RYERSON UNIVERSITY

Introduction to App Development: Lesson 1

Introduction

What is this course?

- Our intention here is to teach you how to code.
  - More accurately, we want to invite you to see code at work, to understand how a machine looks at the world.

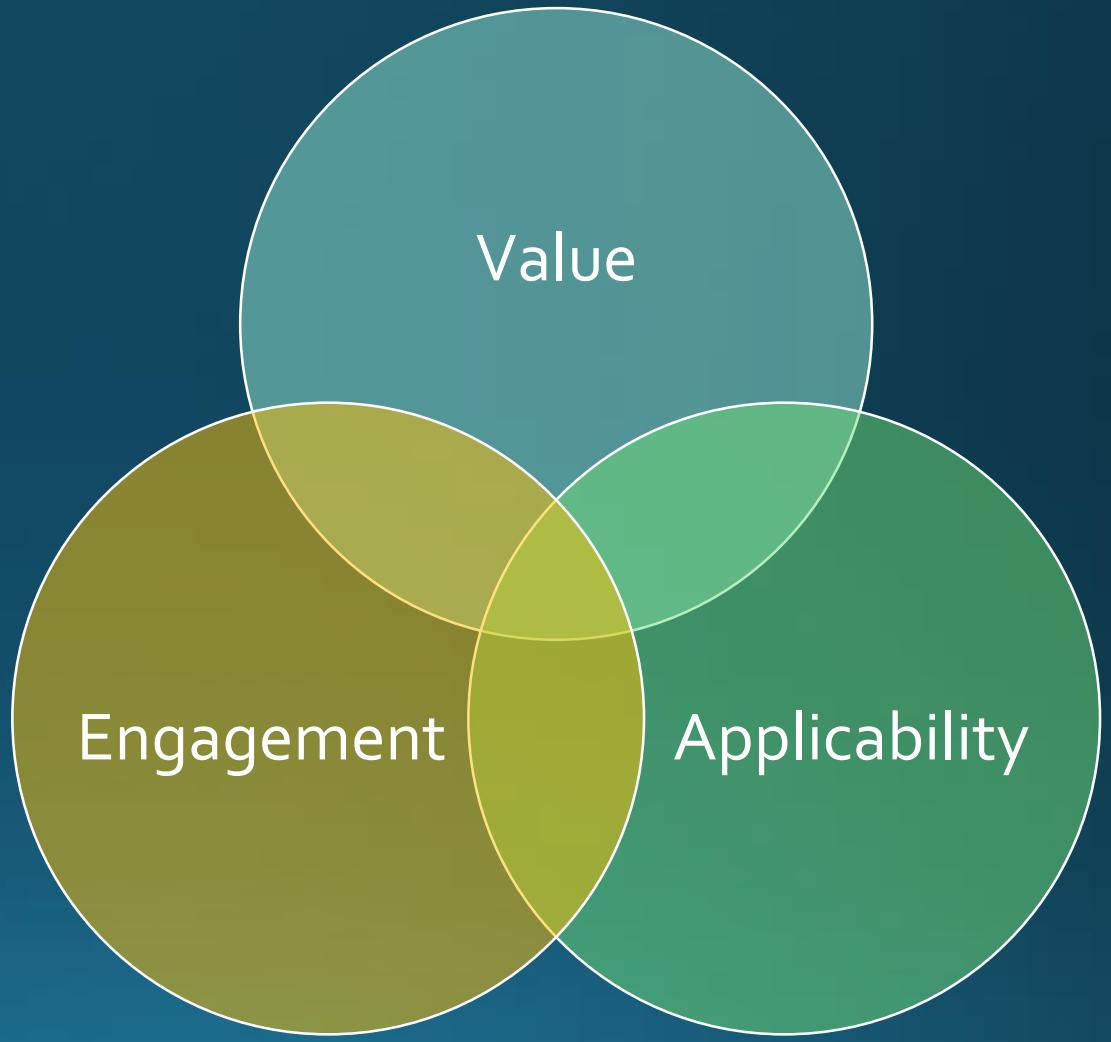
- Computational thinking is useful in a long list of new fields
  - Virtual Reality, Smartphones, Internet, Microcontrollers, Data Cloud, etc.

- Programming is the new literacy.
  - "I have a dream that all University graduates can code"

Further reading: http://www.chris-granger.com/2015/01/26/coding-is-not-the-new-literacy/

Our goals.

- Value:
  - Core computer science concepts.
  - Essentials of coding
- Engagement:
  - Approachable: Beginner oriented
  - Creativity focused
- Applicability:
  - The skills taught can be immediately applied use computer skills in your own field.



Some caveats:

- This course covers a wide range of topics
  - While the first few lectures may seem simple, as we layer topics on top of each other, we will be able to build some impressive apps as we go along.
    - Last month, we made an app to track a space station
- This course is incremental
  - It is important to attend lectures or catch up with missed content.
  - Each lecture builds on previous material
- This course will challenge you.
  - Questions will be asked that you may not know the answer to.

Why you shouldn't worry:

- Each topic is carefully explained.
- You are given plenty of room to explore if you are ahead.

- Less than 10:1 instructor ratio to help you if you are stuck.
- Slides are available online.
- Email us if you are behind. We're willing to help.

- Answering questions you don't know, even if you are wrong, is a great way to learn.

Our approach:

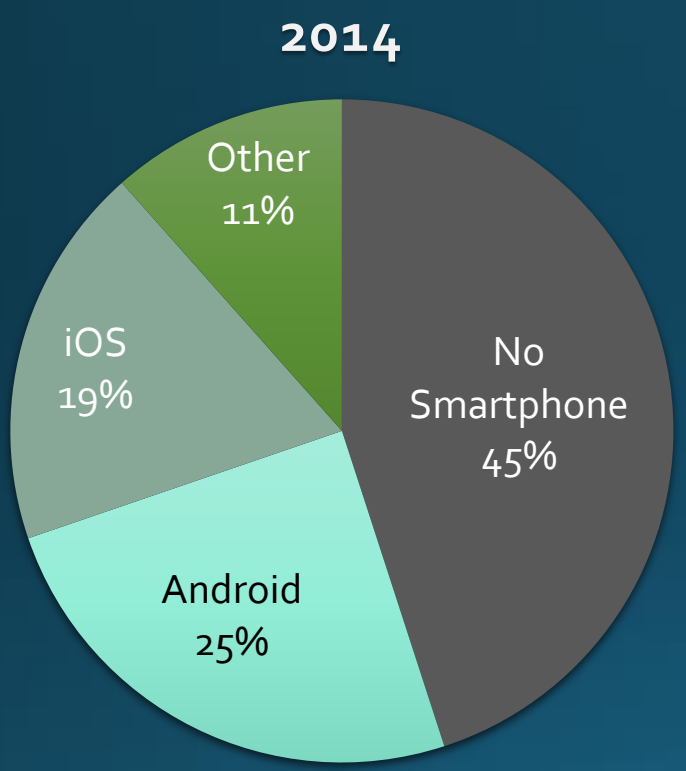
Making Apps

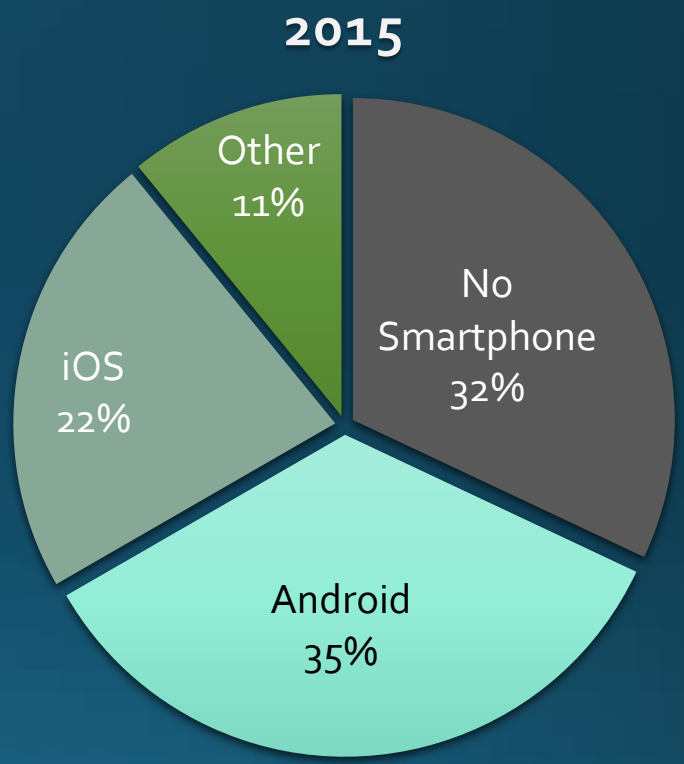
Visual Programming

What is an app?

- An App is one tool of many on a smartphone.
- An App specializes on accomplishing one goal.
- An App is like a single tool in a Swiss army knife.

Canadian Smartphone Adoption





2015 - Ryerson University
Further Reading: http://catalyst.ca/2015-canadian-smartphone-market/

Application Software

- Applications are computer programs that accomplish a specific group functions.
  - e.g. Microsoft Word, Adobe Photoshop, etc.
- Some of these Applications were very complex and engineered.
  - Sometimes came with huge manuals

App is short for Application

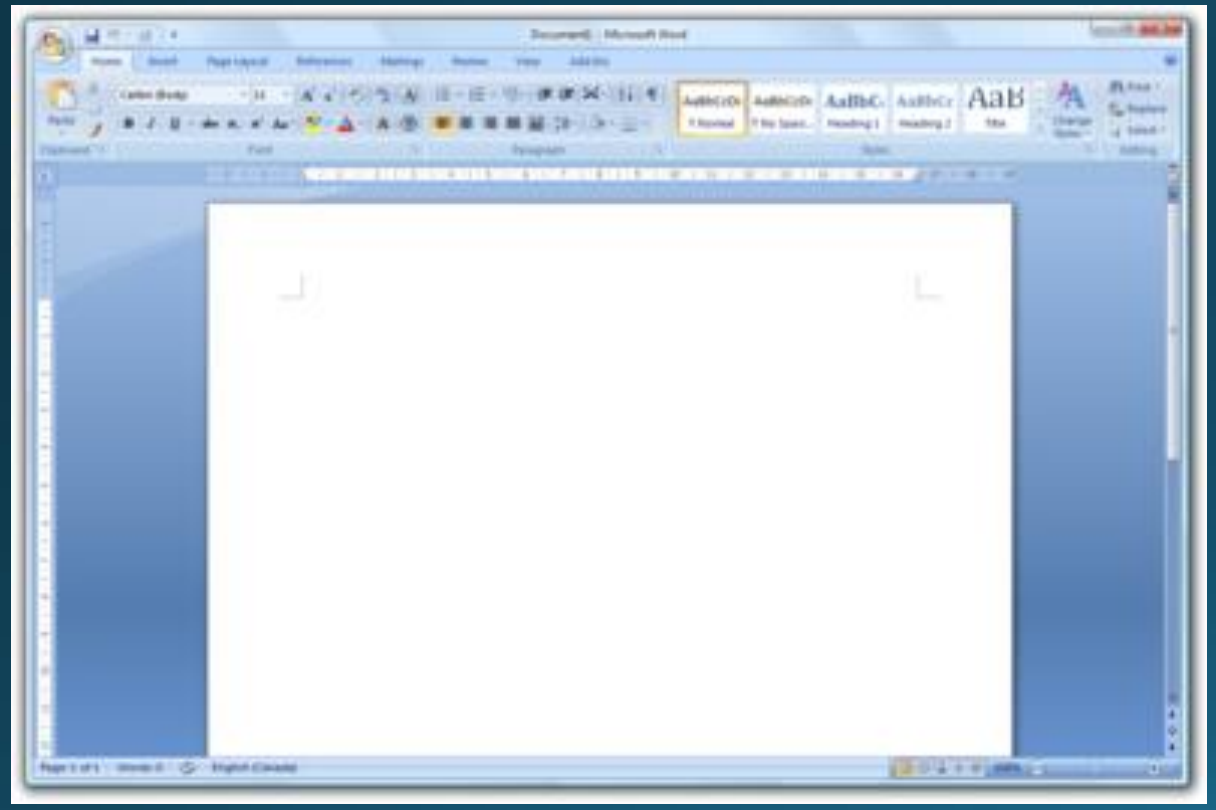
…And not just in name!

- Apps are the new standard, proliferated by smartphones.
- They are now making their way to **PCs**, **TVs** and even **fridges**!

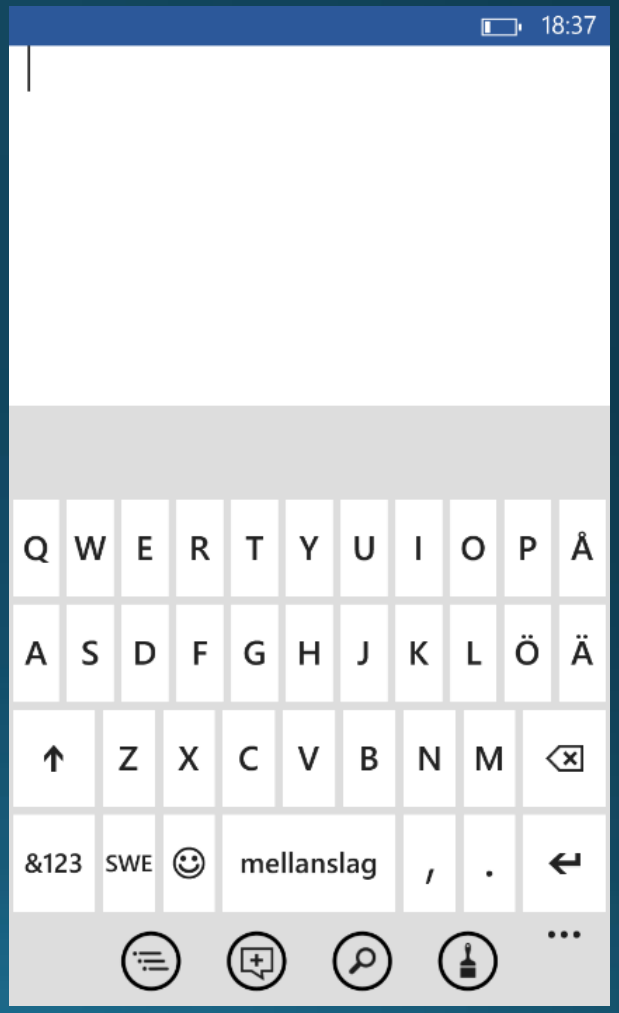
- Key characteristics of apps include:
  - Intuitive
  - Minimalistic
  - Short use time
  - Only one purpose
  - Experience over Function

Further reading: http://www.appcelerator.com/blog/2014/05/apps-v-applications/

Microsoft Office For Desktop



Microsoft Office Mobile App



Why make an app?

- Do you need to keep track of your Exercise routines?
  - Make an app!
- Do you want to be alerted when you're driving towards a traffic jam?
  - Make an app!
- Do you have an idea for a fun game?
  - **Make an app**!

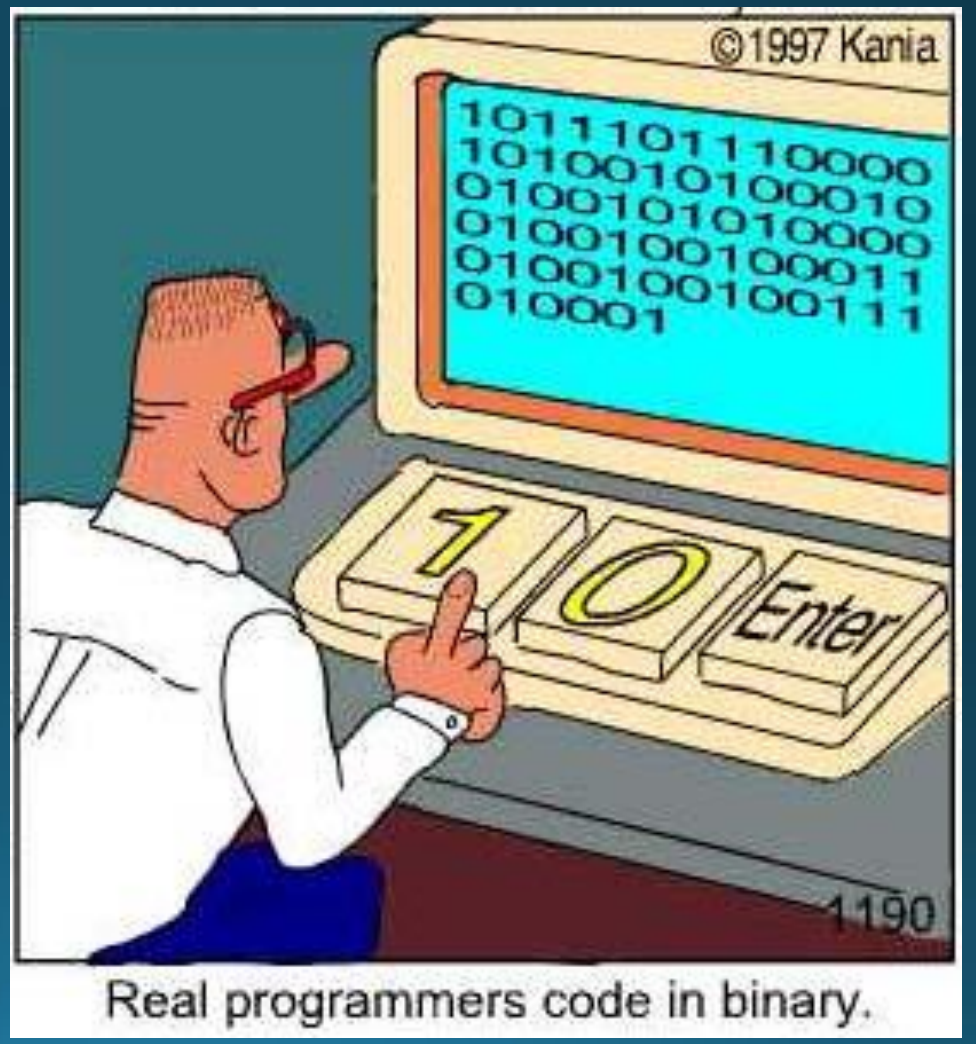
Ok, so we're making apps, but how?

How to make computer programs:

- All computer programs are made up of `0`s and `1`s
  - This is the machine's ***language***
  - Otherwise known as ***binary***

- You just need to find the right arrangement of **1**s and **0**s and type them into your computer

- Clearly this can be very tedious



Real programmers code in binary.

There must be an easier way!

- If you think so too… then you are already thinking like a programmer!
- Even though computers talk in binary, for the last few decades, programmers built tools that can help them express their thoughts more clearly and naturally.
- Programmers write programs in Code that is closer to a natural (human) way of saying things.

Computer:

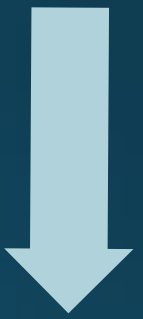
Add 1 to 2.

What is the result?

English

```
int R1, R2, R3;
R1 = 1;
R2 = 2;
R3 = R1 + R2;
```

Source Code

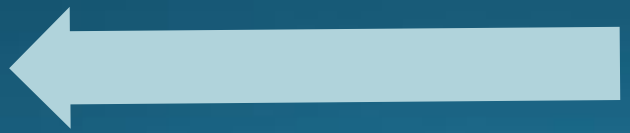


All of these codes do the exact same thing.

As a beginner, which one would you choose?

```
MOV   R1,#01;
MOV   R2,#02;
ADD   R3,R1,R2;
```

Assembly Code



```
0110 0000 1100
0001 0110 1101
0010 1001 1100
1101 1110
```

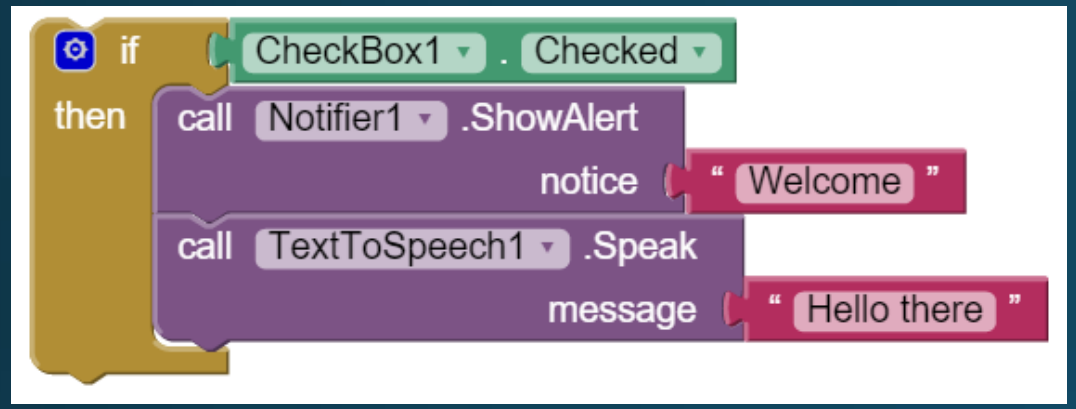
Binary Code

To code or not to code?

- Unfortunately, we can't program in English (yet).
- And while Source code is what is mainly used by professional programmers, it can be very nuanced.
- Learning the syntax of a programming language can distract you when trying to understand the concepts needed to program.

```
def flatMap[B](f: A => StateT[M, S, B])(implicit m: Bind[M]): StateT[M, S, B] =
  stateT[M,S,B](s => apply(s) >>= ((x: (S, A)) => x match { case (sp, a) => f(a)(sp) }))
  semigroup(_.list <::: _)
CanBuildFrom[CC[A], B, CC[B]] forSome {type A; type B}
implicit def Tuple2Traverse[X]: Traverse[({type λ[α]=(X, α)})#λ] =
  new Traverse[({type λ[α]=(X, α)})#λ] {
def traverse[F[_] : Applicative, A, B](f: A => F[B], as: (X, A)): F[(X, B)] =
  f(as._2) ∘ ((b: B) => (as._1, b))
implicit def ZipperTraverse: Traverse[Zipper] =
  new Traverse[Zipper]semigroup(_.value * _.value ∏)
def lift[F[_]](implicit f: Applicative[F]): (F[T1], F[T2]) => F[R] =
  (a: F[T1], b: F[T2]) => (a <**> b)(this)
def promise(implicit s: Strategy): (T1, T2) => Promise[R] = (x: T1, y: T2) =>
  x.pure[Promise].<**>(y.pure[Promise])(k)
implicit def Tuple4Semigroup[A, B, C, D](implicit as: Semigroup[A], bs: Semigroup[B], cs
  semigroup((a, b) => (a._1 |+| b._1, a._2 |+| b._2, a._3 |+| b._3, a._4 |+| b._4))
def traverse[F[_] : Applicative, A, B](f: A => F[B], za: Zipper[A]): F[Zipper[B]] = {
  val z = (zipper(_: Stream[B], _: B, _: Stream[B])).curried
  val a = implicitly[Applicative[F]]
  a.apply(a.apply(a.fmap(
    a.fmap(TraversableTraverse[Stream].traverse[F, A, B](f, za.lefts.reverse),
           (_: Stream[B]).reverse),
    z), f(za.focus)), TraversableTraverse[Stream].traverse[F, A, B](f, za.rights))
}
```

Visual Programming




```
if(Checkbox1.Checked){
    Notifier1.ShowAlert(
        {notice: "Welcome"}
    )
    TextToSpeech1.Speak(
        {message: "Hello There"}
    )
}
```

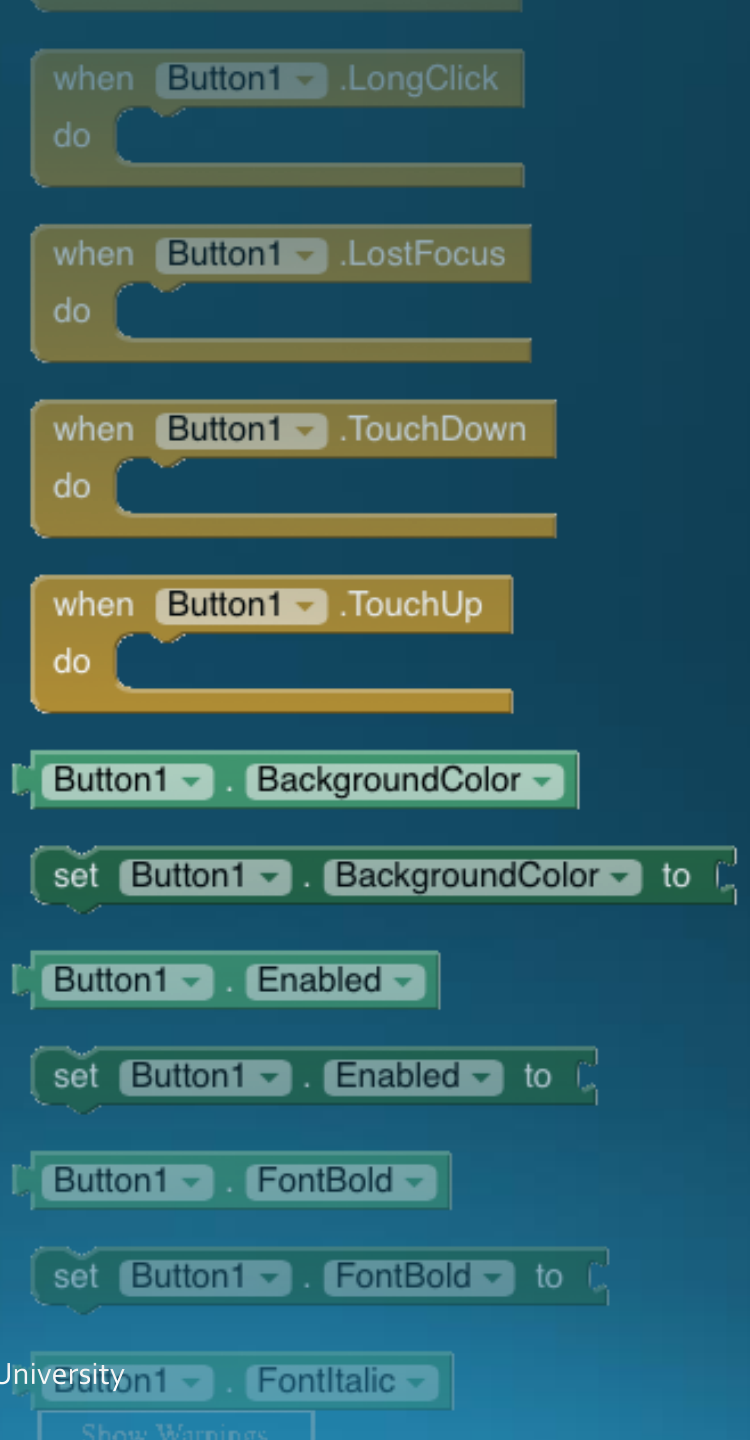
- Instead of Source Code, we'll be using **block based** visual programming.
- As you can see, it is very similar to code.
- Great preparation for learning a programing language (JavaScript, C#, Python, Java)

Visual Programming

- Learning to program using visual programming will give you the same high-level skills as text-based programming
- It will allow us to focus on the concepts behind how our apps will work without getting caught in small details
- We will prototype apps in a matter of minutes.

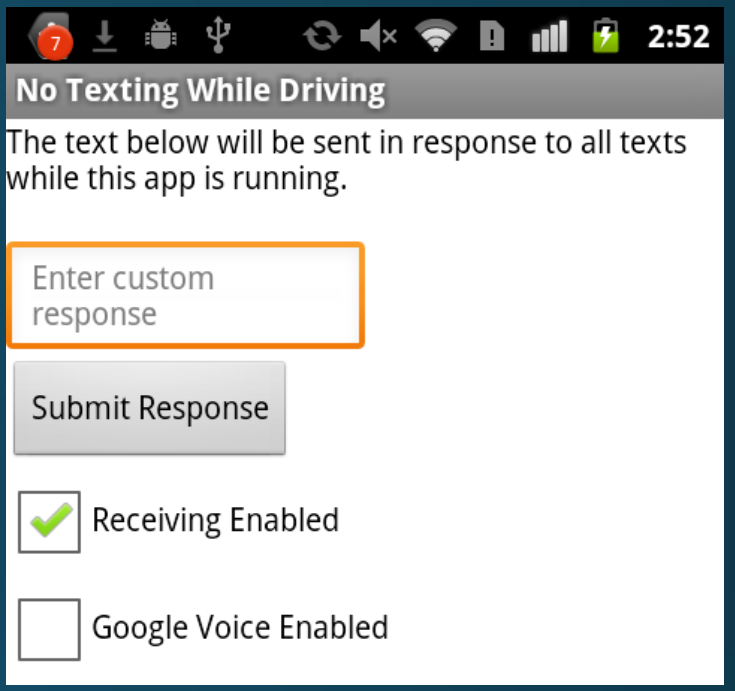
MIT App Inventor

- App Inventor is a Visual Programming system that uses **Blocks** to represent the logic behind apps.
- Browser based (No installer needed!) and open source!
- Developed by MIT and Google Engineers.



No Texting While Driving

- Daniel Finnegan used App Inventor in 2009 to create an App that would automatically reply to text messages when the user was driving.
- The idea was so popular, State Farm Insurance utilized it and distributed it to their clients the following year.
- Software, after all, affects almost everything we do. Pick any major problem and clever software is part of the solution.



Further reading: http://www.wired.com/2010/11/st_thompson_wereallcoders/

First iteration of Challenge Accepted:

Last month we ran our first iteration of this course.
By the end of the course, students were making apps that surpassed our expectations.

Are you ready?
Let's get started.

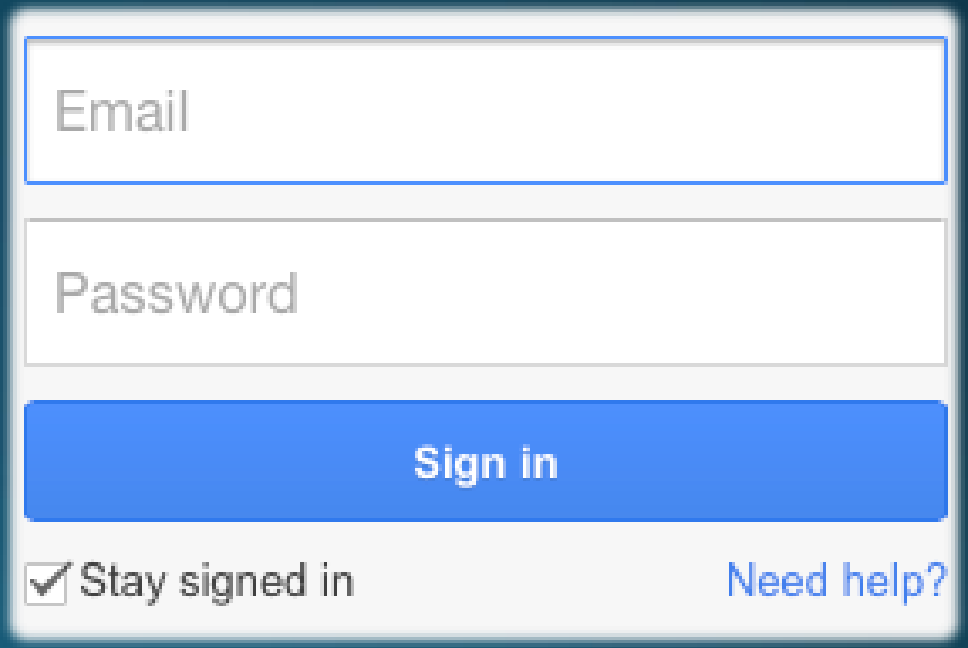
Before we start, requirements:

- Laptop and Charger.
- Modern Browser
  - Please use (preferably) Google Chrome or Mozilla Firefox
  - Be wary that browser extensions may cause problems with AppInventor
- If you do not have a laptop or if you run into problems, the Ryerson Library has a laptop loan program (for Ryerson students).

Getting Started

- Course website:
  - mydmz.ryerson.ca/learntocode/
- Click the link in the top right:
  - Current Students
- Click "Sign in"

- Sign in using a Google account
  - (Your Ryerson account works too)
  - Create a google account if you don't have one.



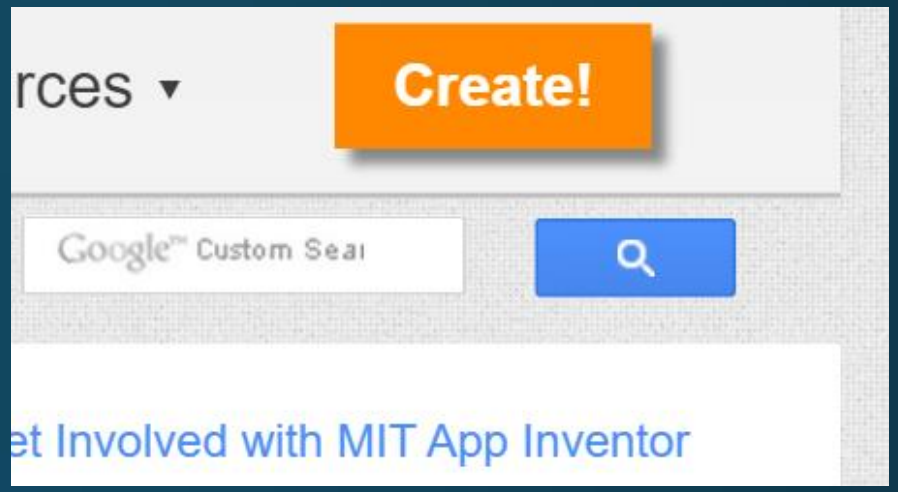
Course Website

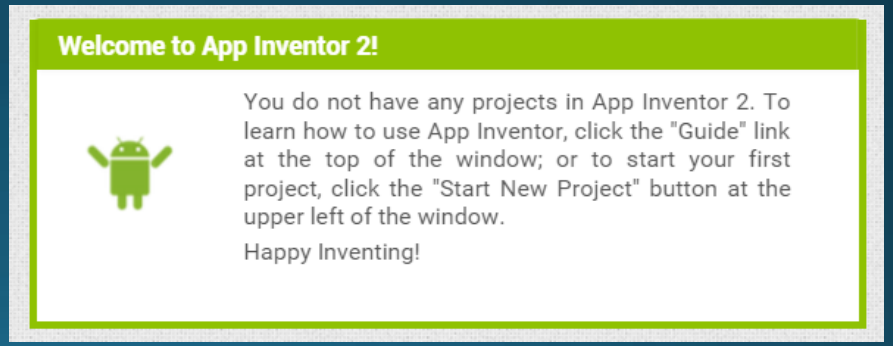
- Fill in the information on the Sign Up webpage.
- Bookmark this page.→

- Slides, links and announcements will be placed here.

Open AppInventor.

- Click on the link at the top that says AppInventor:
  - appinventor.mit.edu
- Click on the orange button that says: Create!
  - Log in with your google account.
  - Accept the terms of service.
  - Dismiss the survey.
  - Dismiss the Popup (Continue).

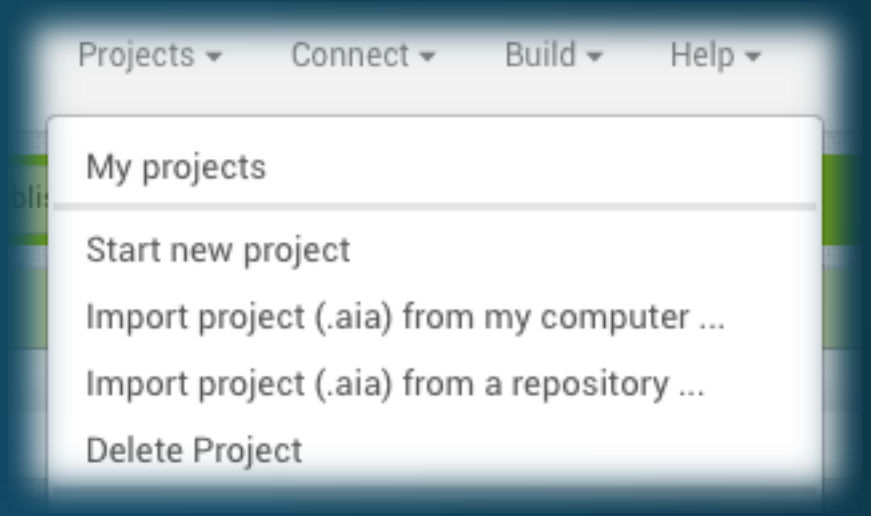


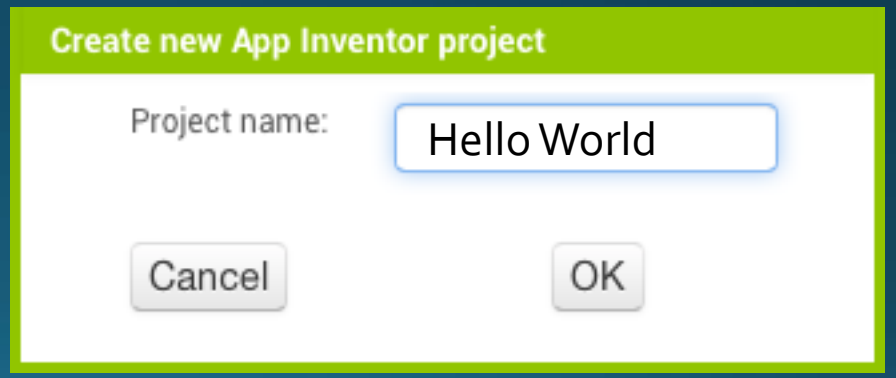


A New Project

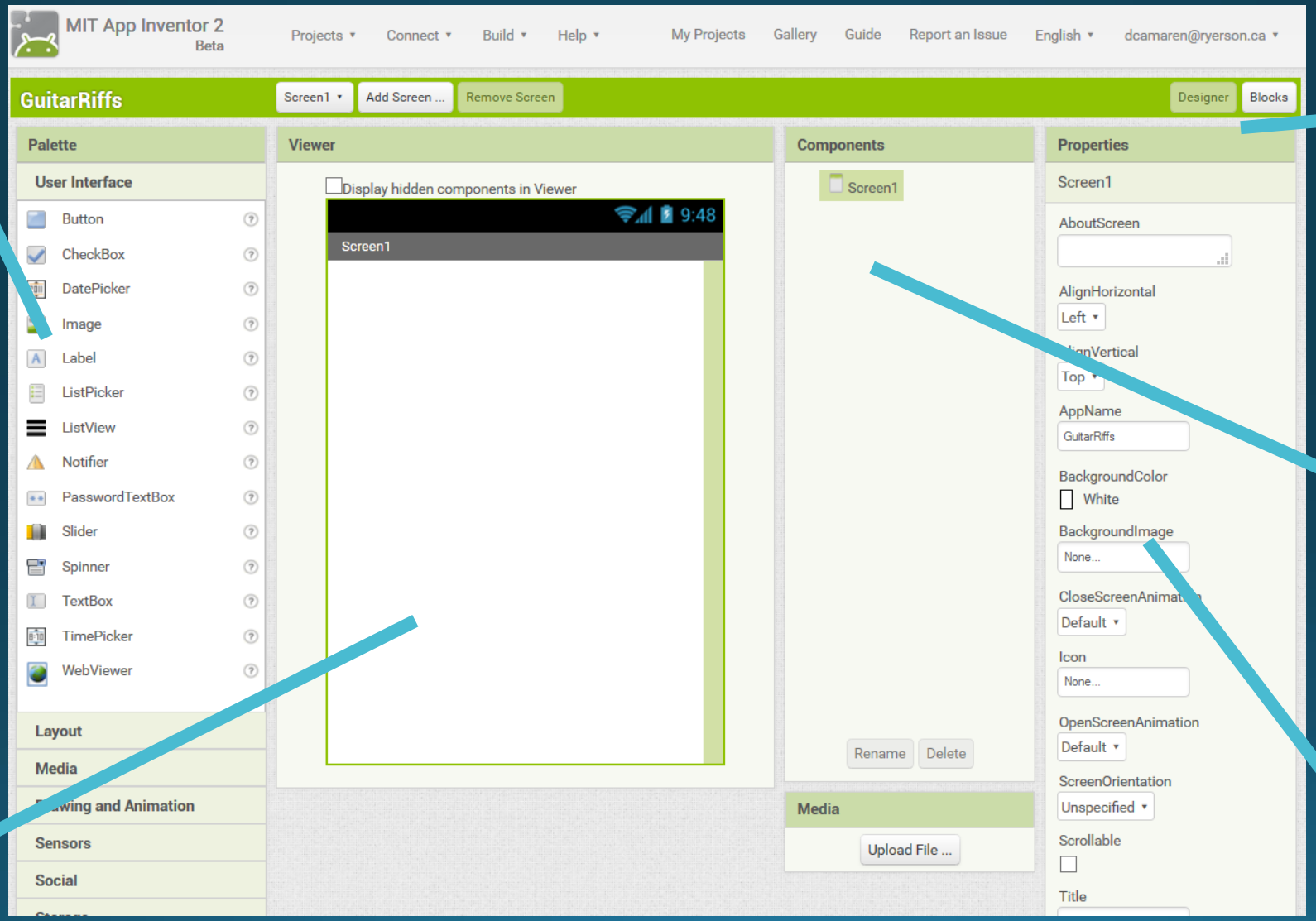
- Click Projects at the top
  - Select 'Start new project'

- Set Project name:
  - HelloWorld
  - Click OK





App Inventor – Designer View



Palette

View switch

Components List

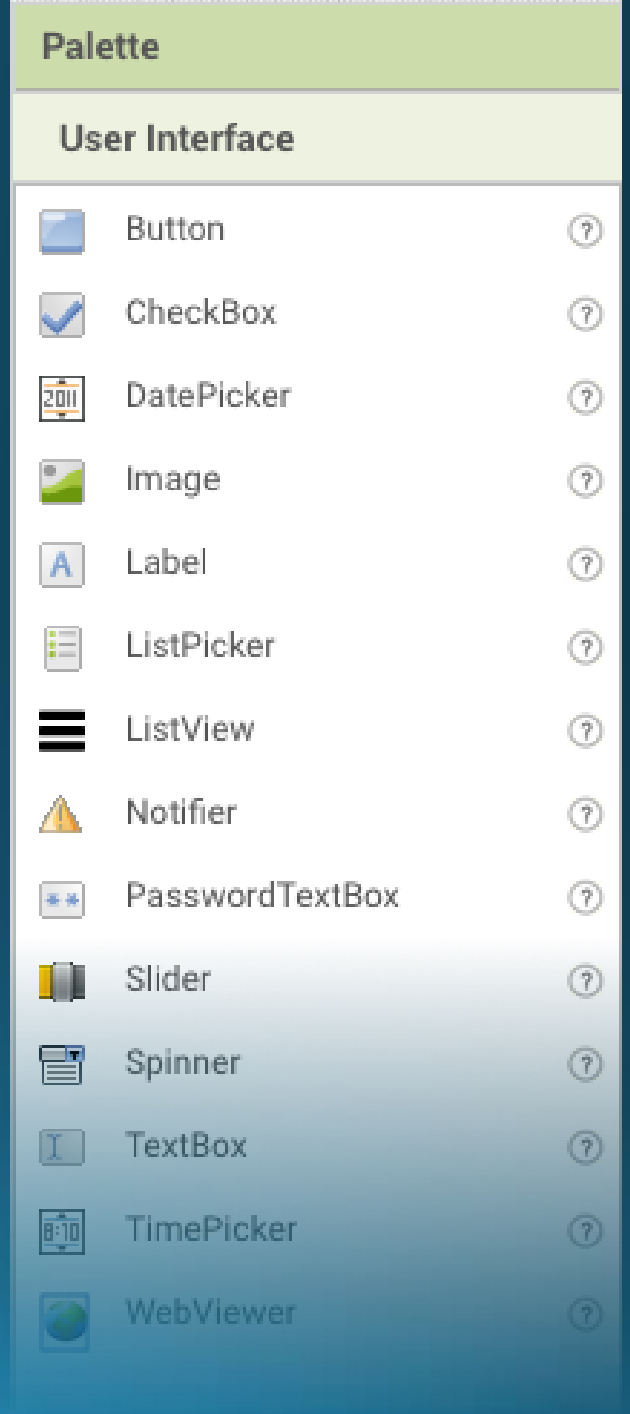
Viewer

Properties

Designer View - Palette

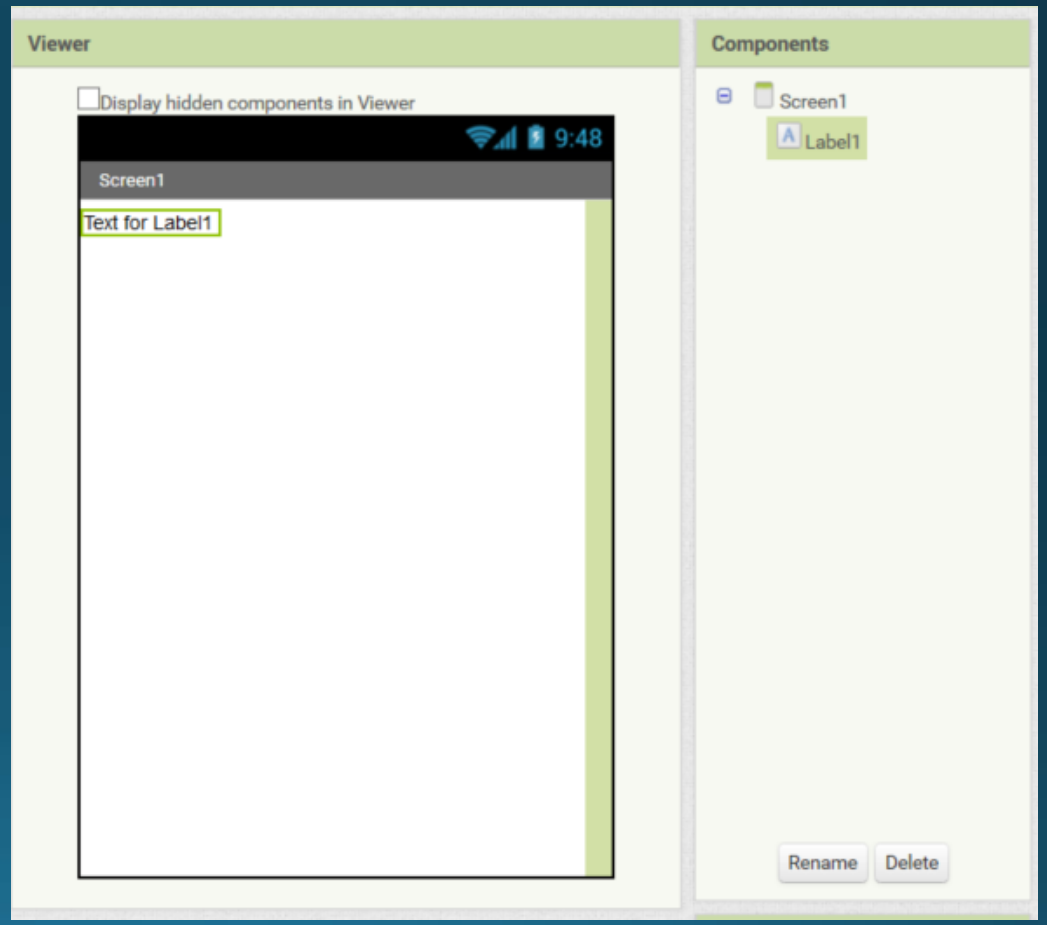
- The palette shows all of the components available for building the UI or adding functionality to your app

- In order to add a component to your app, drag it from the palette to the viewer.



Designer View – Viewer & Components

- The viewer displays the layout of your app's screen. This is what the app will look like to the user.
- The Components Panel is a list of the components that are in use, this may include invisible components
- Drag a **label** component into your Viewer.



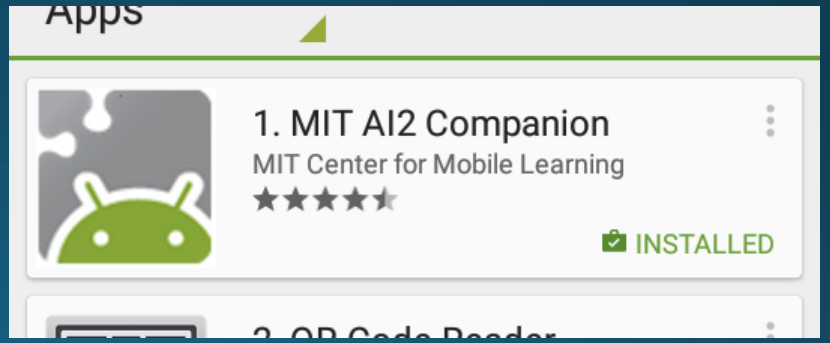
Designer View – Properties

- The properties bar lets you change features of the component you've currently selected
- You can select the component by clicking on it in the viewer or on the components list
- Select the label you've added and change the text to 'Hello World', and the TextColor to Blue

Ok, let's test the app.

Obtaining a device:

- If you don't have an Android device, we can lend you one.
- On the course website, click on the link that says "Borrow a phone"
- After you fill out the forms, You can pick up a device from the front
  - You will need one piece of ID
- If you have your own Android device:
  - Open the Google Play store
  - Search for MIT AI2 Companion
  - Select the Result with this icon:

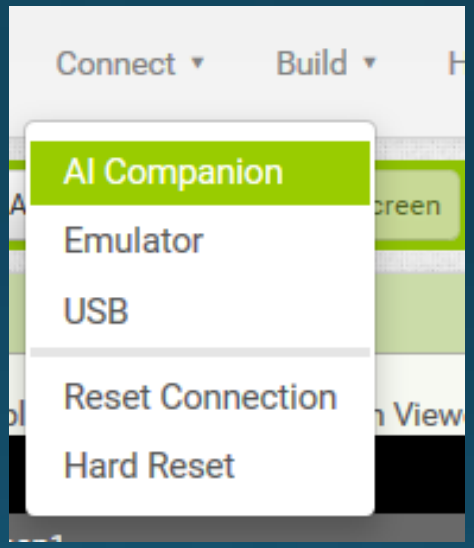


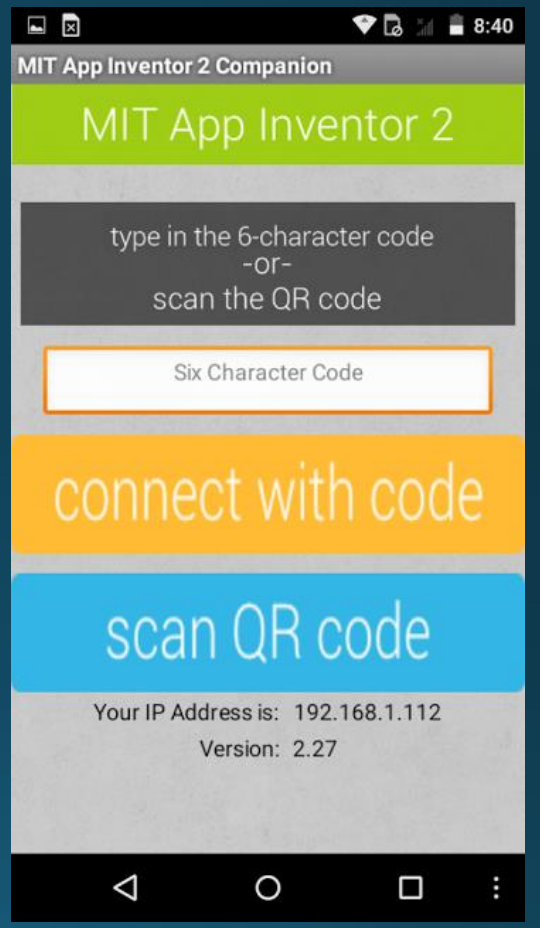
  - Download the app

Testing the app

To run the app:

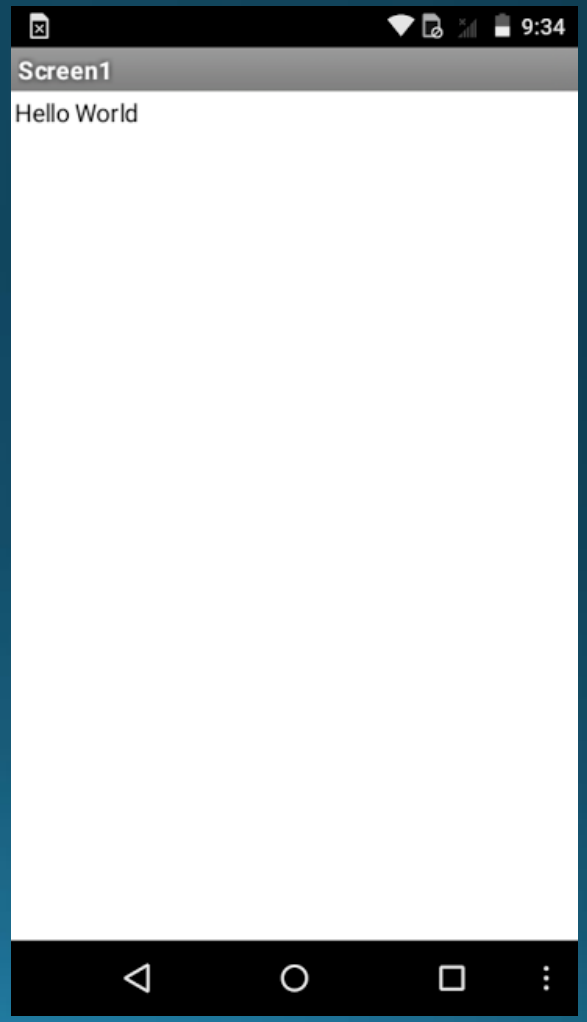
- On AppInventor
  - Go to the connect menu
  - Click AI Companion
- On the phone:
  - Open the AppInventor App
  - Click Scan QR code or enter the code.





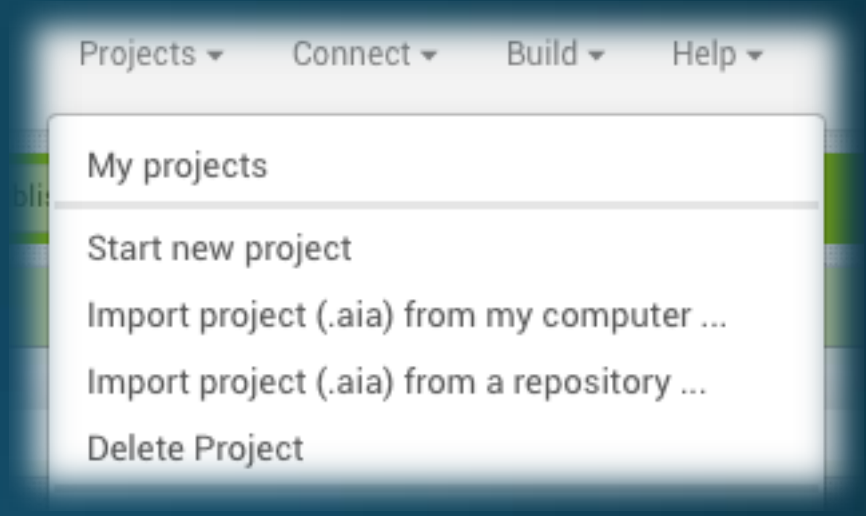
This is what you should see:

- If you see the this screen → on your device, you've successfully Run your first app.
- While it may not do much, it is still an important part of learning:
  - Wikipedia: Hello World
- Let's wait a few minutes to make sure we all have this running.

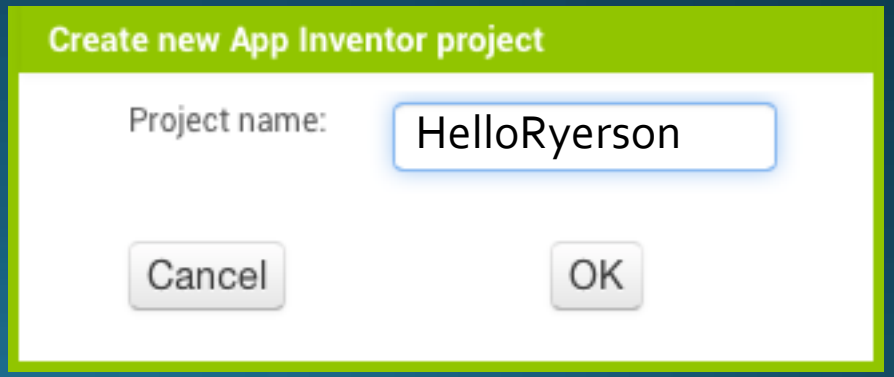


A New Project

- Projects
  - Start new project

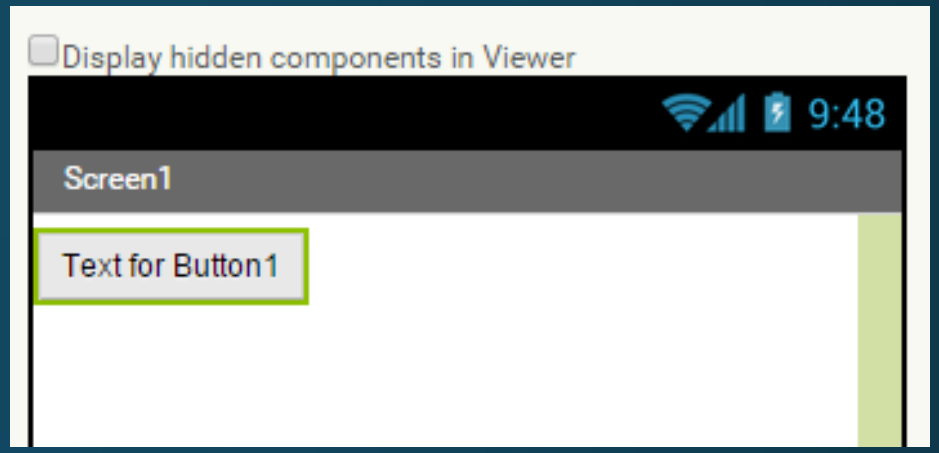


- Project name
  - HelloRyerson
  - Click OK

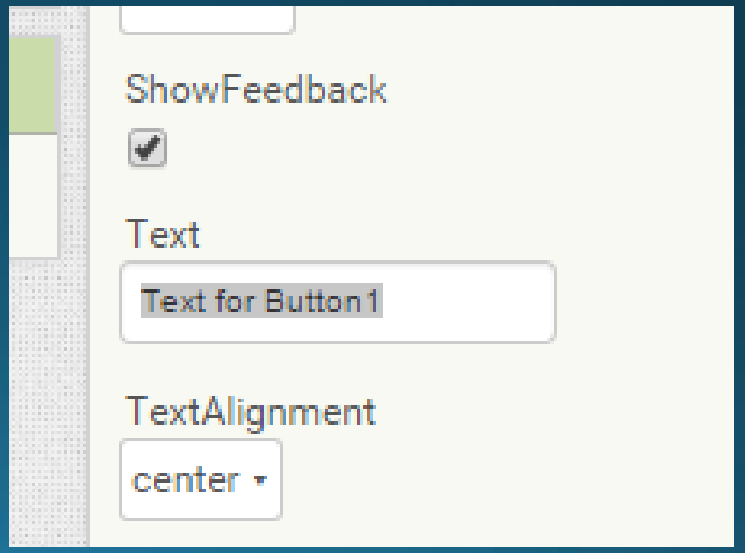


First Button

- Grab a button
- Drag it into the screen

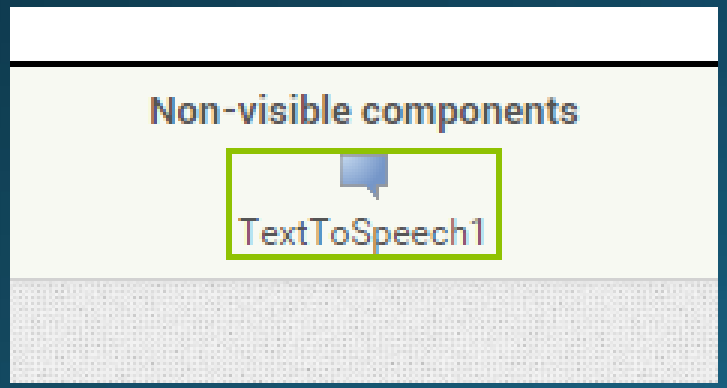


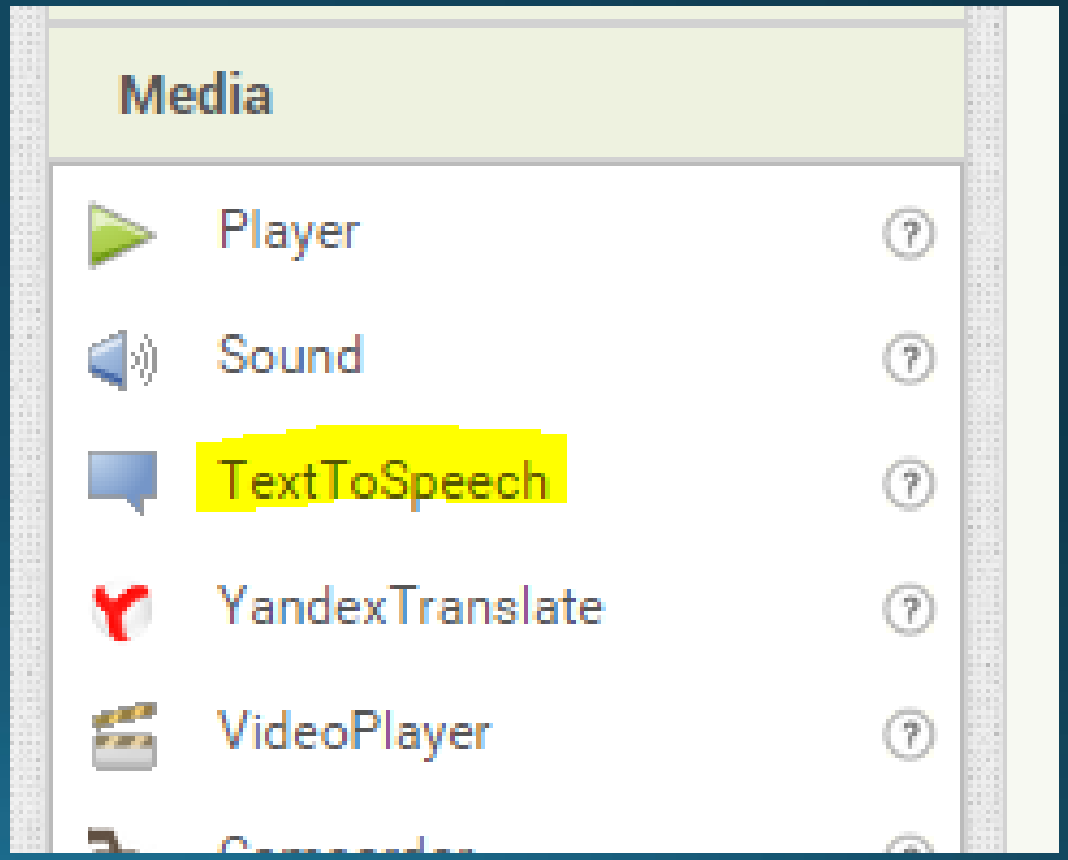
- Change the Text property to "Speak"



Text to Speech

- In the *Palette* (under *Media*), drag a TextToSpeech Component onto the Screen.



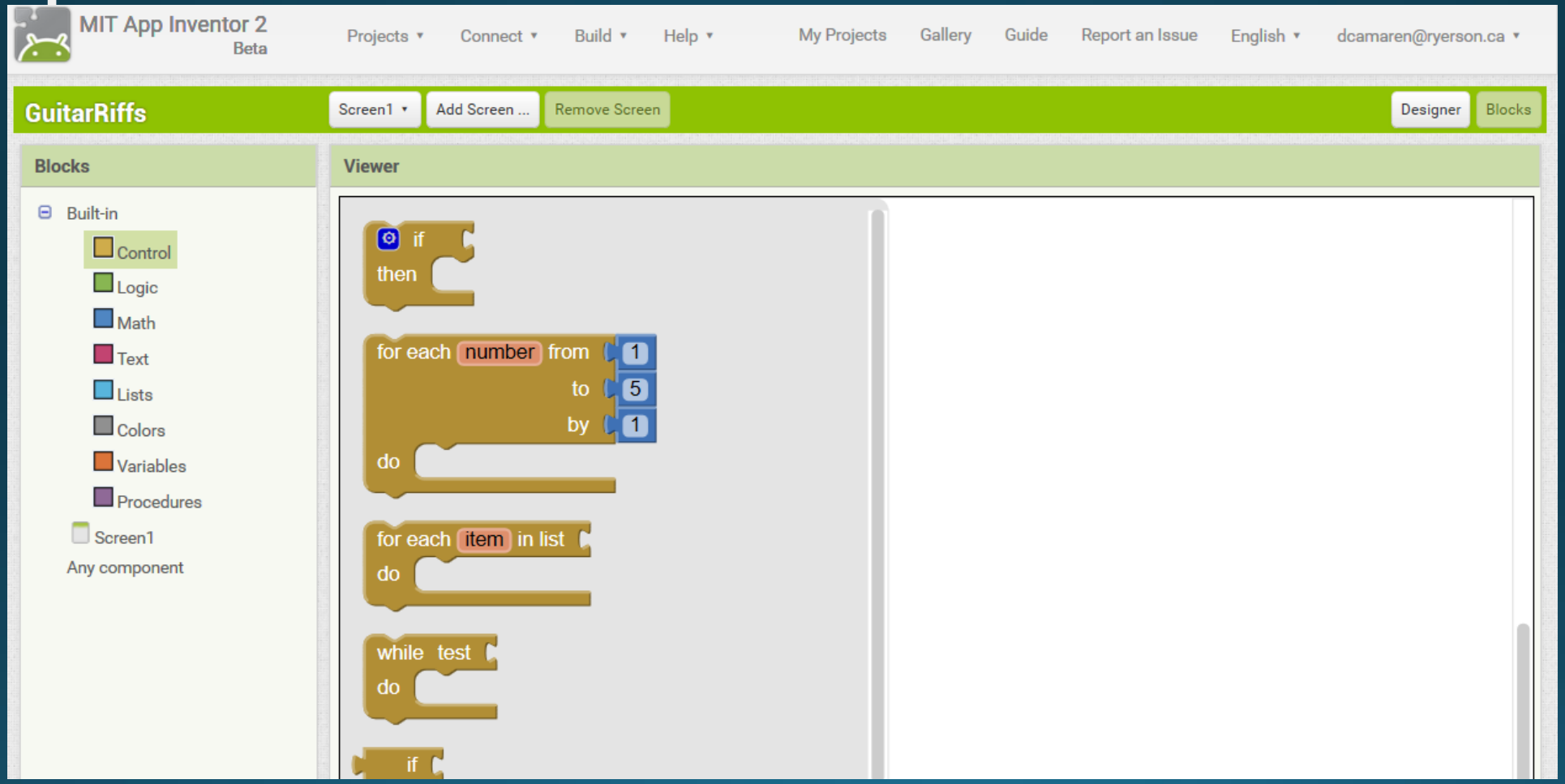


Switching views

- You can switch between Designer and Blocks View by pressing the buttons on the top right.

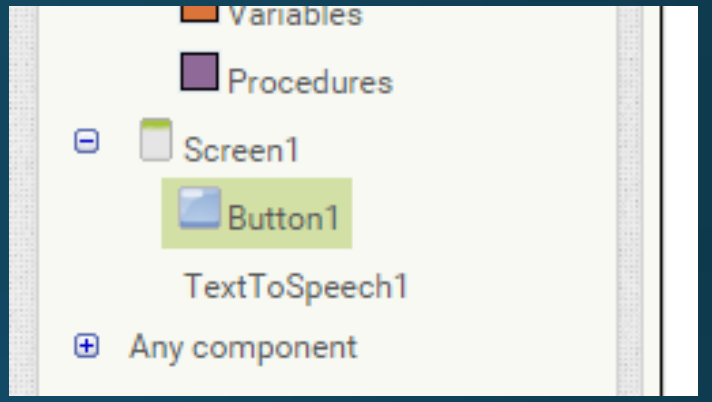
- The Designer view is where we **change the appearance of our app**.
- The Blocks view is where we **change the logic**.

App Inventor – Blocks View



Blocks!

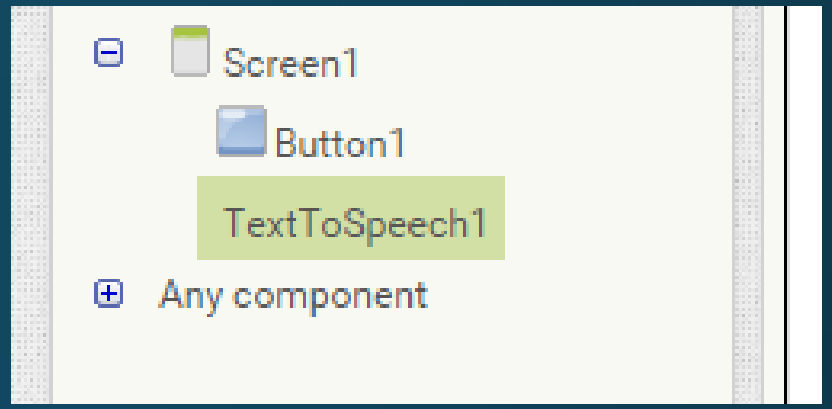
- Switch to the Blocks View
- Click Button1



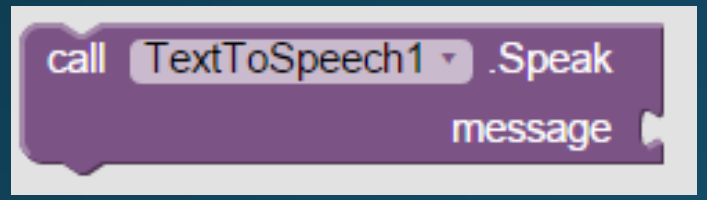
- Drag in the Button1.Click Yellow Block

Text To Speech

- Click The TextToSpeech1 Component

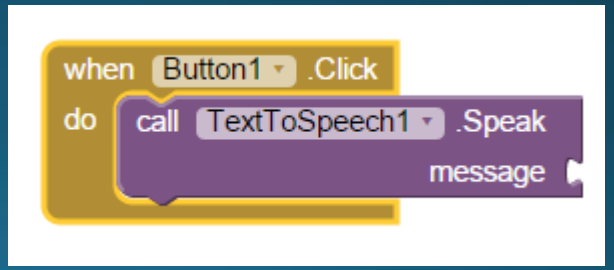


- Drag in a



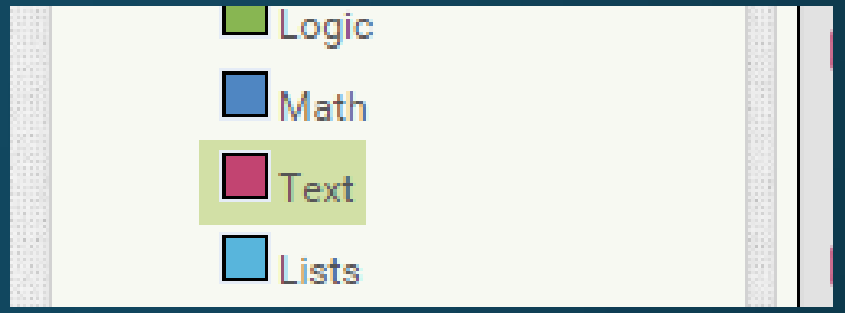
block onto the screen

- Snap it into the Yellow Event block.

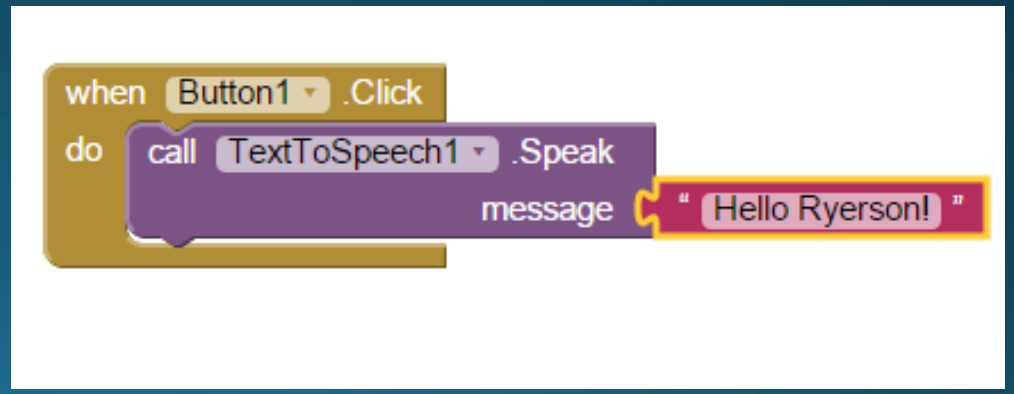


Write a message!

- Click the Text Category
- Drag in a  block.



- Click the space inside and write:
  - Hello Ryerson!
- Snap it into the purple function block.



Does the sound play?

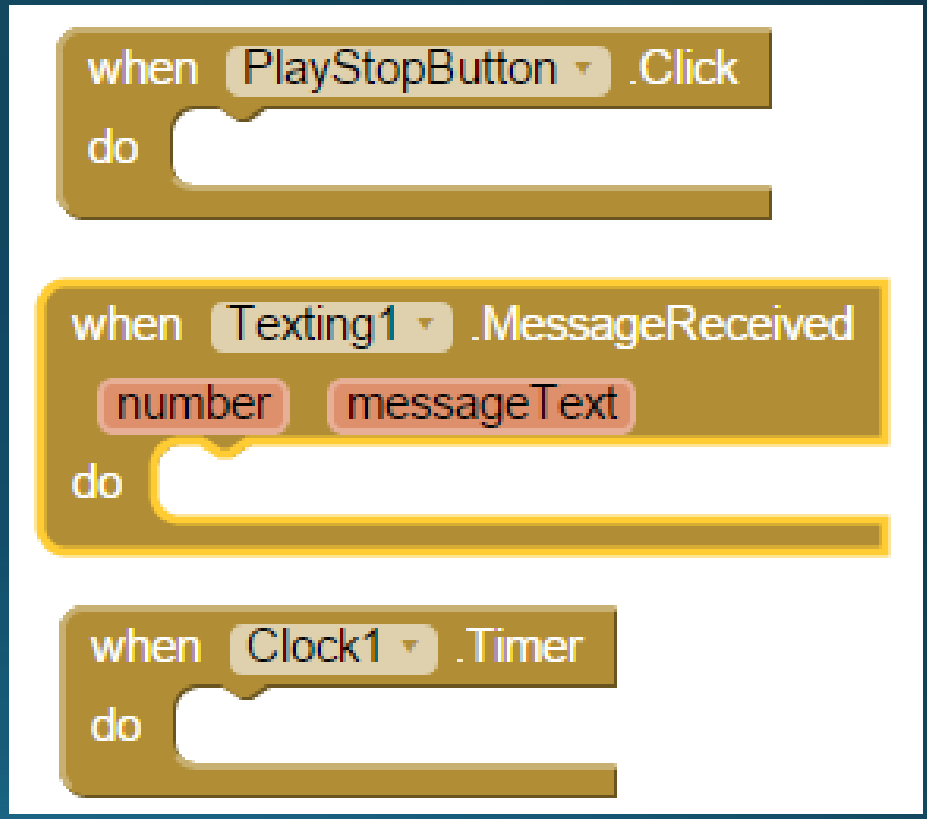
For some android phones, You'll have to restart the app.

Try to test the app.

Events!

Concept #1

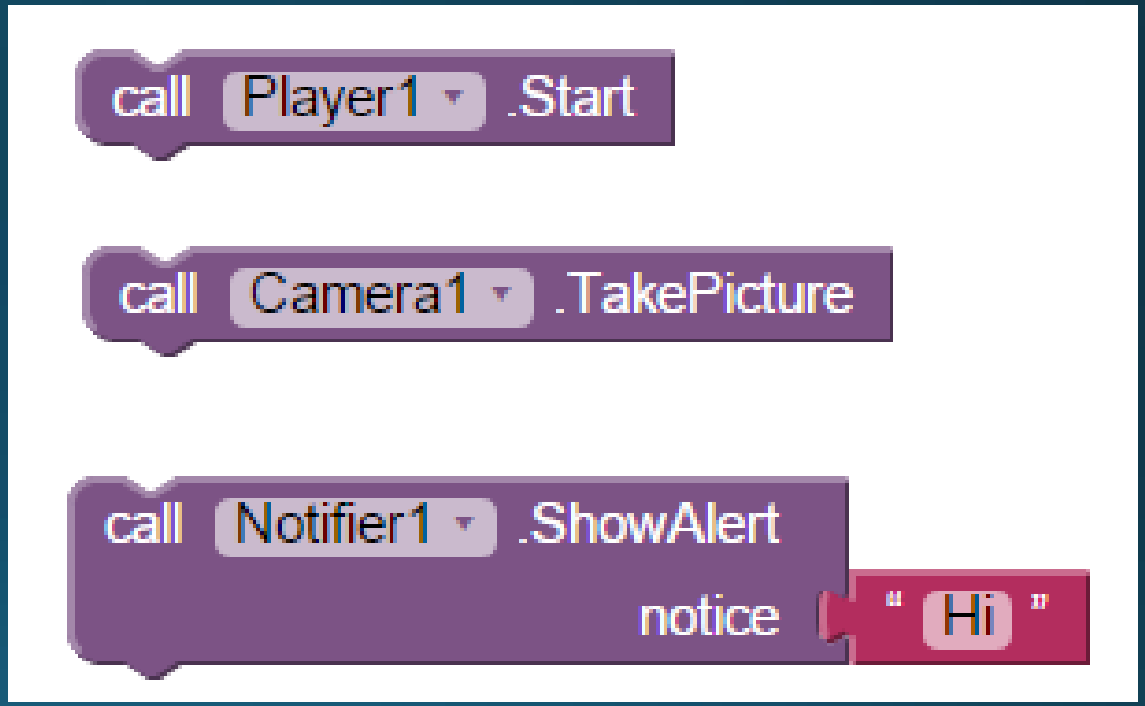
- Events are the starting point of your code.
- Your app (an INTERACTIVE app) reacts to things that happen in the user's world
- For example:
  - The user presses a button
  - The user receives a text message
  - The clock strikes 12.
- Can you think of another type of event?



Function Calls!

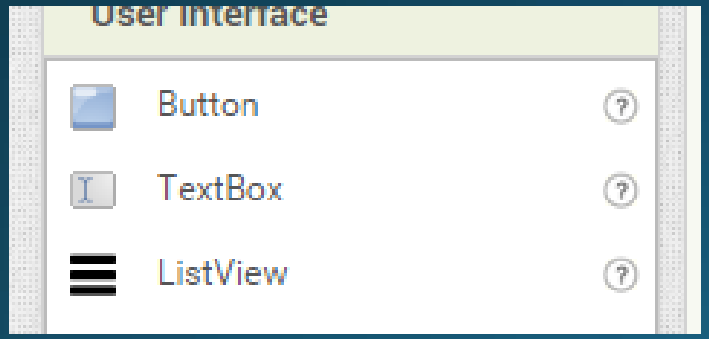
Concept #2

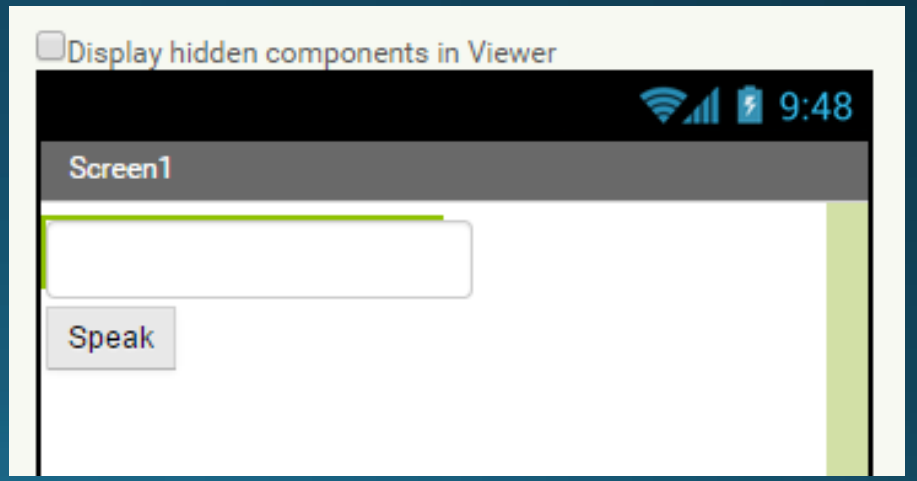
- Function Calls allow us to tell the phone what to do
  - Start playing a music file
  - Take a picture
  - Show a Message to the user
  - We'll learn more about these in a later lecture.



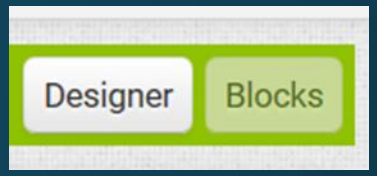
Adding a TextBox

- Go into Designer view.
- From the *Palette* (under *User Interface*), drag a TextBox onto the screen.

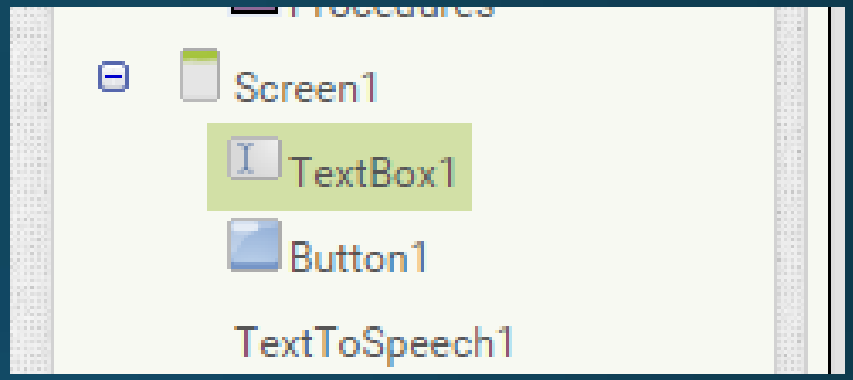




Saying custom messages

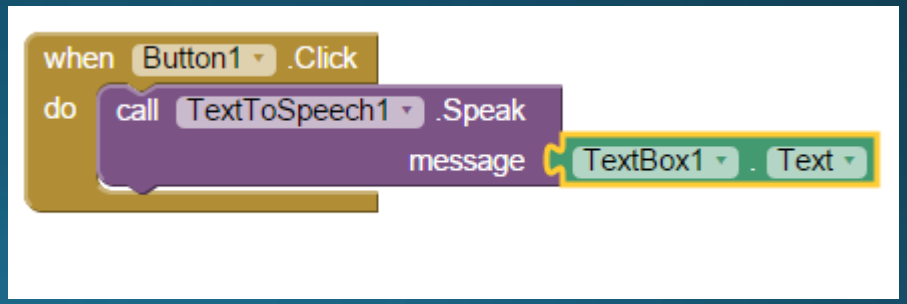


- Go into Blocks View.
- Click on TextBox1.



- Drag the light green **getter** block to set the property (in this case, setting the Text property).

- Snap it into the Purple function Block and Replace the Pink "Hello Ryerson!" block.



Does the sound play?

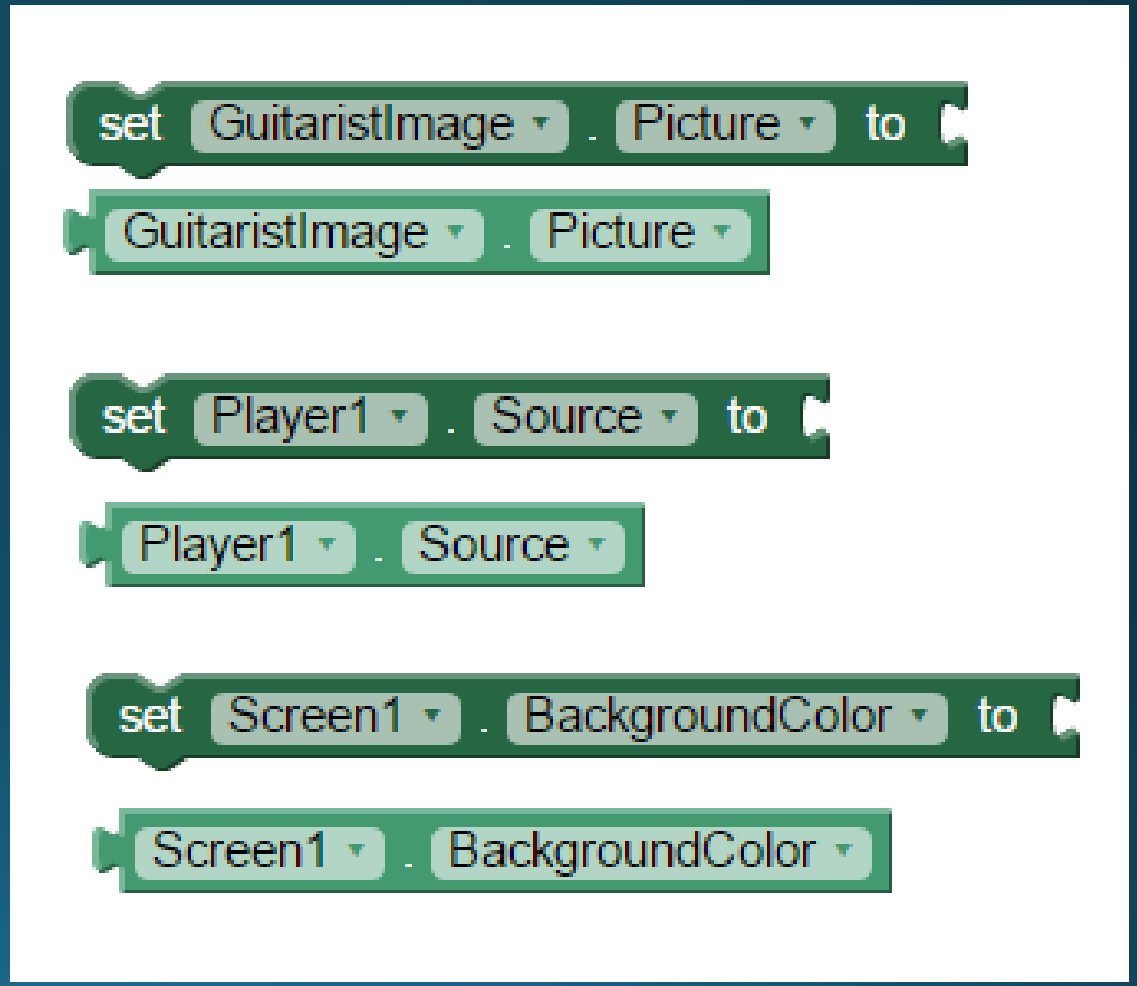
For some android phones, You'll have to restart the app.

Try to test the app.

Properties!

Concept #3

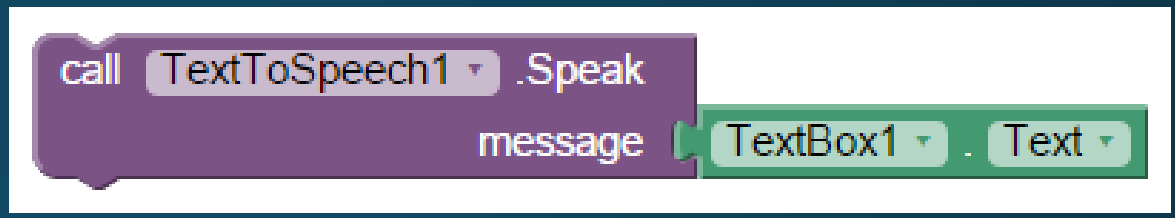
- Properties define the characteristics of components.
- They are available by clicking on the component in the Blocks list
- You can change (***set***) a property or use (***get***) its values
- Some examples of properties:
  - The picture in an Image component
  - The song in a music player
  - The color of the Screen's background



Getter and Setter blocks

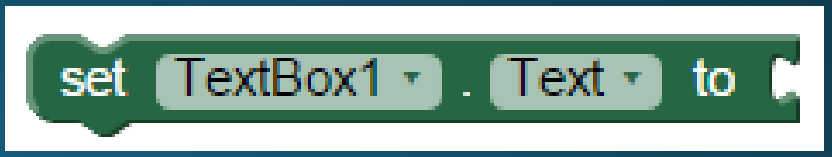
- To access and use the value of a property, use a Getter Block.

Getter



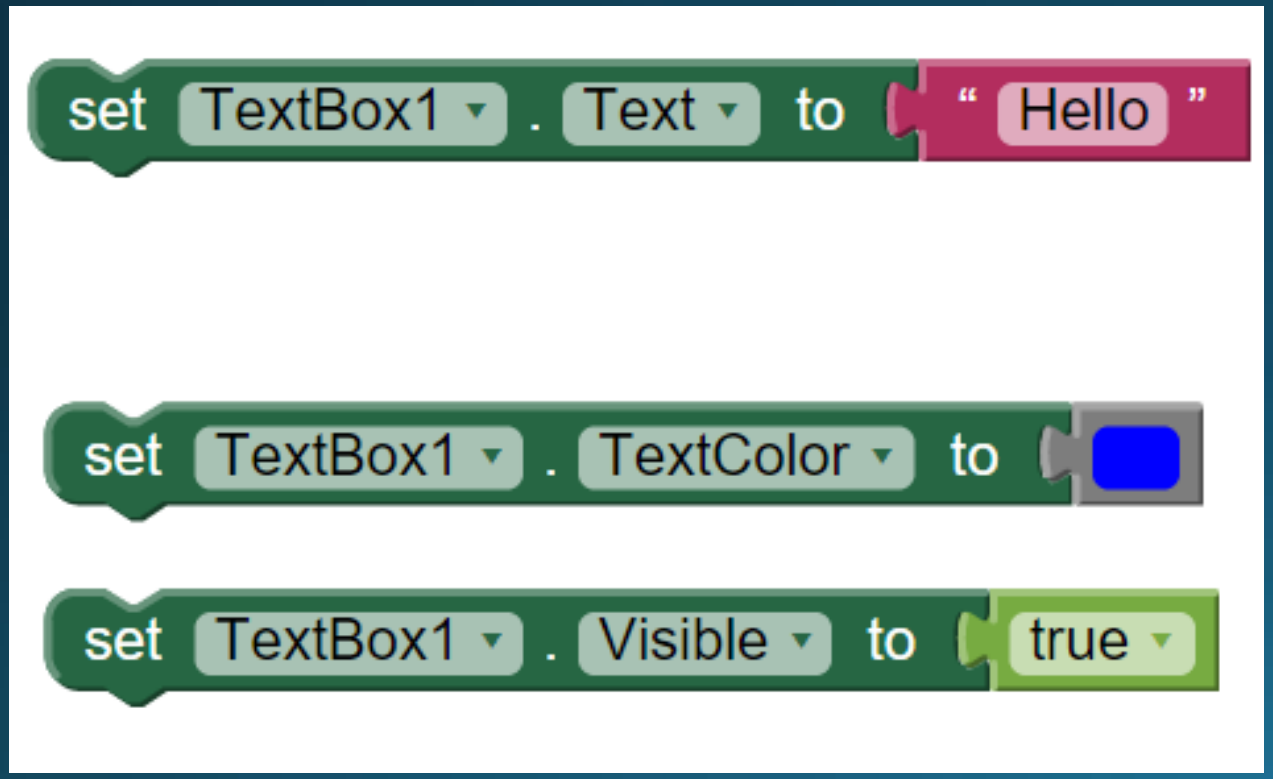
- To change the value of a property, use a Setter Block.

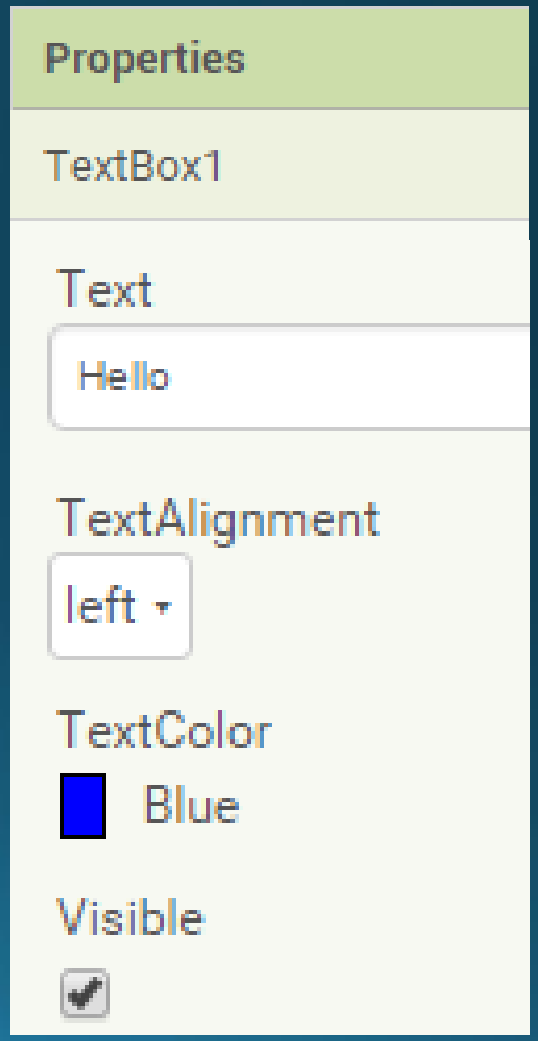
Setter



Properties – Designer vs Blocks

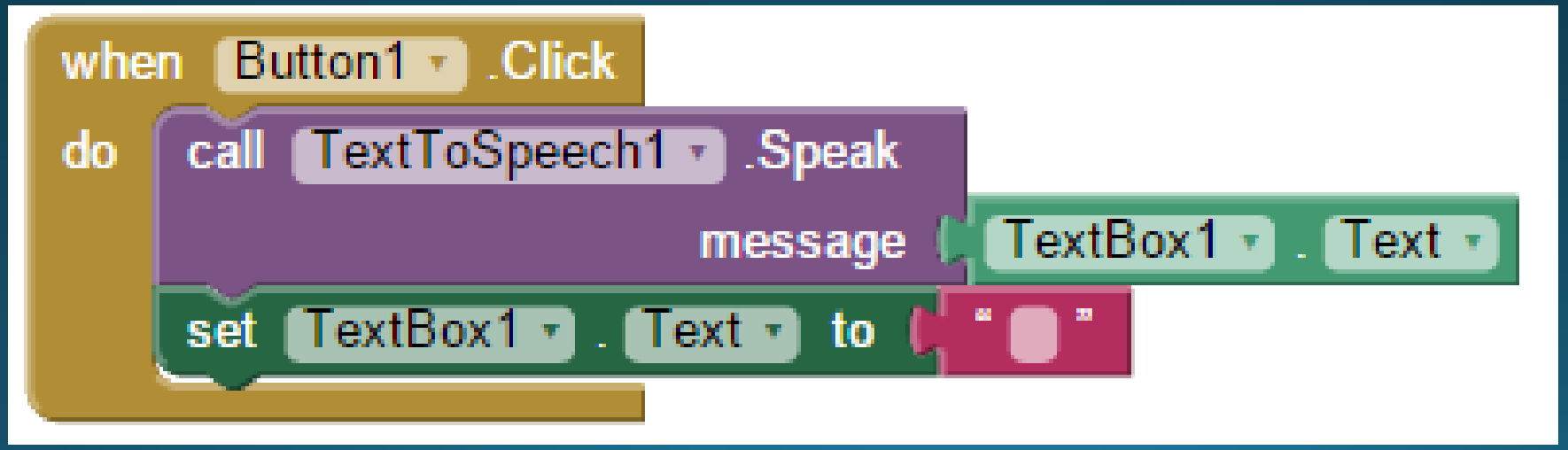
- Properties in the designer are the same as those in the Blocks
- Blocks let you change them on the fly





Clearing on play

- If we want an our app to clear the textbox after saying the message, we have to change the textbox's text
- Use a **setter** block to set the textbox's text to empty:



Challenges ahead.

Test your app and take a break