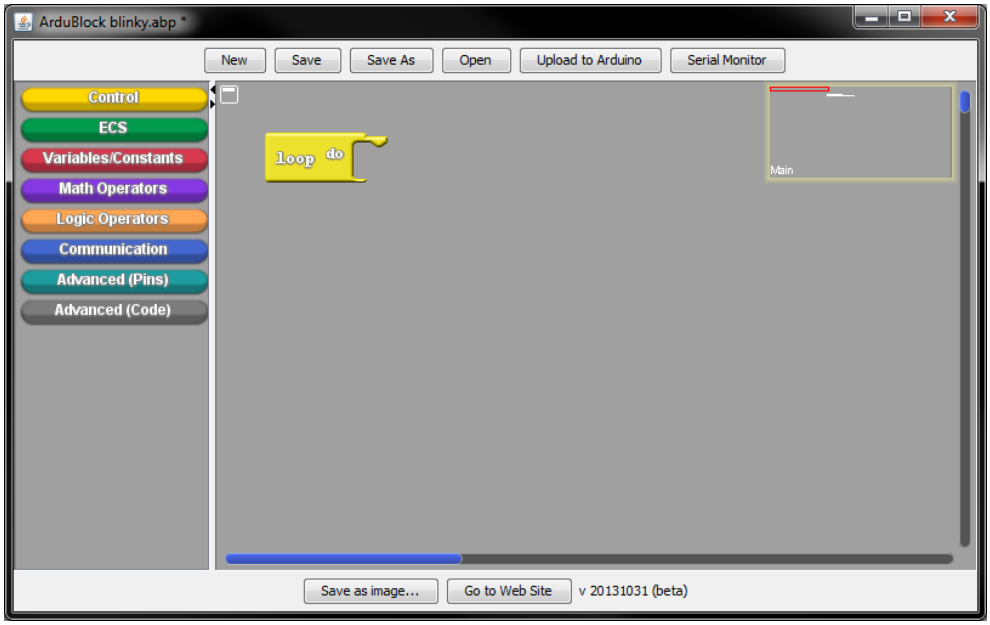
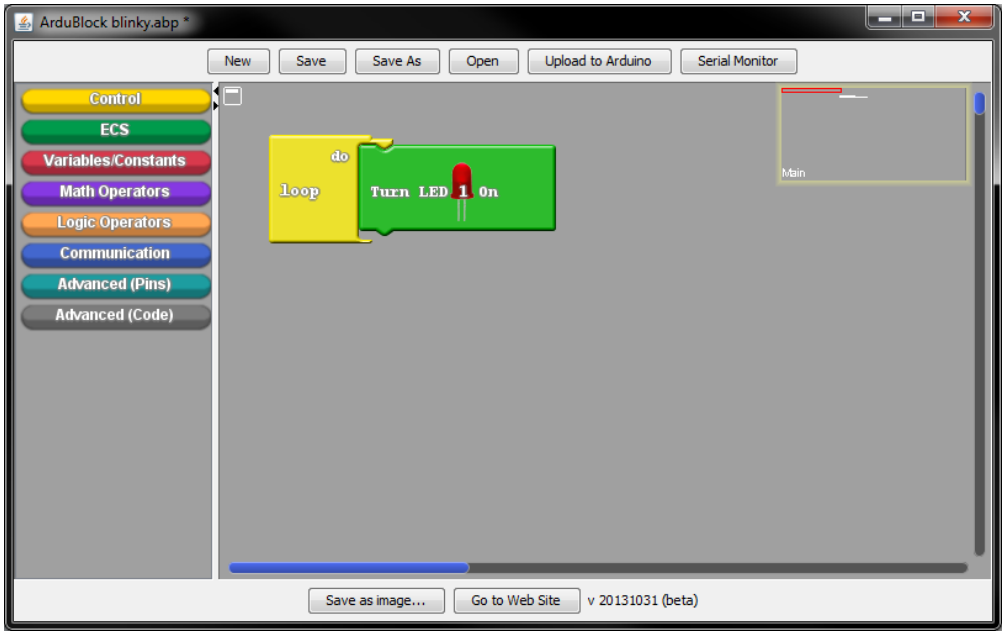
Using the Lights

1. Place a **loop** block, found in the **Control** drawer. We want our program to loop over and over again until we turn the power off in order to match the sample program we just saw.

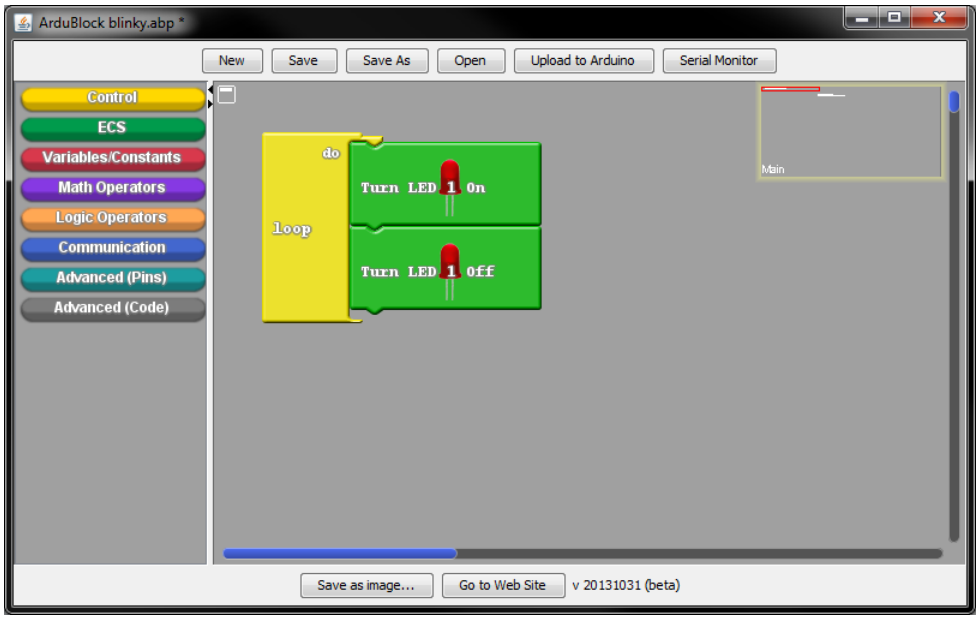


2. Place a block to **Turn LED 1 On** inside of the **loop** block. This can be found in the **ECS** block drawer, and will turn the LED on.

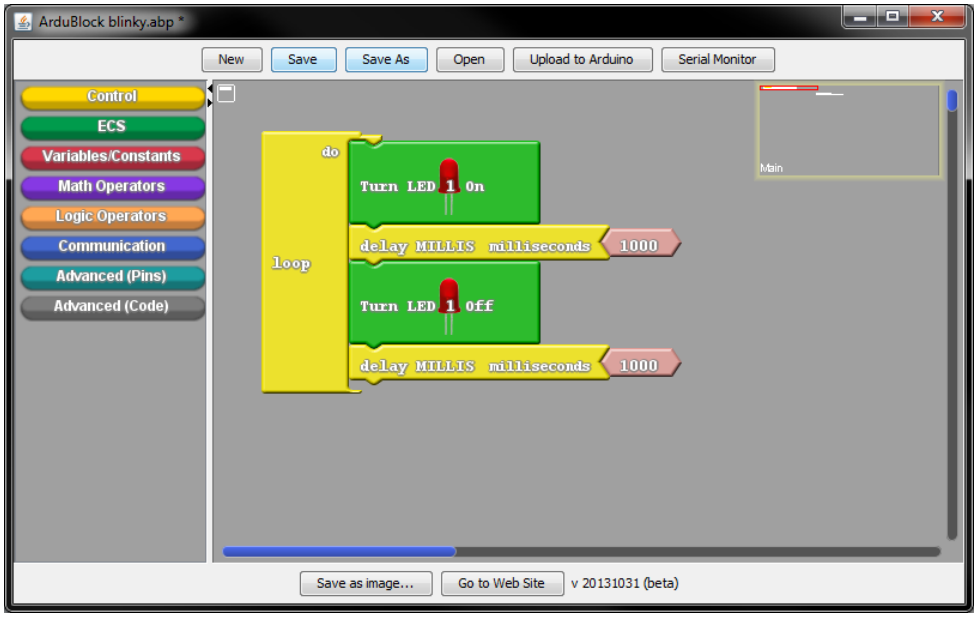


3. Click the *Upload to Arduino* button at the top of the program to load your program to the board. When you reset the board, you will see that the first (red) LED is turned on. This is a good start, but we want our LED to blink on and off, not just stay on.

4. In order to blink the LED, we need to turn it off after we turn it on. Place a **Turn LED 1 Off** block after the **Turn LED 1 On** block inside of the **loop**.



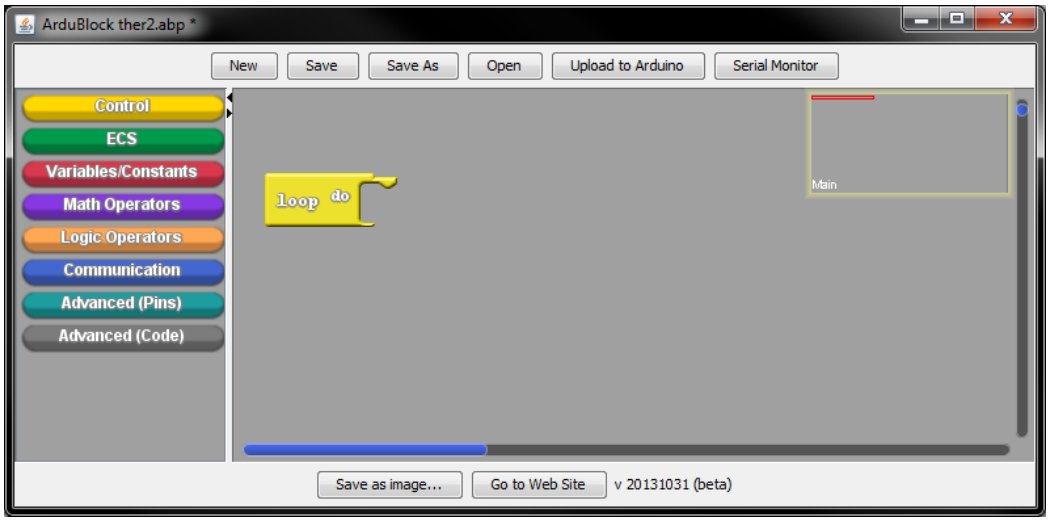
5. It may seem like this is the entire program. Let's give it a try and see how it runs. Just as before, click the *Upload to Arduino* button at the top of the program to load your program to the board.
6. Does the program behave the way you expected it to? What happened when you ran the code? Though it seems finished, our program will turn the LED on and then off so quickly that we won't be able to see what is happening. Place a **delay MILLIS** block (which can be found in the **ECS** drawer) after the **Turn LED 1 On** block. Place another after the **Turn LED 1 Off** block. These will make the program wait in between blocks, with the default wait time set to 1 second.



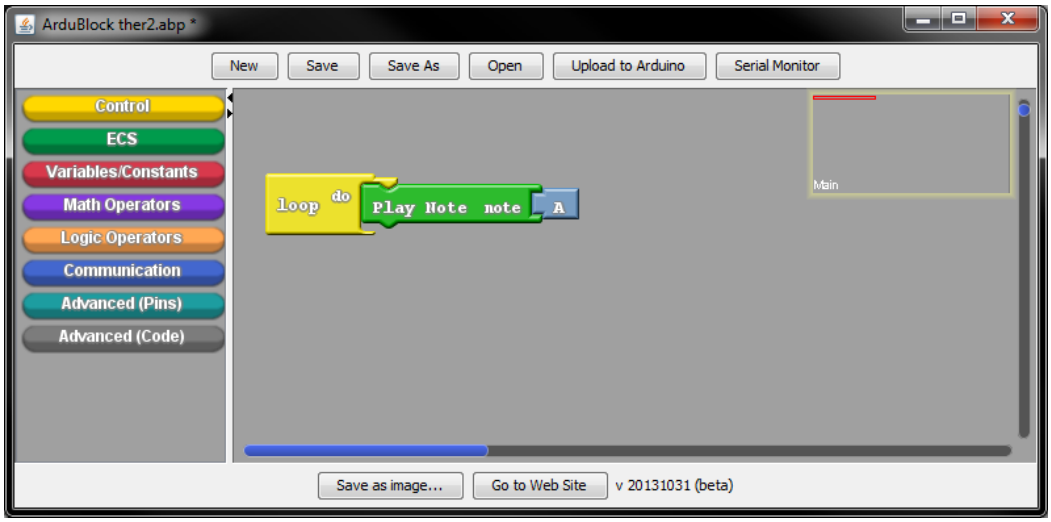
7. Run the program again to see how the delays have changed it. You will see that we now have a program which blinks the red LED on and off forever, until the power is turned off. Congratulations on completing your first Ardublock project! If there is still time left, explore the other LED blocks found in the **ECS** drawer and use them, along with changes to the delay pattern and lengths, to make your own variation on this project.

Playing Notes

1. Because our programs have access to a speaker, they are able to play series of musical notes while they run. Like last time, begin by placing a **loop** block, found in the **Control** drawer.

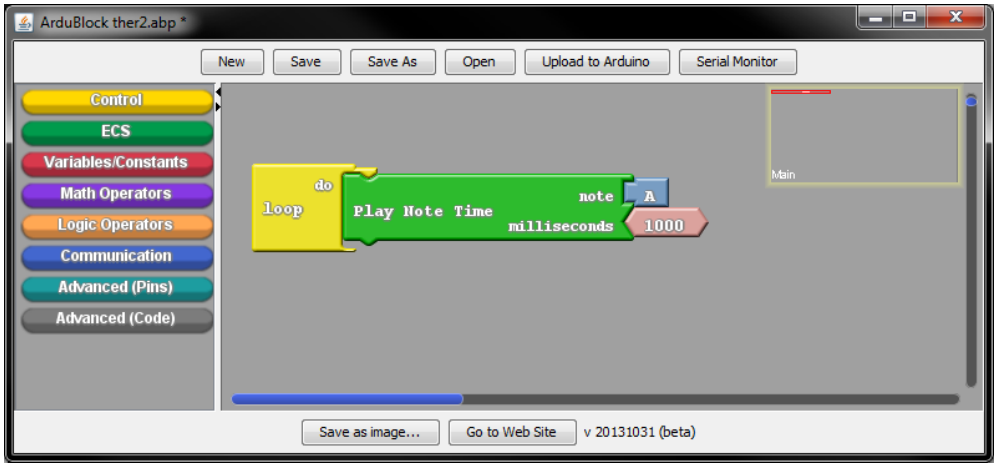


2. Place a **Play Note** block from the **ECS** drawer inside of the **loop**. Leave the note as default, A.



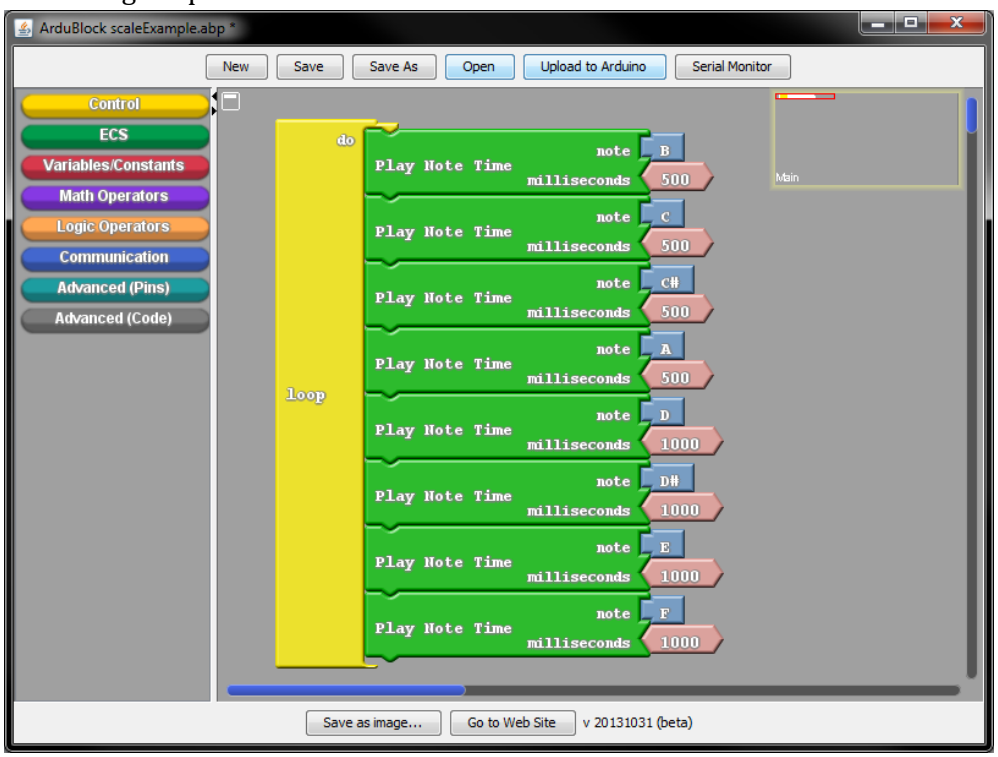
3. Click the *Upload to Arduino* button at the top of the program to load your program to the board. When you reset the board, you will hear an A note playing forever. This isn't much of a song, but we'll add to it throughout this tutorial.

4. We can change the note which plays by modifying the **Note** block connected to the right side of our **Play Note** block. Please use the handout of available notes, which your teacher can provide for you. The **Play Note** block is great for continuously playing a tone, but we want to create a melody. Remove the **Play Note** block and replace it with **Play Note Time**, also found in the **ECS** drawer. This block plays the attached note for as long as the attached time says. By default, it plays an A note for 1000 milliseconds, which is the same as 1 second. Go ahead and try running your code again to hear the difference.



5. As you can see from the notes handout, our programs have access to standard notes and sharps, and can play notes from a range of octaves. We can also specify specific frequencies as numbers if we wish. To do so, use a **Play Frequency** or **Play Frequency Time** block, both of which are found in the **ECS** drawer. They work in the same way as **Play Note** and **Play Note Time**, but take numeric frequencies as inputs instead of letter names for notes.
6. Add notes to your melody to create a scale, as seen below. You will have the chance to create your own song using the **note** and **delay** blocks, but let's start with

something simpler.

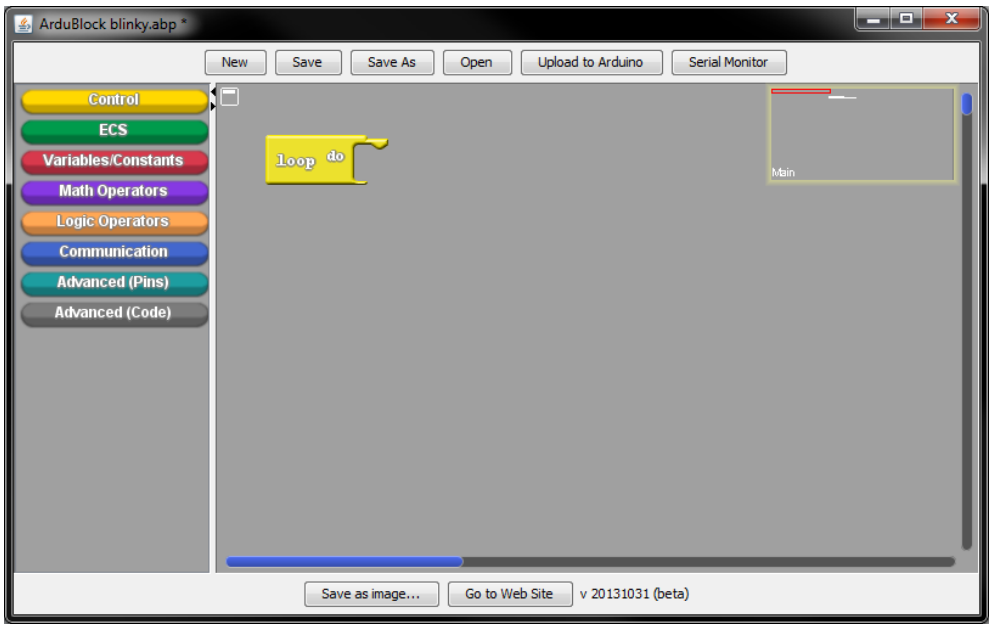


7. Run the program again to hear how it works. You will hear a basic chromatic scale, repeating over and over again. If there is still time left, explore the **note** and **frequency** blocks found in the **ECS** drawer. You can combine these blocks with the **LED** blocks from the previous tutorial to make a light pattern go along with your scale, or add **delay** blocks to insert rests between your notes.

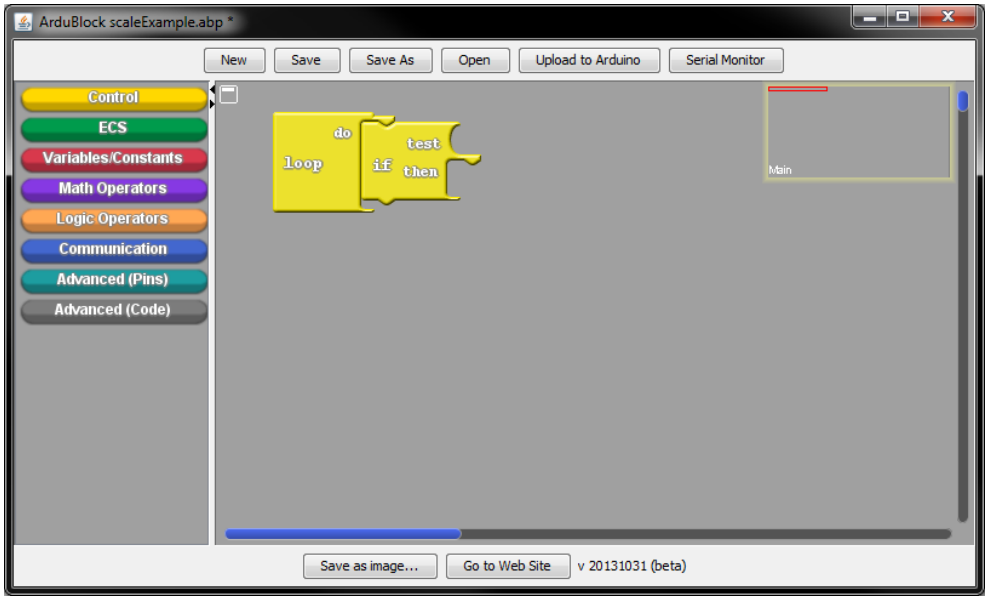
Using the Buttons

1. In this tutorial, we will learn to use the board's buttons to control what our programs do. As in the other tutorials, begin by placing a **loop** block, found in the

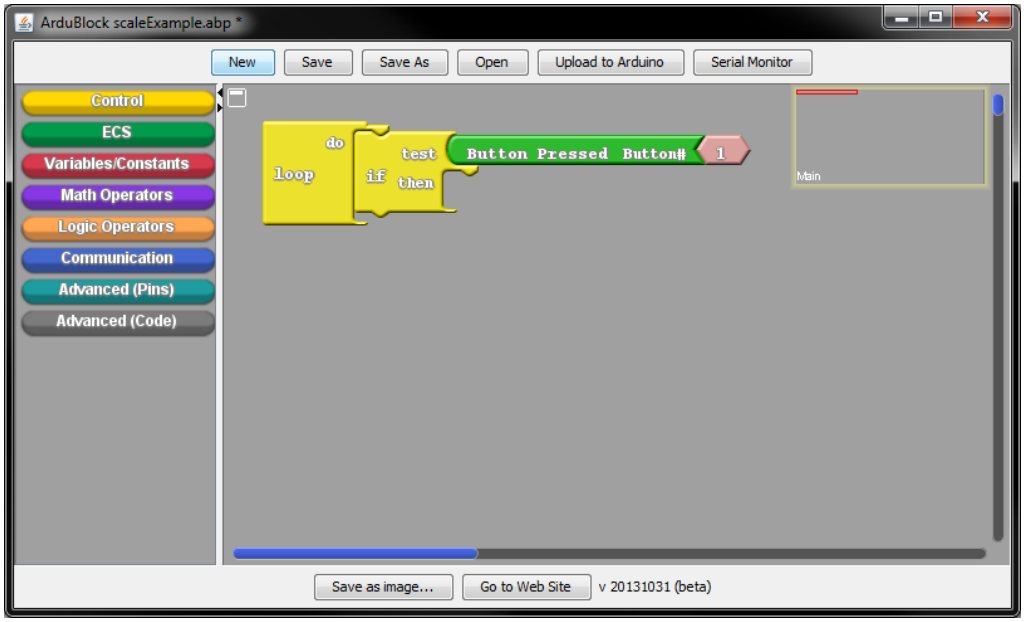
Control drawer.



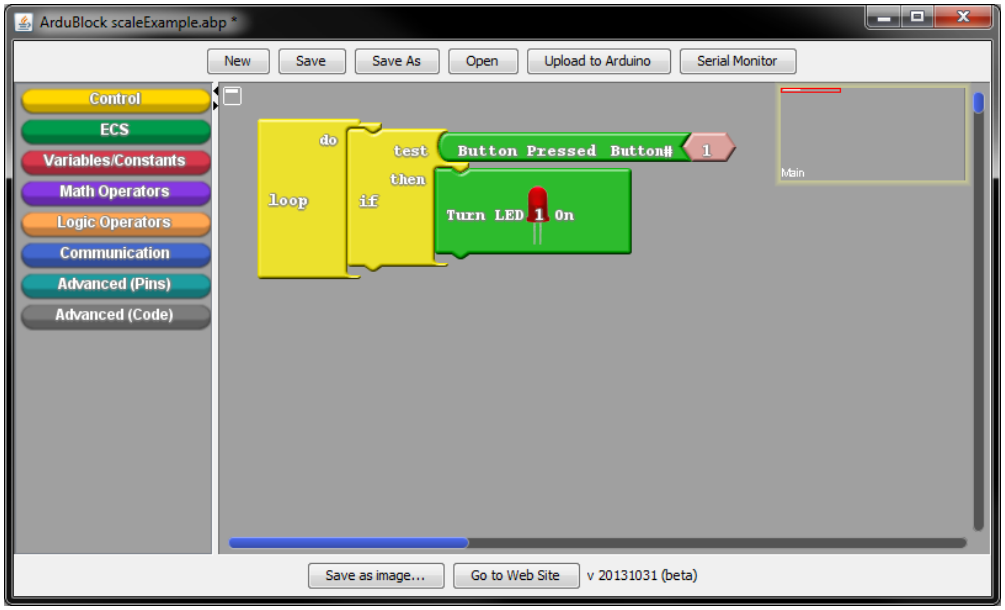
2. Place an **if** block, found in the **Control** drawer into the **loop**. This will determine whether the *test* in the top of the block is true and execute the blocks in the *then* socket if it is.



3. Let's start by making the **if** check to see if the button is pressed. Place a **Button Pressed** block, found in the **ECS** drawer, inside the *test* socket of the **if** block. This block will return a true value only while the button is pressed, so whatever we place in the *then* socket will execute when we press button 1.

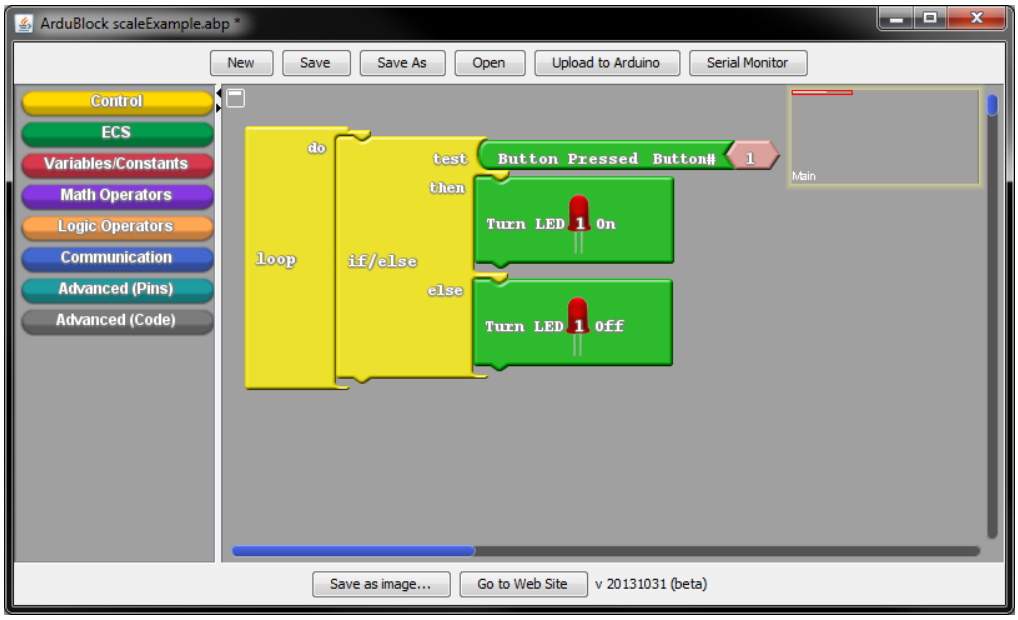


4. Place a **Turn LED 1 On** block into the *then* socket.



5. Run the program to test how it works. When you press button 1, you will see the LED light up. The main problem with this program right now is that the LED stays on forever, so we can only test it out once without resetting.

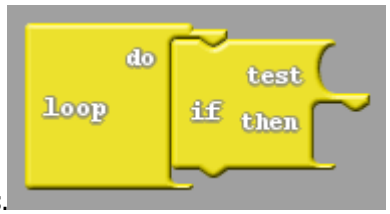
6. Replace the **if** block with an **if/else** block and place a **Turn LED 1 Off** block in the *else* socket. This will execute if the button isn't being pushed, so our new program will only light the LED while the button is pushed.

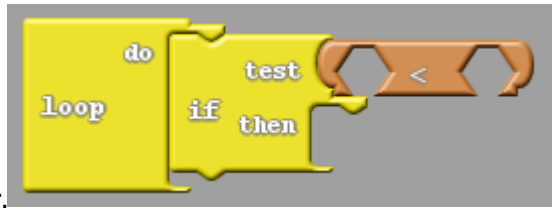


7. Run the program again to test this. Now that you have a program which executes one set of blocks if the button is pushed and another set if it isn't, you can use the other available blocks to make your own variation on this program. Consider using multiple **if/else** blocks to use the multiple buttons available on the board. Have fun!

Using the Distance Sensor

1. In this tutorial, we will learn to use the board's distance sensor to control what our programs do. As in the other tutorials, begin by placing a **loop** block, found in the **Control** drawer. 

2. Place an **if** block, found in the **Control** drawer into the **loop**. This will determine whether the *test* in the top of the block is true and execute the blocks in the *then* socket if it is. 

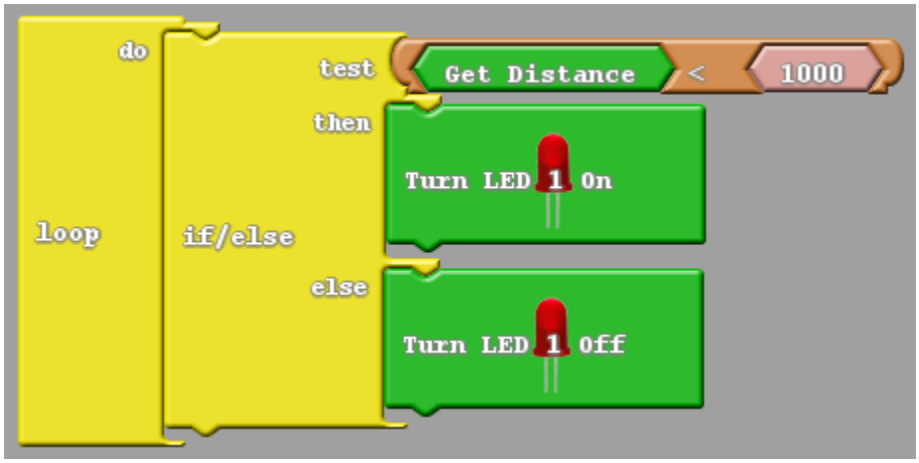
3. Because our distance sensor block, **Get Distance**, returns a number for how far it is reading, we will be comparing its value to a constant value. To do this, place a **Less Than (<)** block in the **if** block's test field. You can find the **Less Than (<)** block in the **Logic Operators** block drawer. 

4. Place a **Get Distance** block (found in the *ECS* block drawer) in the left side of the **Less Than (<)** block and a constant 1000 in the right side. This 1000 value is fine for use in this tutorial, but you will need to fine tune the constant to which you compare for an actual project.The constant number block can be found in the **Variables/Constants** block drawer, and is set to 1 by default.


5. Now that we have the basic frame of our program, we can make it do something. Place an **LED 1 On** block in the then field of the **if** block. Run your program and wave your hand above the shield. The distance sensor will recognize that an object is close to its sensor and turn the light on. You've officially made your first project

with the distance sensor!

6. Our program is already interesting, but we can do better. To make the LED turn off again when your hand is moved out of the distance sensor's beam, change your **if** block to an **if/else** block and put a **LED 1 Off** block in the then field. Test your program and see how it works. Can you think of any ways to improve this design and make it do something more interesting?

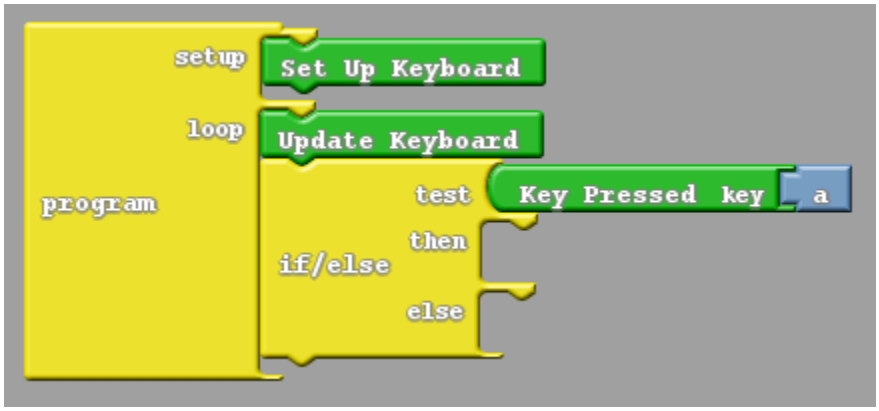


Using the Keyboard

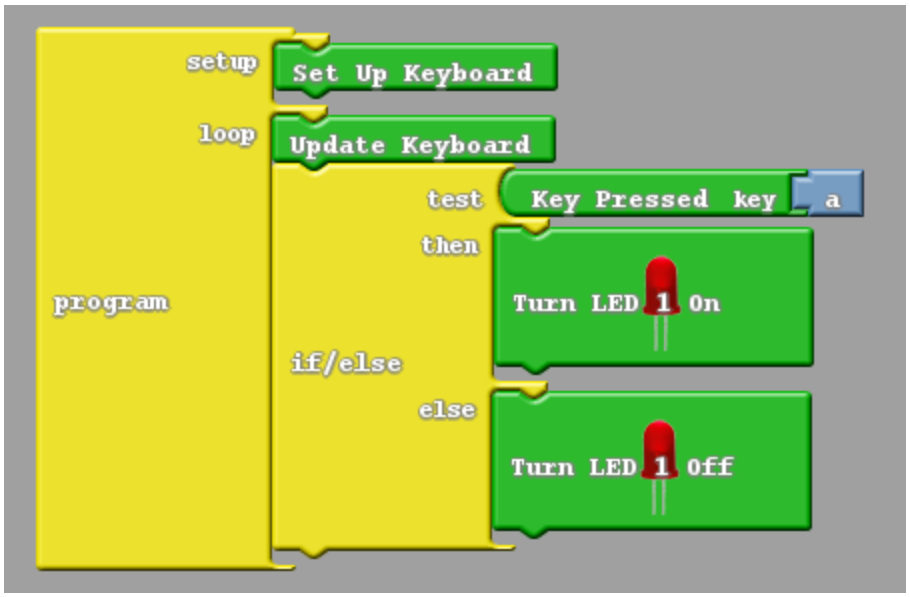
1. In this tutorial, we will learn to control what our programs do on the Arduino using input from a keyboard. As in the other tutorials, you need to start by placing your first block. Unlike the other tutorials, though, where you used a **loop** block which repeated its code forever, this tutorial will use a **program** block, also found in the **Control** drawer. This block works similarly to **loop**, but its setup code will run once before the loop begins. Inside of the setup field, place a **Set Up Keyboard** block, found in the **ECS** block drawer. This block must be placed at the beginning of the setup field of any program which uses the keyboard.

2. Place an **Update Keyboard** block at the beginning of the **program** block's loop field. This block updates the communication between the keyboard and the Arduino and must be used whenever a fresh value should be read.

3. Now that we have the two setup blocks in place, we are ready to start reading from the keyboard. Place an **if/else** block beneath the **Update Keyboard** block so we can do one thing if the key we are looking for is pressed and another if it is not. Also place a **Key Pressed** block in the test field of the **if/else**, allowing us to test whether the 'a' key on the keyboard is being pressed.



4. As we have done in other tutorials, place an **LED 1 On** block in the then field of the **if/else** block and an **LED 1 Off** block in the else field. This will turn the first LED on while the 'a' key is pressed and turn it off the rest of the time. Despite providing many options for controlling your programs, using the keyboard is that easy. You're ready to upload your program to the Arduino and try it out.



5. When your program runs, a window labeled “ECS Keyboard” will pop up on your computer screen. Keys typed into this window will be sent to the Arduino when the **UpdateKeyboard** block is used. If you have time, try testing for other keys or making other outputs result from the keyboard input.