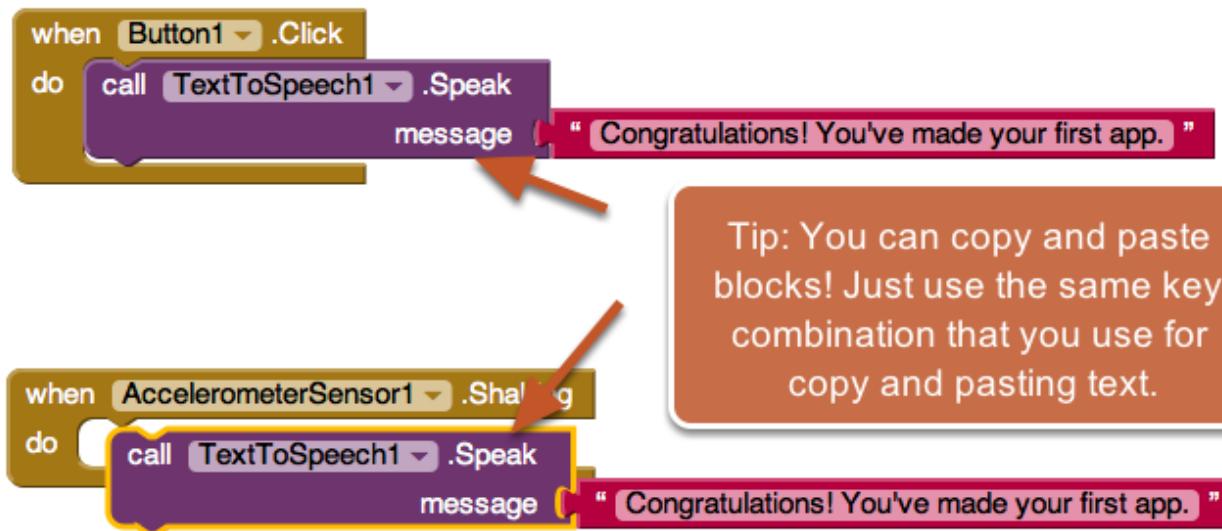
- A yellow 'when AccelerometerSensor1 .AccelerationChanged' block with three variables: xAccel, yAccel, zAccel. It has a 'do' slot followed by a 'call' block to 'TextToSpeech1 .Speak'.
- An orange 'when AccelerometerSensor1 .Shaking do' block, which is also circled in red and has an orange arrow pointing from the sidebar to it.
- Green blocks for 'AccelerometerSensor1 . Available', 'AccelerometerSensor1 . Enabled', 'set AccelerometerSensor1 . Enabled to [ ]', 'AccelerometerSensor1 . MinimumInterval', and 'set AccelerometerSensor1 . MinimumInterval to [ ]'.



## What do we want the app to do when the accelerometer detects shaking?

Copy and paste the blocks that are currently inside the when Button1.Click event handler. You can select the purple block, then hit the key combination on your computer to copy and then to paste. You'll have a second set of blocks to put inside the when Accelerometer.Shaking block.

(Alternatively, you can drag out a new *call TextToSpeech1.Speak* block from the TextToSpeech drawer, and a new pink *text* block from the Text drawer.)



## Change the phrase that is spoken when the phone is shaking.

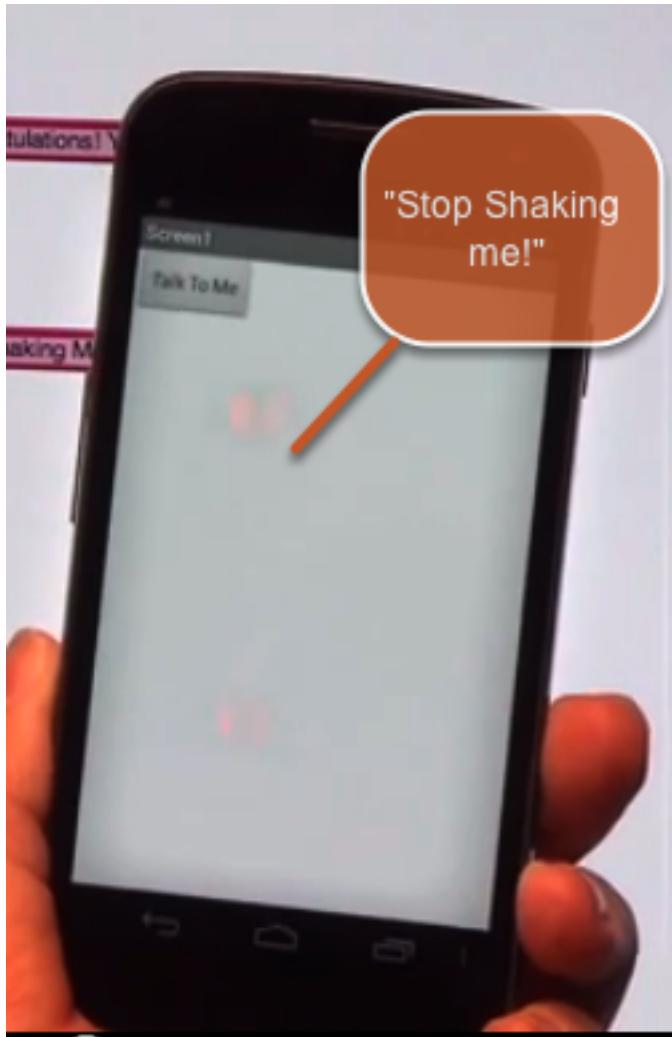
Type in something funny for when the phone responds to shaking.





## Test it out!

You can now shake your phone and it should respond by saying "Stop shaking me!" (or whatever phrase you put in.)



## Say Anything

Is your phone talking to you? Cool! Now let's program the button click so that it causes the phone to speak whatever phrase the user put into the text box. Go back to the Designer.





## Add a Text Box to your user interface.

From the User Interface drawer, drag out a TextBox and put it above the Button that is already on the screen.

The screenshot shows the MIT App Inventor Designer interface. On the left is the **Palette**, which contains the **User Interface** section with various components: Button, CheckBox, Clock, Image, Label, ListPicker, Notifier, PasswordTextBox, Slider, TextBox (which is highlighted with a red oval), and WebViewer. Below this is the **Layout** section. In the center is the **Viewer**, showing a mobile phone screen with the title "Screen1". It displays a white rectangular area (the TextBox) positioned above a blue button labeled "Talk To Me". A red arrow points from the "TextBox" entry in the Palette towards the white rectangle in the Viewer. On the right is the **Components** panel, which lists the components used in the current screen: Screen1, TextBox1, Button1, TextToSpeech1, and AccelerometerSensor1.

## Back to the Blocks Editor!





## Get the text that is typed into the TextBox.

Get the text property of the TextBox1. The green blocks in the TextBox1 drawer are the "getters" and "setters" for the TextBox1 component. You want your app to speak out loud whatever is currently in the TextBox1 Text property (i.e. whatever is typed into the text box). Drag out the **TextBox1.Text** getter block.

The screenshot shows the MIT App Inventor interface. On the left, the **Blocks** palette is open, displaying categories like Built-in, Screen1, and Any component. Under **Screen1**, the **TextBox1** component is selected, highlighted with a green background. In the center, the **Viewer** pane displays a list of green code blocks for the TextBox1 component. One block, **TextBox1.Text**, is circled in red and has a large orange arrow pointing from it towards the right edge of the screen, where two yellow rectangular blocks labeled "when do" are visible.

Block Type	Description
Setter	set TextBox1 . Height to [value]
Setter	set TextBox1 . Hint to [value]
Setter	set TextBox1 . MultiLine to [value]
Setter	set TextBox1 . NumbersOnly to [value]
Getter (Targeted)	TextBox1 . Text
Setter	set TextBox1 . Text to [value]
Setter	set TextBox1 . TextColor to [value]



## Set the Button Click event to speak the text that is in the Text Box.

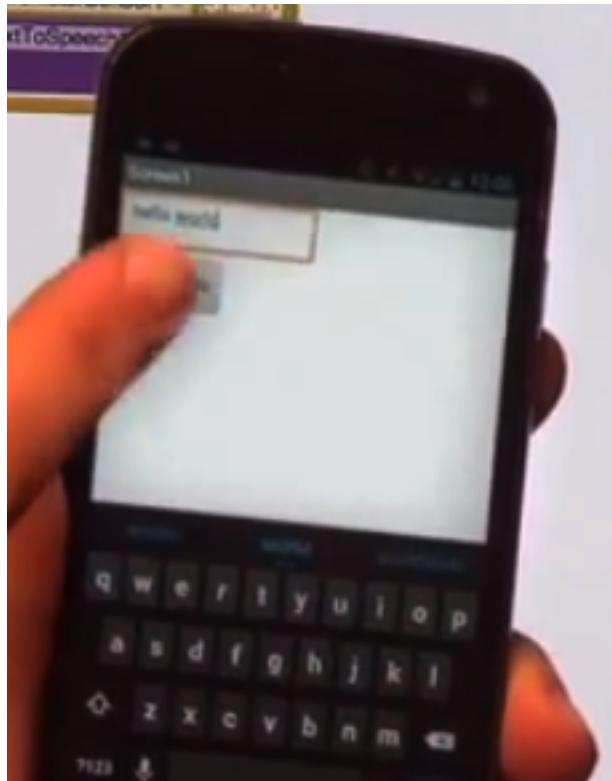
Pull out the "congratulations..." text box and plug in the TextBox1.Text block. You can throw the pink text block away by dragging it to the trash in the lower right corner of the workspace.



## Test your app!

Now your app has two behaviors: When the button is clicked, it will speak out loud whatever words are currently in the Text Box on the screen. (if nothing is there, it will say nothing.)

The app will also say "Stop Shaking Me" when the phone is shaken.





## Congrats! You've built a real app!

Give some thought to what else this app could do. Here are some ideas for extensions:

- Random phrase generator
- Mad Libs - player chooses noun, verb, adjective, adverb, person and it picks one from a list that you program.
- Magic 8 Ball App
- Name picker - useful for teachers to call on a student

You could also play around with Speech-To-Text. Have fun!



## BallBounce: A simple game app

In this tutorial, you will learn about animation in App Inventor by making a Ball (a sprite) bounce around on the screen (on a Canvas).

### Start a New Project

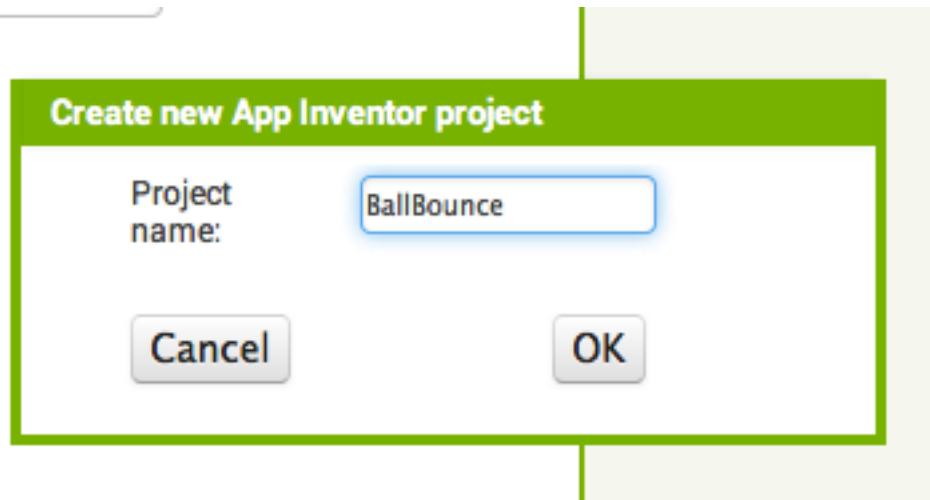
If you have another project open, go to My Projects menu and choose New Project.

The screenshot shows the MIT App Inventor 2 Beta interface. At the top, there is a navigation bar with icons for file, connect, build, and help, followed by the text "MIT App Inventor 2 Beta". Below the navigation bar is a project title "TalkToMe" and a palette section titled "User Interface" containing various UI components like Button, CheckBox, Clock, Image, Label, ListPicker, Notifier, and PasswordTextBox. To the right of the palette is a workspace with a single screen named "Screen1" containing a button labeled "Talk To Me". Overlaid on the workspace is a context menu for "My Projects" with the following options: "My Projects", "New ...", "Import ...", "Delete", "Save", "Save As...", "Checkpoint ...", "Export", "Export all", "Import Keystore", "Export Keystore", and "Delete Keystore". The option "New ..." is highlighted with a red circle.



## Name the Project

Call it something like "BallBounce". Remember, no spaces. But underscores are OK.



## Add a Canvas

From the **Drawing and Animation** drawer, drag out a **Canvas component** and drop it onto the viewer.

The screenshot shows the MIT App Inventor interface with the following components:

- Palette:** On the left, under the 'User Interface' section, the 'Drawing and Animation' category is highlighted with a red oval. Inside this category are three components: 'Ball', 'Canvas', and 'ImageSprite'. A large red arrow points from this category towards the 'Viewer' window.
- Viewer:** In the center, the 'Viewer' window displays a simulated Android screen labeled 'Screen1'. At the top of the screen is a status bar showing signal strength, battery level, and the time '9:48'. Below the status bar is a navigation bar with a back arrow and a home button. The main area of the screen contains a small icon representing a canvas component.
- Components:** On the right, the 'Components' pane shows a tree structure with 'Screen1' expanded, revealing 'Canvas1' as a child component.



## Set the Screen so that it does not scroll

The default setting for App Inventor is that the screen of your app will be "scrollable", which means that the user interface can go beyond the limit of the screen and the user can scroll down by swiping their finger (like scrolling on a web page). When you are using a Canvas, you have to **turn off the "Scrollable" setting** (UNCHECK THE BOX) so that the screen does not scroll. This will allow you to make the Canvas to fill up the whole screen.

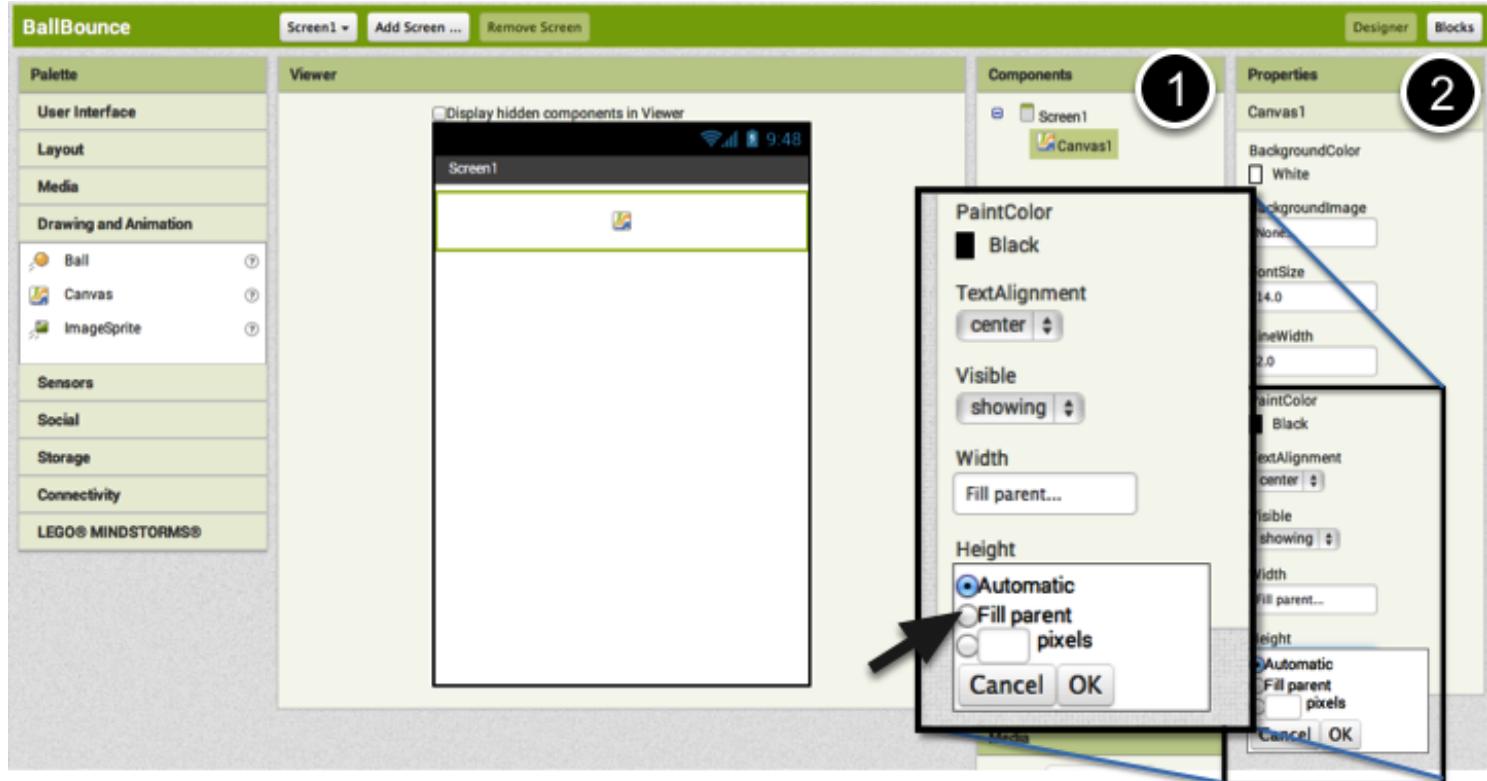
The screenshot shows the MIT App Inventor Designer interface with the following components:

- Palette:** On the left, under the **User Interface** category, are the **Ball**, **Canvas**, and **ImageSprite** components.
- Viewer:** In the center, it displays a preview of the app's screen titled "Screen1". It shows a black ball at the top center and a small blue square icon near the bottom center.
- Components:** On the right, the "Components" panel lists "Screen1" which contains "Canvas1" and "Ball1".
- Properties:** The "Properties" panel shows settings for "Screen1":
  - AlignHorizontal: Left
  - AlignVertical: Top
  - BackgroundColor: White
  - BackgroundImage: None...
  - CloseScreenAnimation: Default
- Callout Box:** A callout box highlights the "Scrollable" property of the "Screen1" component. The property is currently checked (indicated by a blue checkmark). The "Title" field shows "Screen1".



## Change the Height and Width of the Canvas to Fill Parent

Make sure the Canvas component is selected (#1) so that its properties show up in the Properties Pane (#2). Down at the bottom, **set the Height property to "Fill Parent"**. Do the **same with the Width property**.





## Add a Ball

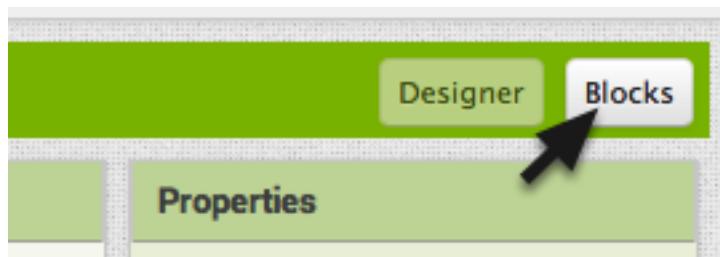
Now that we have a Canvas in place, we can add a Ball Sprite. This can also be found in the **Drawing and Animation drawer**. Drag out a **Ball component** and drop it onto the Canvas (#1). If you'd like the ball to show up better, you can change its **Radius** property in the Properties pane (#2).

The screenshot shows the MIT App Inventor Designer interface with four main panes:

- Palette:** On the left, under the **Drawing and Animation** category, the **Ball** component is highlighted with a red oval and circled with a black number 1.
- Viewer:** In the center, a mobile phone screen labeled "Screen1" displays a black ball sprite on the canvas.
- Components:** On the right, the **Components** tree shows **Screen1**, **Canvas1**, and **Ball1**.
- Properties:** The properties for **Ball1** are listed:
  - Enabled:** checked
  - Heading:** 0
  - Interval:** 100
  - PaintColor:** Black
  - Radius:** 15 (highlighted with a red oval and circled with a black number 2)
  - Speed:** 0.0
  - Visible:** checked
  - X:** 48
  - Y:** 11
  - Z:** 1.0

An orange arrow points from the circled "Ball" component in the palette to the ball sprite on the screen.

## Open the Blocks Editor.





Open the Ball1 Drawer to view the Ball's blocks.

**BallBounce**      Screen1 ▾    Add Screen ...    Remove Screen

**Blocks**

- ▀ Built-in
  - █ Control
  - █ Logic
  - █ Math
  - █ Text
  - █ Lists
  - █ Colors
  - █ Variables
  - █ Procedures
- ▀ Screen1
- ▀ **Ball1**  Ball1
- ▀ Any component

**Viewer**

```
when Ball1.CollidedWith
other
do [redacted]

when Ball1.Dragged
startX startY prevX prevY currentX currentY
do [redacted]

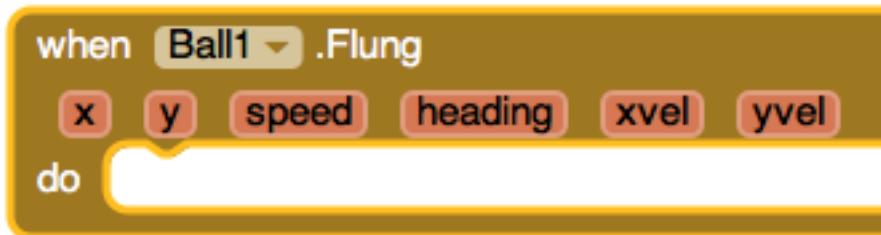
when Ball1.EdgeReached
edge
do [redacted]

when Ball1.Flung
x y speed heading xvel yvel
do [redacted]

when Ball1.NoLongerCollidingWith
```

Drag out the Flung Event Handler

Choose the block **when Ball1.Flung** and drag-and-drop it onto the workspace. Flung refers to the user making a "Fling gesture" with his/her finger to "fling" the ball. Fling is a gesture like what a golf club does, not like how you launch Angry Birds! In App Inventor, the event handler for that type of gesture is called *when Flung*.





## Set the Ball's Heading and Speed. First get the setter blocks.

Open the Ball drawer and scroll down in the list of blocks to get the **set Ball1.Heading** and **set Ball1.Speed** blocks

The screenshot shows the MIT App Inventor interface. On the left, the 'Blocks' palette is open, displaying categories like 'Built-in' (Control, Logic, Math, Text, Lists, Colors, Variables, Procedures), 'Screen1', 'Canvas1' (with 'Ball1' highlighted and circled in red), and 'Any component'. At the bottom of the palette are 'Rename' and 'Delete' buttons. The right side shows the 'Viewer' pane where various blocks are listed. Two specific blocks are circled in red: 'set Ball1 . Heading to [ ]' and 'set Ball1 . Speed to [ ]'. A callout box with a brown background and white text points to these blocks with the instruction: 'Scroll down to the green "setter" blocks for the Ball's Heading and Speed'. A large red arrow points downwards from the callout box towards the bottom of the viewer pane, indicating where to scroll.

Blocks

- Built-in
  - Control
  - Logic
  - Math
  - Text
  - Lists
  - Colors
  - Variables
  - Procedures
- Screen1
- Canvas1
  - Ball1
- Any component

Rename Delete

Viewer

- set Ball1 . Enabled to [ ]
- Ball1 . Heading
- set Ball1 . Heading to [ ]
- Ball1 . Interval
- set Ball1 . Interval to [ ]
- Ball1 . PaintColor
- set Ball1 . PaintColor to [ ]
- Ball1 . Radius
- set Ball1 . Radius to [ ]
- Ball1 . Speed
- set Ball1 . Speed to [ ]
- Ball1 . Visible

Scroll down to the green "setter" blocks for the Ball's Heading and Speed

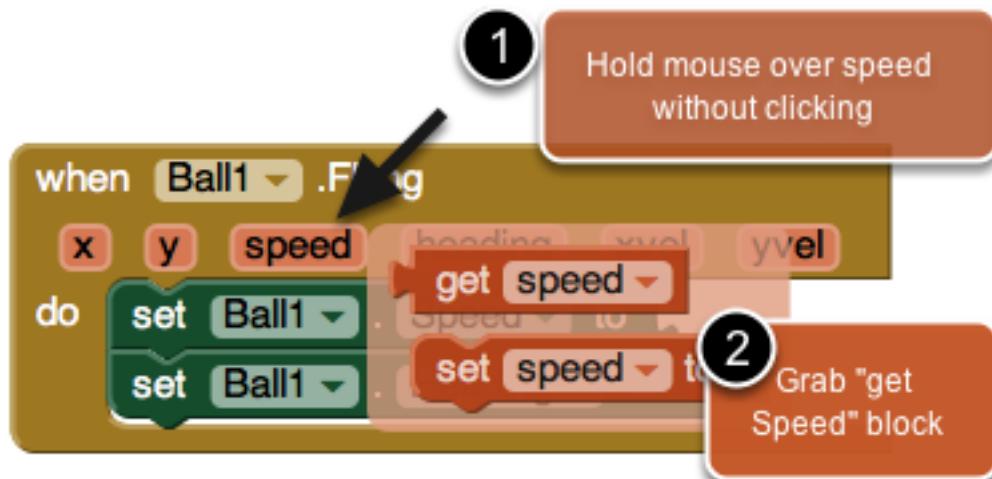


## Plug the set Ball1.Speed and set Ball1.Heading into the Fling event handler



## Set the Ball's speed to be the same as the Fling gesture's speed

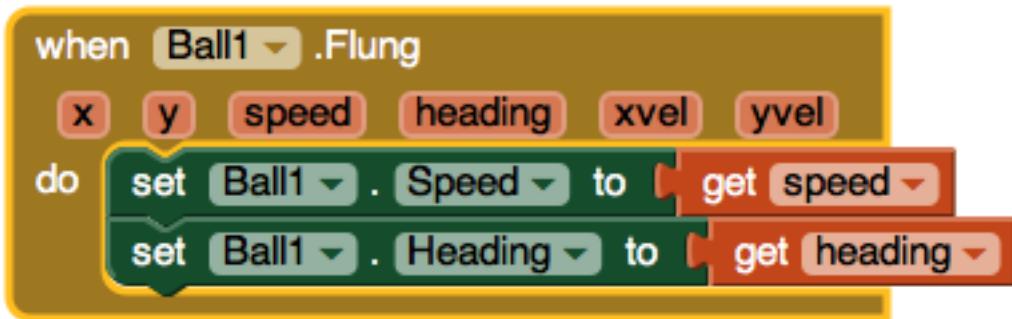
Mouse over the "speed" parameter of the **when Ball1.Flung** event handler. The get and set blocks for the speed of the fling will pop up. Grab the **get speed** block and plug that into the **set Ball1.Speed** block.





## Set the Ball's heading to be the same as the Fling gesture's heading

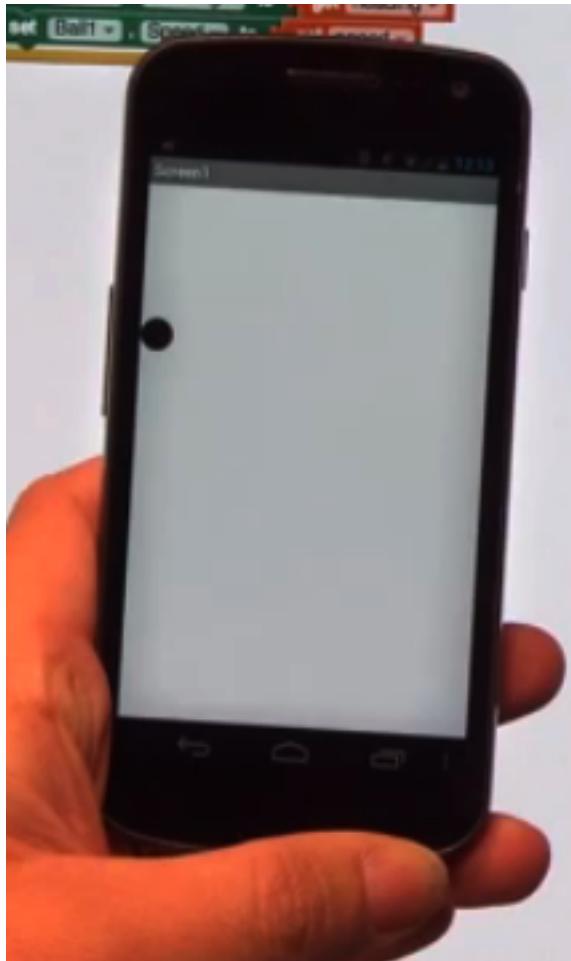
Do the same for the Ball's heading. Mouse over the **heading** parameter and you'll see the **get heading** block appear. Grab that block, and click it into the **set Ball1.Heading** block.





## Test it out

A good habit while building apps is to test while you build. App Inventor lets you do this easily because you can have a live connection between your phone (or emulator) and the App Inventor development environment. If you don't have a phone (or emulator) connected, go to the connection instructions and then come back to this tutorial. (Connection instructions are in Tutorial #1 or on the website under "Getting Started".)



## Why does the Ball get stuck on the side of the screen?!

After flinging your ball across the screen, you probably noticed that it got stuck on the side. This is because the ball's heading has not changed even though it hit the side of the canvas. To make the ball "bounce" off the edge of the screen, we can program in a new event handler called "When Edge Reached".



## Add an Edge Reached Event

Go into the Ball1 drawer and pull out a **when Ball1.EdgeReached do** event.

**Blocks**

- Built-in
  - Control
  - Logic
  - Math
  - Text
  - Lists
  - Colors
  - Variables
  - Procedures
- Screen1
- + Any component
  - Ball1

**Viewer**

- when Ball1 .CollidedWith other do
- when Ball1 .Dragged startX startY prevX prevY currentX currentY do
- when Ball1 .EdgeReached edge do
- when Ball1 .Moving x y speed heading xvel yvel do
- when Ball1 .NoLongerCollidingWith other do

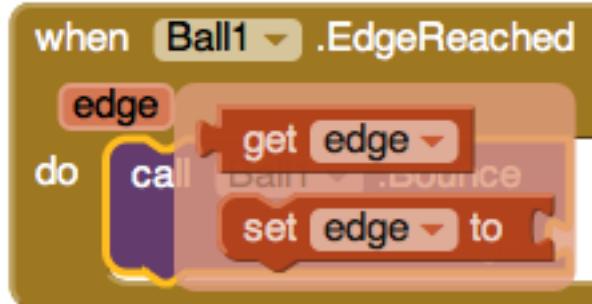


Go back into the Ball1 drawer and pull out a Ball.Bounce block.

The screenshot shows the MIT App Inventor interface. On the left, the 'Blocks' palette is open, displaying categories like Built-in (Control, Logic, Math, Text, Lists, Colors, Variables, Procedures), Screen1, Canvas1 (with Ball1 selected), and Any component. In the center, the 'Viewer' workspace shows a script for the Ball1 component. It includes event blocks for TouchDown, TouchUp, and Touched, along with several call blocks. One call block, 'call Ball1 .Bounce edge', is highlighted with a red oval. An orange arrow points from this block to a 'when Ball1 .EdgeReached' event block on the right side of the workspace.

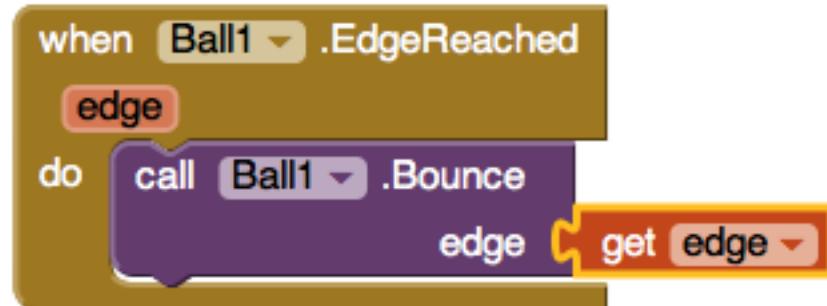
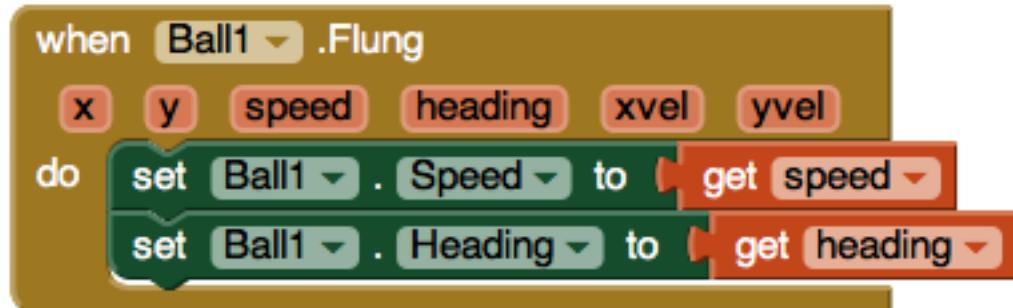
Add the edge value for the Ball.Bounce block

The **Ball.Bounce** method needs an edge argument. Notice that the **Ball1.EdgeReached** event has an "edge" as a parameter. We can take the **get edge** block from that argument and plug it into the call **Ball1.Bounce** method. Grab the **get edge** block by mousing over (hover your mouse pointer over) the "edge" parameter on the **when Ball1.EdgeReached** block.





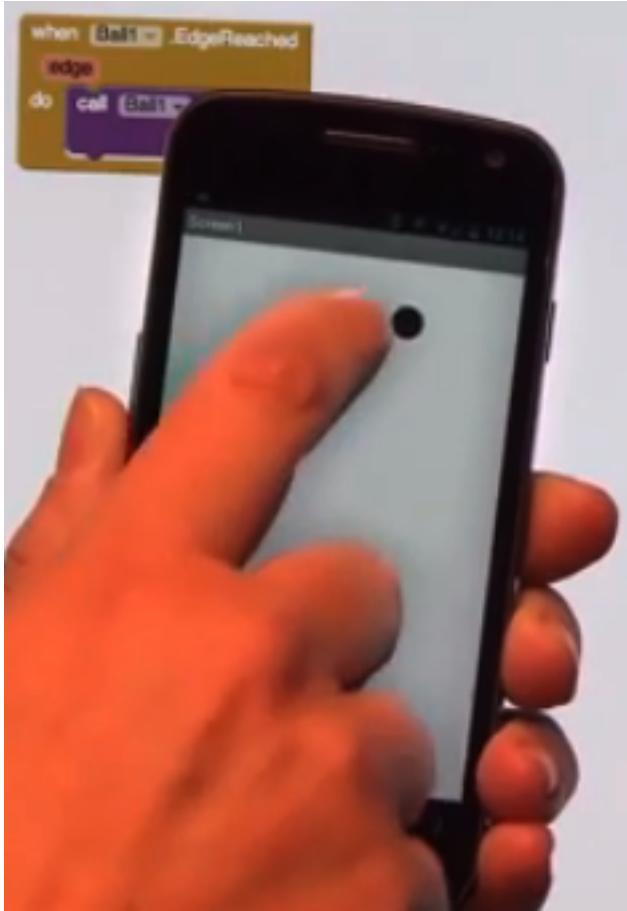
Your final blocks should look like this. Now test it out!





## Test it out!

Now, when you fling the ball, it should bounce off the edges of the canvas. Great job!



## There are many ways to extend this app.

Here are some ideas... but the possibilities are endless!

- Change the color of the ball based on how fast it is moving or which edge it reaches.
- Scale the speed of the ball so that it slows down and stops after it gets flung.
- Give the ball obstacles or targets to hit
- Introduce a paddle for intercepting the ball, like a Pong game

Visit the [App Inventor website](#) to find tutorials that help you extend this app, particularly the [Mini Golf tutorial](#).

Have fun with these extensions, or others that you think up!

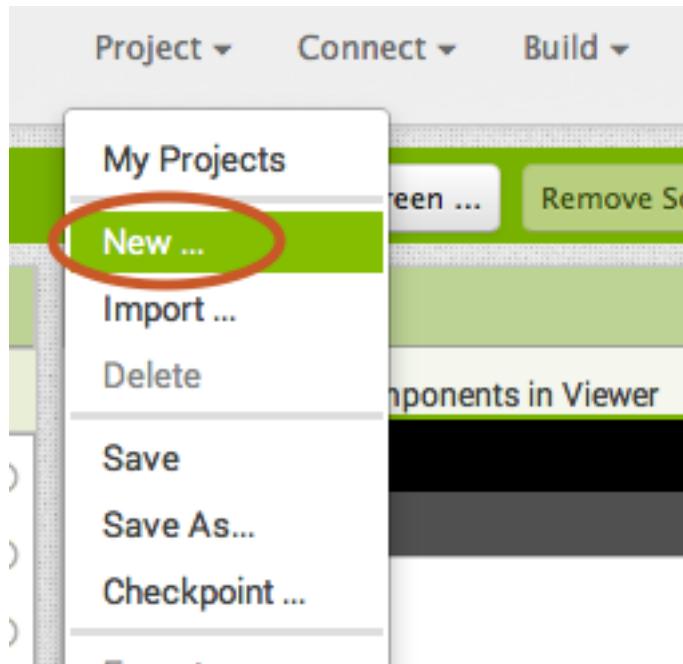


## DigitalDoodle: Drawing App

This tutorial will show you how to draw a line on the screen as the user drags a finger around.

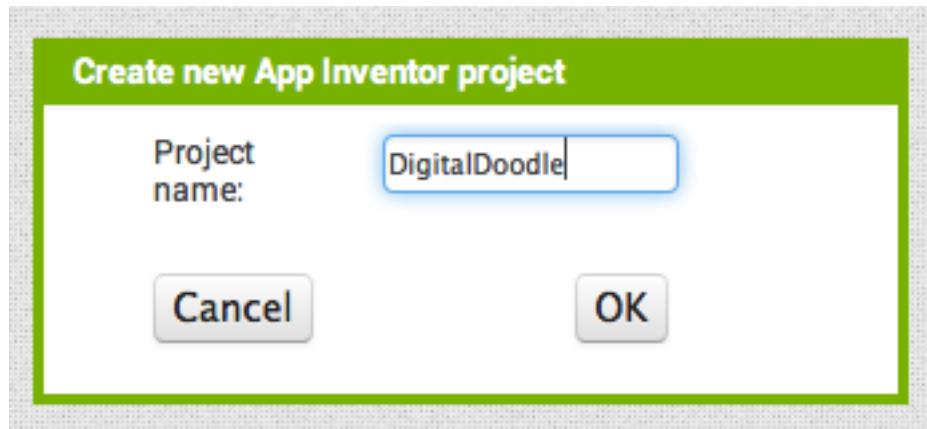
### Start a New Project

From the My Projects page, click New Project. If you have another project open, go to My Projects menu and choose New Project.



### Name the Project

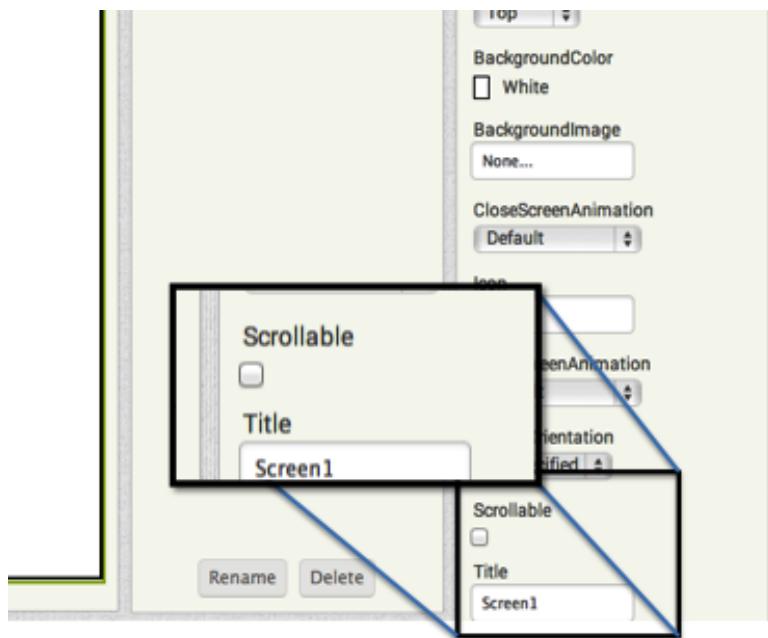
Call this project DigitalDoodle, or create your own name for this drawing app.





## Set the Screen so that it does not scroll

The default setting for App Inventor is that the screen of your app will be "scrollable", which means that the user interface can go beyond the limit of the screen and the user can scroll down by swiping their finger (like scrolling on a web page). When you are using a Canvas, you have to **turn off the "Scrollable" setting (UNCHECK THE BOX)** so that the screen does not scroll. This will allow you to make the Canvas to fill up the whole screen.





## Add a Canvas

From the Drawing and Animation drawer, drag out a Canvas component.

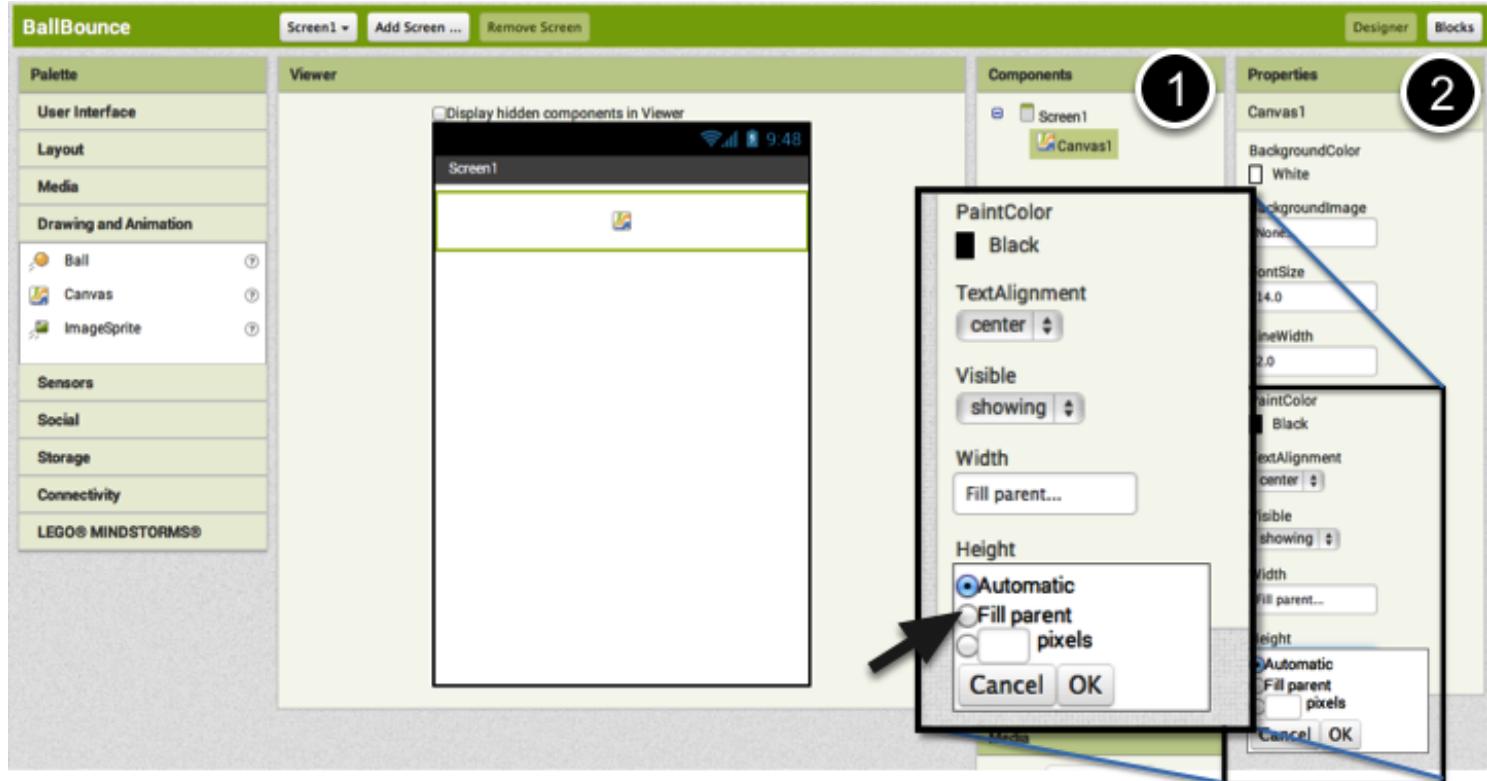
The screenshot shows the MIT App Inventor workspace for the project "DigitalDoodle". The interface is divided into three main sections: a "Palette" on the left, a "Viewer" in the center, and a "Components" tree on the right.

- Palette:** On the left, under the "Drawing and Animation" category, the "Canvas" component is highlighted with a green box and has a red arrow pointing to it from the text above. Other components listed include Ball, ImageSprite, Sensors, Social, Storage, Connectivity, and LEGO® MINDSTORMS®.
- Viewer:** In the center, the "Screen1" screen is displayed. It shows a black mobile phone interface with a "Screen1" title bar. A small icon representing the "Canvas" component is placed on the screen.
- Components:** On the right, the "Components" tree shows a single node: "Screen1" containing a "Canvas1" component.



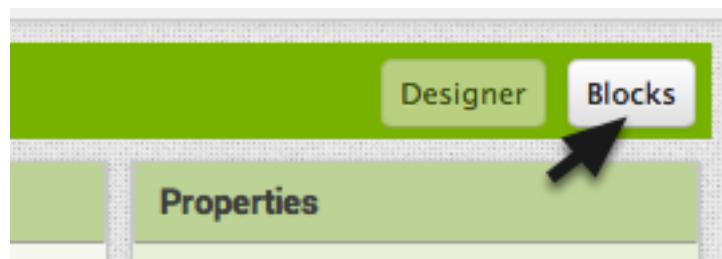
## Change the Height and Width of the Canvas to Fill Parent

Make sure the Canvas component is selected (#1) so that its properties show up in the Properties Pane (#2). Down at the bottom, **set the Height property to "Fill Parent"**. Do the **same with the Width property**.



## That's all for the Designer! Go over to the Blocks.

Believe it or not, for now this app only needs a Canvas. Go into the Blocks Editor to program the app.





## Get a Canvas.Dragged event block

In the Canvas1 drawer, pull out the **when Canvas1.Dragged** event.

The screenshot shows the MIT App Inventor interface. The top bar includes the title "DigitalDoodle", a screen dropdown set to "Screen1", a "Add Screen ..." button, and a "Remove Screen" button. The left sidebar, titled "Blocks", lists categories like Built-in (Control, Logic, Math, Text, Lists, Colors, Variables, Procedures), Screen1 (Canvas1), and Any component. The "Canvas1" block is highlighted with a red oval. The main area, titled "Viewer", displays three event blocks:

- when Canvas1 .Dragged**: This block has parameters for startX, startY, prevX, prevY, currentX, currentY, and draggedSprite. It is circled with a red oval.
- when Canvas1 .Flung**: This block has parameters for x, y, speed, heading, xlabel, ylabel, and flungSprite.
- when Canvas1 .TouchDown**: This block has parameters for x and y.



## Get a Canvas.DrawLine call block

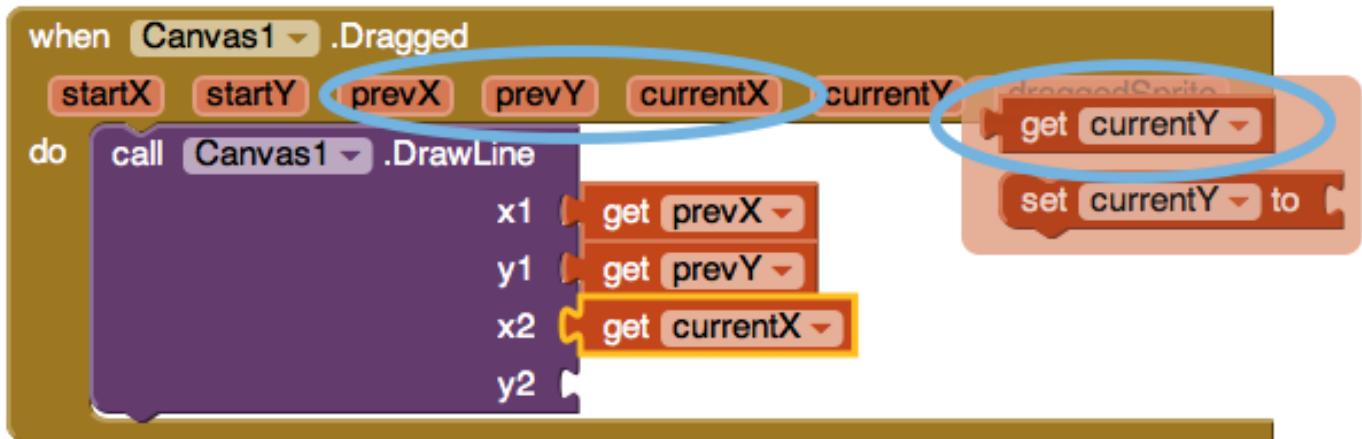
In the Canvas1 drawer, pull out the **when Canvas1.DrawLine** method block..

The screenshot shows the MIT App Inventor interface. The left pane, titled 'Blocks', lists various categories: Built-in (Control, Logic, Math, Text, Lists, Colors, Variables, Procedures), Screen1, and Canvas1. The 'Canvas1' category is highlighted with a red oval. The right pane, titled 'Viewer', shows a sequence of blocks: a 'when' block (when Canvas1.Touched) followed by a 'do' block (yellow bar), then a 'call' block (Canvas1.Clear), another 'call' block (Canvas1.DrawCircle), and finally a 'call' block (Canvas1.DrawLine). The 'Canvas1.DrawLine' block is circled in red.



**Use the get and set blocks from the Dragged block to fill in the values for the Draw Line block.**

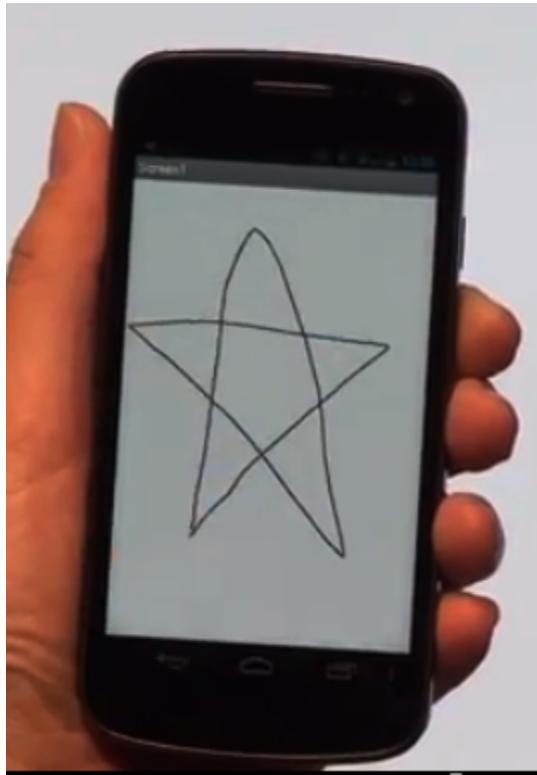
The Canvas Dragged event will happen over and over again very rapidly while the user drags a finger on the screen. Each time that Dragged event block is called, it will draw a small line between the previous location (prevX, prevY) of the finger to the new location (currentX, currentY). Mouse over the parameters of the Canvas1.Dragged block to pull out the get blocks that you need. (Mouse over them, don't click on them.)





## Test it out!

Go to your connected device and drag your finger around the screen. Do you see a line?



## Great work! Now extend this app

Here are some ideas for extending this app. You can probably think of many more!

- Change the color of the ink (and let the user pick from a selection of colors). See [Paint Pot tutorial](#).
- Change the background to a photograph or picture.
- Let the user draw dots as well as lines (hint: Use DrawCircle block).
- Add a button that turns on the camera and lets the user take a picture and then doodle on it.