CyberCamp at UNK

MIT App Inventor 2

What you will learn with this tutorial...

In this tutorial we are going to build and Android application. We will develop a basic calculator that can run on any Android device (smart phone or tablet). This guide will teach you the basics of the design process, programming and testing. At the end, you will have a fully functional calculator application. Let's start!

1 Definition

The **MIT App Inventor 2**, also called *"App Inventor for Android"*, is an open-source web application where you can easily create apps for your Android devices. It has a lot of resources that you can make use of, and real time debugging, which means that you can see the changes directly on your device while you develop your project. We will now go through the steps to develop a basic calculator using the App Inventor.

2 Requirements and log in

As stated on the previous section, the App Inventor is a web application, which means we will access it through a website. The URL for it is: `http://ai2.appinventor.mit.edu/`. You will **need to have a Google account**. Gmail is a Google account, so if you have a Gmail log in with your credentials now. If you don't have a Google account, you will have to create one in order to be able to access the App Inventor.

After you click on log in, you may have to authorize the App Inventor to access some information of your Google account. Click on *Allow*. A window with the *Terms of Service* will show up, click on the button *I accept the terms of service* to continue.

Another box asking about a voluntary survey will show up, take it if you wish, otherwise just click on *Never take survey*.

A box welcoming you will then pop up, go ahead and close it. The log in process is complete. Now you have to prepare the device where you will test your projects.

Get you phone or tablet, open the play store and search for: **mit ai2 companion**. Select the first result and install it. An icon will then be created on your home screen with the launcher for the application. That's it for now, we are ready to start our project.

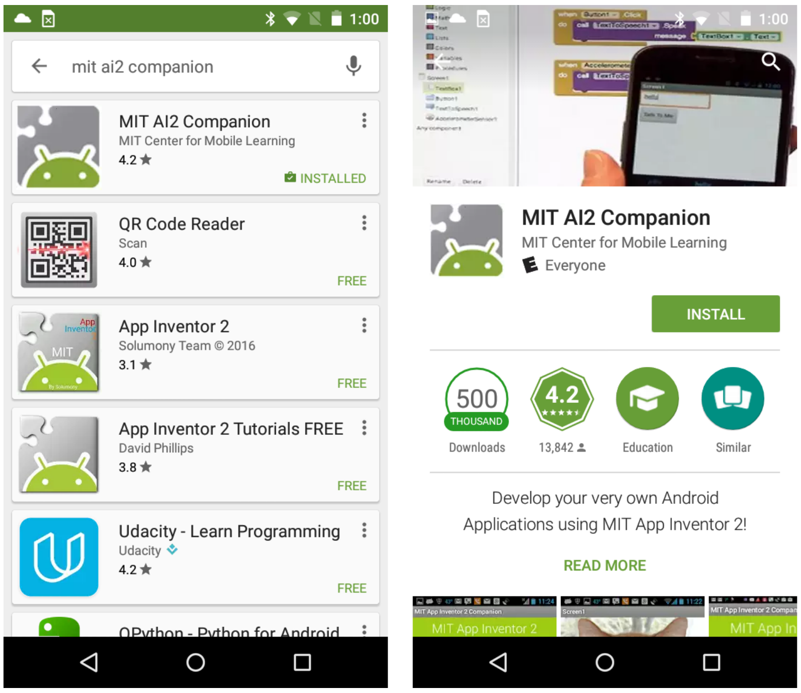
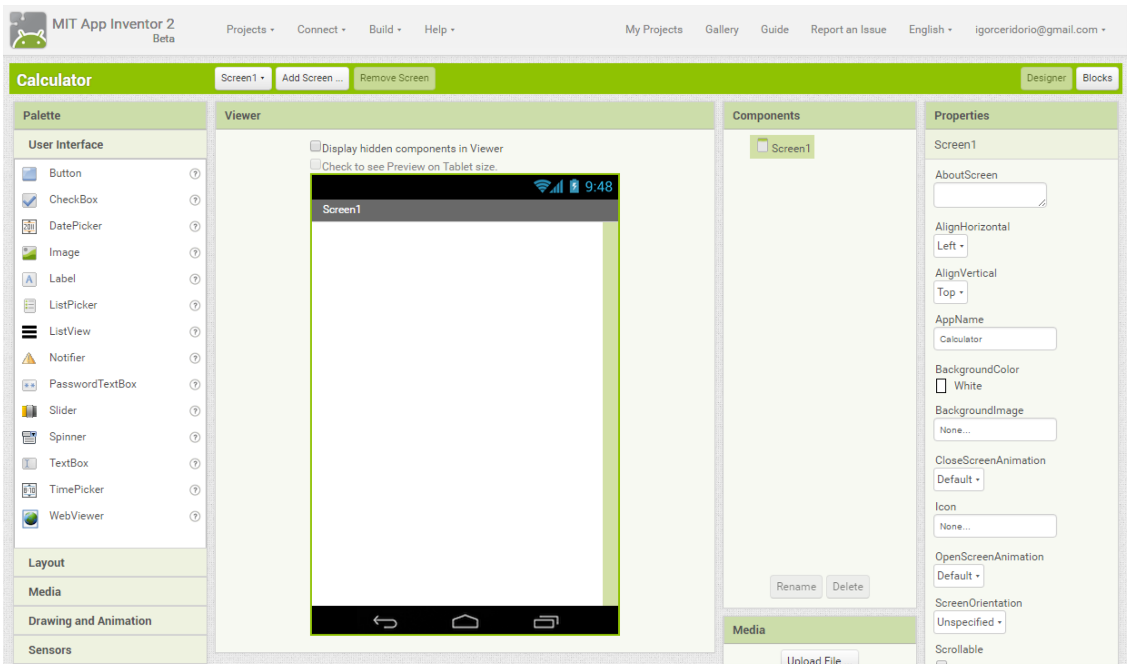


Figure 1: MIT AI2 Companion installation

3 Designing the application

First step is to create our project. Go to the App Inventor website, and after the log in click on the top left button *Start new project*. Give it a name of your preference. Once it's created, double click over it to access your new project.

You will now be on the main screen of your application, similar as follows:



AboutScreen

AlignHorizontal: Left

AlignVertical: Top

AppName: Calculator

BackgroundColor: White

BackgroundImage: None...

CloseScreenAnimation: Default

Icon: None...

OpenScreenAnimation: Default

ScreenOrientation: Unspecified

Scrollable

Figure 2: Project main screen

Now that we have our project created and opened we can start creating our interface.

Our calculator will be made of:

- One display;
- Number buttons from 0 to 9;
- Four operation buttons (+, -, x, /);
- One clear button (C);
- One equal button (=).

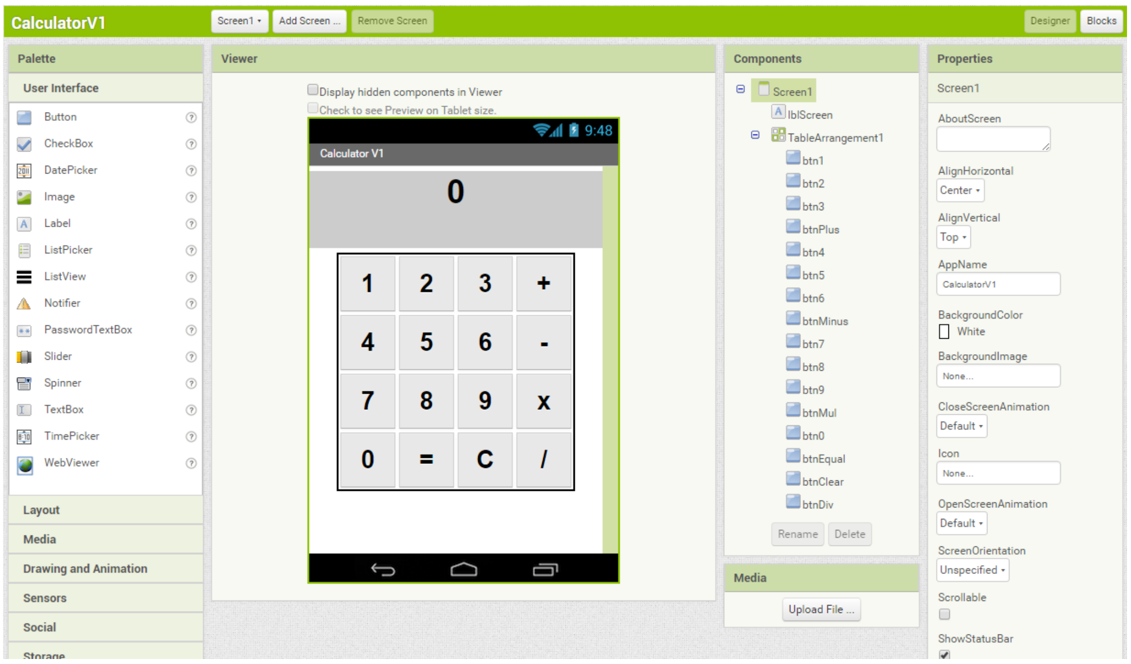
This brings a total of 17 components, 16 buttons and 1 label (the display). Follow these steps to create the interface:

1. In the right side of the screen, note that you have **Components** and **Properties** boxes. The component box will list all the components we add to the design. The properties box allows us to change many characteristics of each one of these components. Click on the *Screen1* component, and in the properties box set *AlignHorizontal* to *Center*.
2. In the left side, on the **User Interface** category, select and drag a *Label* component to inside the screen. This will be the display of our calculator. Change the text of the *Text* property to "0". This is the text that will show up when you first load your application. You can also change some other properties, like *FontSize*, *Height*, *Width* and *TextColor*, if you want. The customization is up to you!

3. We need now to place our 16 buttons. We will organize them inside a *Table arrangement* layout. In the left side, under the **Layout** category, select and drag a *Table arrangement* component to inside the screen. In the properties of this component, set *Columns* and *Rows* values to 4. Each cell of the table will hold one button.

4. The last part is to create the 16 buttons. The layout of your calculator is up to you. This item will describe how to create one of the buttons. You will then repeat this process for the other 15 as well. Select and drag a *Button* component from the **User Interface** category to inside the *Table arrangement* component on the screen. Since we will have a lot of buttons it's interesting to give meaningful names to them. We will need these names later when we start programming the behavior of our app. To rename a component: select it in the **Components** box, and click on the *Rename* button at the bottom of the box. Give it a name like *btn1* or *button1*, for example (something that can identify and be easily remembered later). Now change the *Text* property to reflect what is the button. If it's the button one, change it to "1". You can now change other properties like *TextColor*, *FontSize*, *Height*, *Width* and so on (just like you did with the *label* component).

After all this work the design of your calculator should be done! Down below you can check a simple example of how it can look like:



Screen1
AboutScreen
AlignHorizontal: Center
AlignVertical: Top
AppName: CalculatorV1
BackgroundColor: White
BackgroundImage: None...
CloseScreenAnimation: Default
Icon: None...
OpenScreenAnimation: Default
ScreenOrientation: Unspecified
Scrollable: ☐
ShowStatusBar: ✓

Figure 3: Example of project design

Note how each one of the buttons has a meaningful name. This will help a lot when we get to the programming part of the project.

4 App preview and debugging

To see how your interface will look like in your Android device we can use that application we installed before on section 2.

Open up the *MIT AI2 Companion* app on your Android. At the website, in the top menu select **Connect** > **AI Companion**. This will open a QR Code in the middle of the screen. On the Android device tap on *scan QR code*, and then scan the code of the screen. After the code is scanned tap on *connect with code*. That's it, now you should be able to preview your calculator design and behavior directly on your Android.

The next step is to program our app. Let's do it.

5 Programming the calculator

Now that the design part is done, we will start programming the behavior of our calculator.

App Inventor has a ***block structure*** programming style. All of our code will be created with blocks that connect with each other, like a puzzle. It's really easy programming with App Inventor.

First step is to access the *Blocks* environment. On the top right side of the screen you will see two buttons: *Designer* and *Blocks*. Since we are done with the design part, go ahead and click on *Blocks*. This is the blocks screen that you should be right now:

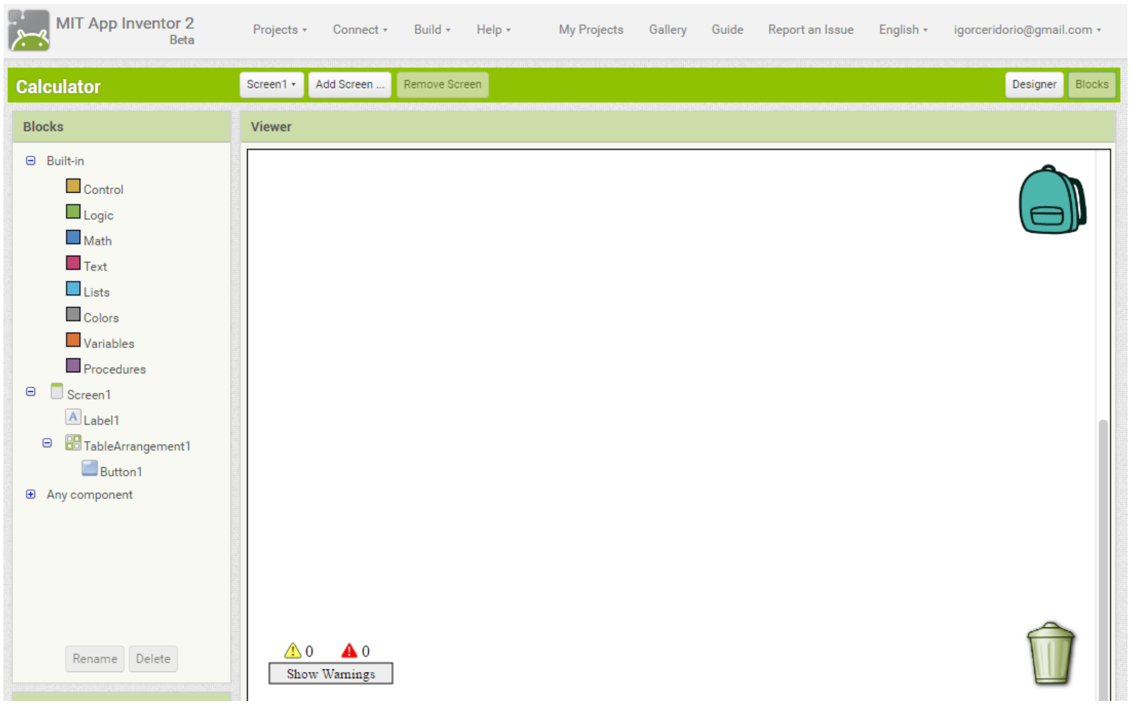


Figure 4: Project blocks screen

On the left side of the screen you will see a list with **Built-in** blocks, and under that another list with all the components of your created design. We will use both as we create our desired behavior.

Let's start thinking about our problem. We want to develop a calculator that is capable of executing the four basic operations with 2 different numbers. So our mathematical expression will be made of:

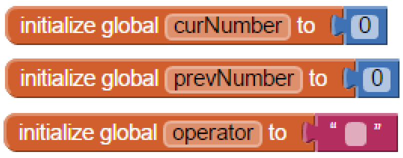
- First number;
- Operator;
- Second number;
- Equal sign to evaluate the expression.

This section is subdivided now in the different parts that we need to handle.

5.1 Declaring the variables

As discussed before, we will need three different variables, two of them to store the numbers and another one for the operator. Let's create these variables now.

- Build a number variable first. Click on *Variables* inside the **Built-in** blocks category. Select the *initialize global to* block, drag and drop it on the blank area of the screen. Click over *name* inside the block, and rename your variable. Remember, we need three variables, one for the **current number**, another one for the **previous number**, and the last one for **operator**. Give meaningful names to your variables, this is as important as the meaningful names of the design components. Now we need to initialize the variable to 0. On the *Math* subcategory select the first block, the one with a 0 inside it. Drag and drop it connecting this block to the previous one, like in a puzzle. You will see they will connect. Since the initial value of the number is 0 we don't need to change the value of the block. That's it, you created your first number. Repeat this procedure for the other number, and for the operator. The only difference for the operator is that you are not going to initialize it with a 0, but with a blank text. The text block can be found inside the *text* subcategory. At the end you should have something like:



```
initialize global curNumber to 0
initialize global prevNumber to 0
initialize global operator to " "
```

Figure 5: Variables declaration

5.2 Number input

Since we have our variables ready, now we need to handle the number input process. Think about it: when you type 1 and 2 in a calculator you expect it to show 12, right? But how exactly should we do this? If we type 1 and 2 without any processing it will show 1 and after that erase and show only 2.

Consider we have a variable called `curNumber` (stores the current number we are working with), that is initialized with 0, and another variable called `typedNumber`, that represents the number that is being typed at the moment.

If we want to obtain 123 for example, we should do:

- `curNumber` = `curNumber` (0) x 10 + `typedNumber` (1) = 1
- `curNumber` = `curNumber` (1) x 10 + `typedNumber` (2) = 12
- `curNumber` = `curNumber` (12) x 10 + `typedNumber` (3) = 123

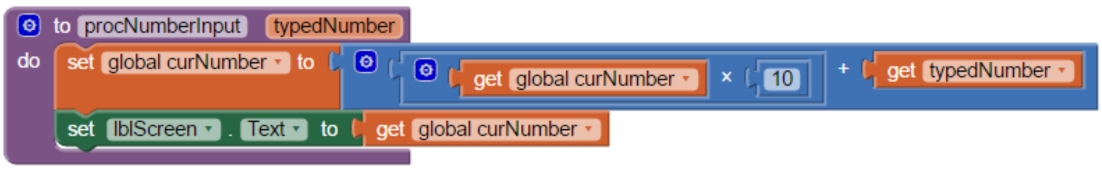
Applying this method we guarantee that for each new input number we will have the correct and desired end value.

Thinking now on our block programming style, we have to apply this method for each one of the 9 number buttons. Thankfully, in App Inventor, we can create something called **procedure**. Basically, a procedure is a piece of code that we can use later, so we create this method just once and can apply it later for as many other components as we want.

Let's create a procedure for our number input, try follow the steps below, use the image at the end to help you understand how it should be at the end.

1. Click on the *procedures* subcategory and drag and drop a *to procedure do* block into the screen, rename it as you wish, I renamed mine to `procNumberInput`;
2. We need an argument to our procedure, that is the number that is being typed by the user. To add an argument click on the little blue gear on the procedure block, drop and drag an *input* block inside the *inputs* block you will see on the right side;
3. Now we will create the mathematical method we discussed before. Under *variables* select a *set to* block, grab and drop it inside the procedure. On the drop down menu inside the block choose the variable you want to use, in my case it is `curNumber`;
4. Now under the *math* subcategory you should add a multiplication block inside a sum block, and attach it to the *set global `curNumber` to*. The multiplication should be between the `curNumber` and 10, and sum this result to the input of the procedure, named on my code as `typedNumber`;
5. Last thing is to show the final number on the display of our calculator. Click over your screen component, in my case `lblScreen`. Select a *set `lblScreen.Text` to* block and drag and drop it below the previous line of blocks. Attach another *variable* block called *get* to it, and select your current variable to it (`curNumber`).

Your procedure is done! Since we are working with blocks, I'm aware this is really confusing. So check the image below as it can help you understand the whole procedure in a better way:



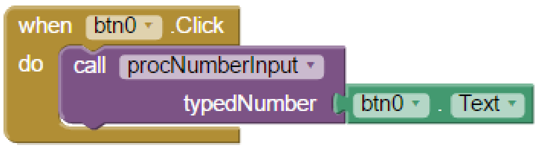
```
to procNumberInput typedNumber
do  set global curNumber to ( get global curNumber × 10 ) + get typedNumber
    set lblScreen . Text to get global curNumber
```

Figure 6: Number input procedure

As we discussed moments ago, we will need this procedure to executed when any of the 10 number buttons are pressed. Let's do it together for one of them, after that you should replicate this code to the other 9 left.

1. Click on the desired component button, in my case `btn0`. Select a *when* ***btn0.Click*** *do* block;
2. Click on the *procedures* subcategory and select our recently created `procNumber Input` procedure, drag and drop it inside the button when block;
3. Last thing is pass the paramenter for our procedure. Under the button component, select the `btn0.Text` block, this will take the number and pass it as a parameter to our already discussed procedure.

You should have:



```
when btn0 .Click
do  call procNumberInput
         typedNumber  btn0 . Text
```

Figure 7: Button number input

Now repeat these steps for the other 9 buttons.

5.3 Operators input

We already have our variables and our number input, now we need to handle when an operator is pressed.

Let's think about what happens when an operator is typed. At the moment one of the four operators is typed we should:

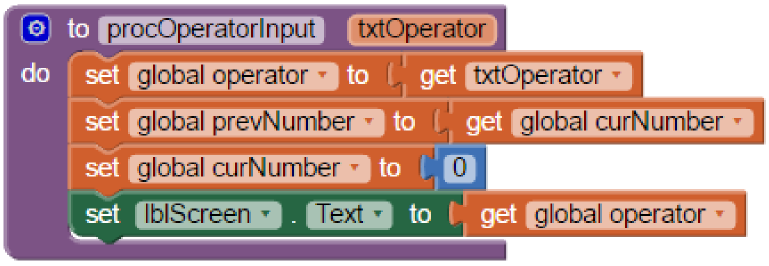
- Store in the operator variable the operator that was selected by the user;
- Copy the stored current value variable to our previous number variable, since after the operator a new number will be given as input, that's why we need to keep the old value obtained stored somewhere;

- Display the typed operator on the screen of the calculator.

Same principle we used in number input applies here. We can create a procedure to handle the operators input and call it for every operator button. That's what we are going to do. Since you already know how to create a procedure with arguments, do the following:

1. Create a procedure with one argument. I named my procedure as `procOperator Input`, name yours as you wish. My argument is called `txtOperator`;
2. Set the `operator` variable to receive the procedure input value (`txtOperator`);
3. Set the previous number variable to receive the current number variable stored value;
4. Reset the current variable value to 0;
5. Display the operator value on the screen of the calculator.

This is your second procedure! At the end you should have something like:



```
to procOperatorInput txtOperator
do  set global operator to get txtOperator
    set global prevNumber to get global curNumber
    set global curNumber to 0
    set lblScreen . Text to get global operator
```

Figure 8: Operator input procedure

Now you should connect this procedure to each one of the four operator buttons. Let's do it together for the plus button:

- Click on the desired component button, in my case `btnPlus`. Select a *when btnPlus.Click do* block;
- Link your recently created procedure with it;
- Pass the `btnPlus` text as parameter of the procedure.

You should have:

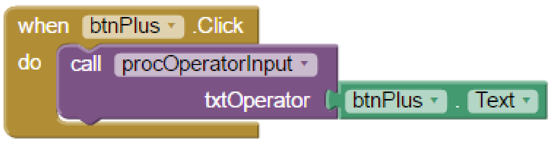


Figure 9: Operator input

Repeat these steps for the other 3 remaining operator buttons.

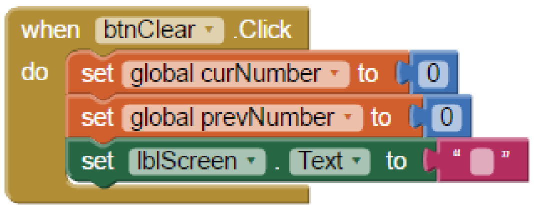
5.4 Clear button

We have just two more parts to program: the clear and the equal buttons. Let's start with the clear.

Bascially when the user hits clear we should:

- Set the current number value to 0;
- Set the previous number value to 0;
- Clean the screen.

Program the steps above inside your clear button click event. You should have:



when btnClear .Click
do set global curNumber to 0
set global prevNumber to 0
set lblScreen . Text to " "

Figure 10: Clear button

5.5 Evaluating the expression

So far we have the two numbers and the operator stored. So what is left is evaluate the expression when the user hits the equal button.

When the user clicks on the equal sign button we should identify which operation should be executed, execute it and then show the result on the calculator display. That's what we are going to do now.

Do the following:

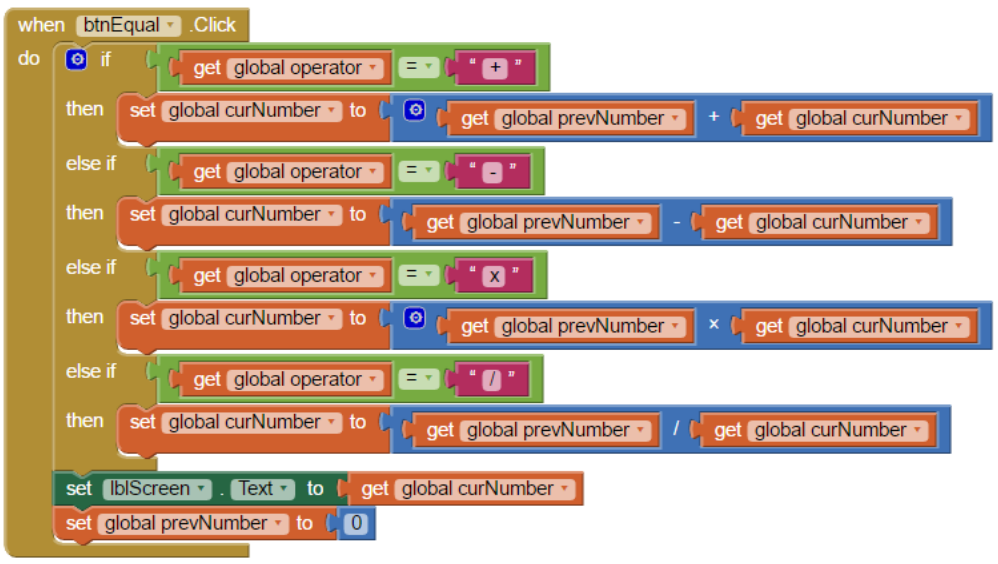
1. Just as you did with the other components, call the click event for the equal button (`btnEqual` in my case);

2. Inside it we should add an **if statement**. The *if* block is located inside the *control* subcategory. Since we are dealing with four operations we should add three *else if* statements to it. To do so click on the little blue gear in the *if* block, select an *else if* and drag inside the *if.* Do this three times;

3. Each one of the cases will be responsible for one operation. We will use an *logic* block to compare the values for each case. In the first case for example we compare the value stored on the variable `operator` with the text "+". If the operator is +, then you will set the current number variable (`curNumber`) to the sum of the previous number (`prevNumer`) with the current number (`curNumber`). Do this for each one of the *if* cases, changing the operator that is compared and the operation executed. Remeber, the operations block are located inside the *math* subcategory;

4. Last thing to do is to display the result. After the *if* block, but still inside the *when* block, set the calculator display to show the value of the current number variable and reset the value of the previous number variable to 0.

This is the toughest block so far, but not too complicated. After following the steps above you should obtain something like this:



```
when btnEqual .Click
do  if      get global operator = " + "
    then    set global curNumber to   get global prevNumber + get global curNumber
    else if get global operator = " - "
    then    set global curNumber to   get global prevNumber - get global curNumber
    else if get global operator = " x "
    then    set global curNumber to   get global prevNumber × get global curNumber
    else if get global operator = " / "
    then    set global curNumber to   get global prevNumber / get global curNumber
    set lblScreen . Text to   get global curNumber
    set global prevNumber to   0
```

Figure 11: Evaluating the expression block

That's it! This is all we need to build a basic calculator that evaluates simple operations with two numbers and one operator. Connect your project with you Android device through the MIT AI2 Companion app and give it a try!

6 Exporting your project

Once your project is done, you can download it to your computer and even generate a file called **apk** to share your work and install it in other Android devices.

To download a project file that you can keep and upload to App Inventor again when necessary:

- Inside the desired project click on **Projects** on the top left of the screen and select *Exported selected project (.aia) to my computer*. This will download an exported file of the project that you can keep stored on your hard drive as backup.

To build your project and create the apk file:

- Inside the desired project click on **Build** on the top left of the screen and select *App (save .apk to my computer)*. This will build your program and download the .apk after it's done. You can share via Gmail this .apk to other people's Gmail, so they can download it and install your app on their Android devices.

That's it! You just finished developing your first Android application! Of course this is a really simple project, but if you're interested in it there are a lot of material on line about App Inventor and Android development as well.

If you want to improve a little bit your calculator go to the end of this tutorial to check an exercise that is provided for you. Good luck!

7 Reference

This tutorial was all based on a really good series of video found on Youtube while I was doing my researches about App Inventor. You can check the first of a four video series about how to build this calculator here: `https://www.youtube.com/watch?v=TMRlu3LfCtE`.

Now try for yourself!

You had just finished your basic calculator, good job!

What happens when you type the following expression: 2 + 2 - 1? Does it calculate properly if you don't type the = sign?

Can you think about a way to solve this issue?
Here are some hints to guide you on that:

- *You should modify the previous created operator input procedure.*
- *If the variable of the previous number is different than zero when a new operator is pressed it means that the expression has more than one operator.*
- *If the case above happens for each new operator typed you should immediately evaluate the expression so far.*
- *The evaluated sub expression should be stored as the new previous number of the outer expression.*

Good luck!