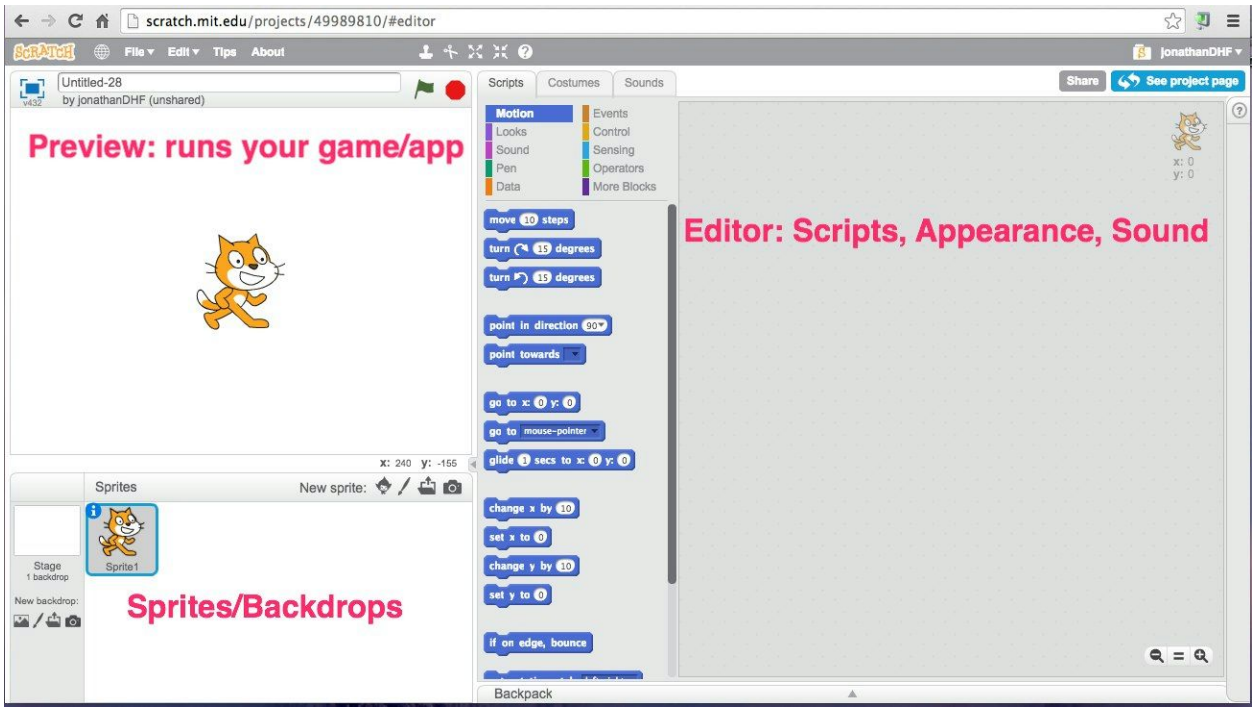
SCRATCH

Overview of Interface:



Preview - This is a preview of your game/app. Click the **Green** flag to begin running your game/app. Your code scripts will begin running and you can use this to actively preview and troubleshoot. This also serves as a playthrough of your project!

Editor - This is where you edit your scripts, open the appearance editor for sprites and backdrops, or edit your sound files.

Note: You can switch between the *Scripts*, *Costumes*, and *Sounds* tabs to access different things.

Stage - This is the backdrop of your game and can be changed using the *switch backdrop to…* block in Looks

Sprites - These are the characters/objects that act or interact with each other. Each sprite has its own code and can have several different *costumes* to change between. Costumes can be changed with the *switch costume to* block in Looks

- Changing costumes is used to animate Sprites and can give the appearance of walking

Block Categories:

Motion - The **motion** blocks are used to *move* sprites, change which direction they face, and move sprites directly on the x/y plane. There are also blocks for causing *bounce*, which is used in many types of game.

Looks - The **looks** blocks are used to change the appearance of sprites and environments. This section also contains blocks used to switch costumes and backdrops, scale sprites, show/hide sprites, and many other useful appearance related blocks.

Sound - The **sound** section contains blocks that play different audio. You can either use the sound library or upload your own sounds.

Pen - The **pen** section contains blocks related to the use and style of a pen in your project.

Data - The **data** section contains blocks that allow for the addition of variables that can be used to track different events or display a score on the screen. You can create your own variables and lists in Scratch. Anything created here is stored for use in your project.

Events - The **events** blocks are used to trigger scripts on actions such as *key press*, *click*, and *backdrop changes*. This section also contains broadcast and listener blocks, which are used for sending/receiving global signals across your project.

Control - The **control** blocks contain conditional and loop blocks. If/then statements and other logic operations are in this section.

Sensing - The **sensing** blocks detect certain events such as collisions between sprites, as well as checking for user input and tracking mouse position.

Operators - The **operators** blocks allow mathematical operations on variables/numbers. This section also contains the random number generator, which is useful for randomizing motion.

More Blocks - Allows for the creation of your own commands.

Tips and Tricks

Stack Blocks

Stack blocks are rectangular blocks shaped to fit above and below other blocks. The majority of blocks in Scratch are *Stack blocks*, as they're available in every category except **Operators**. Creating connections between these blocks creates **scripts**. When you create a script, the commands run from top to bottom.

The general shape of a stack block:

This allows for other blocks to be connected to the notches above and below, creating a series of commands known as a script.

Starting Your Code

When writing your code scripts you'll need to be able to start different scripts at certain points. If you don't put one of these pictured blocks at the top of a grouping of blocks, the code won't know when to run.

You'll notice that the blocks have different types of joints.These Event blocks all share the same 'notch' that can be connected to other blocks.

If you put code under the *when_key pressed*, the linked code will execute any time the specified key is pressed.

This example code segment would be placed inside a sprite. The sprite would then *move 10 steps* when the Green flag (start) is clicked.

Simple Loop

Loops are used to repeat lines of code. Repeat will run whatever is inside of them a set number of times. In the first example, the *move 10 steps* will repeat 10 times, moving 100 units total (10 steps x 10 repeats).

Forever loops will run until they are told to stop, either by another code block or by clicking the stop sign. The example forever loop would cause the sprite to continuously move until the game stops, or told otherwise.

Creating Clones

Clones can be useful when you don't want to have to create (and code) multiples of the same sprite. Clones also need their own code to behave, which is handled with the create clone of myself block. After the clones are created, their behavior is handled with the when I start as clone block.

This clone code inside the *forever loop* will spawn a new clone of itself every second. The

when I start as a clone block then gives it the following behavior: Each clone will move forward 10 units, wait 1 second, turn, and then delete itself.

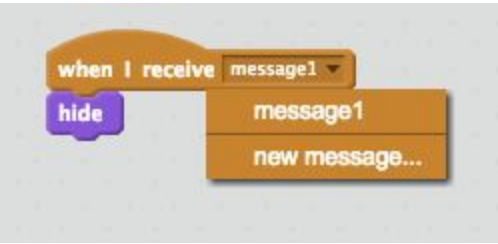
It is important to use the delete this clone block instead of hide when you want to remove a clone, as the clone will still count toward the max clone count of 300. It will also continue to count as an additional Asset, slowing down performance.

Broadcasting and Listening

Broadcasting sends a message across the entire program and is used to start lines of code in different sprites, allowing for communication between lines of code.

How to Listen: Any sprite that is *hatted* with the *when I receive…* block will activate when the specific message is broadcast. This is essential to have sprites communicate across the entire program.

In the above example, when the *space key* is pressed the sprite will wait one second and then broadcast 'message1' to all sprites, across your entire program. If any of the sprites have a starting block (hatted) with *when I receive [message1]* they will execute.



In this case, the sprite will hide itself when receiving this message.

Note: The sprite sending the message can be completely independent from the sprite receiving the message.
Broadcasting is useful if a script needs to be activated *after the start of the program without a user prompt* such as a keypress or mouse click.

Create A Variable

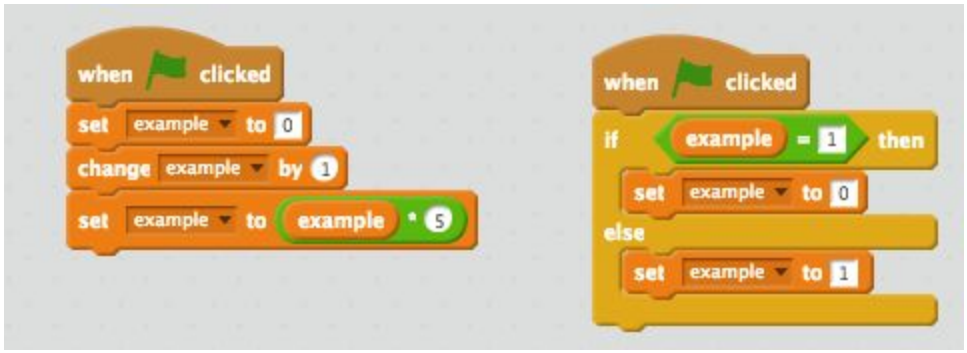
Variables are contained in the **Data** category and can be used to track and trigger events.

Variables can be used to display a score, healthbar, or time elapsed to the player. Variables can also be used to track calculations inside code.

Variables are a crucial component of coding and becoming familiar with how to use and manipulate them will provide a solid foundation for any future programming.

Whether or not a variable is displayed depends on if the check mark next to it in this view is checked. In the above example, you can see a check next to the variable named 'example.' Since it's checked here, it will be displayed as a visible bar on the project:

Variables can also be used for calculations and for interacting with *if* statements. In the following example two separate scripts are referencing the variable named *example*.



The code on the left changes the value of the variable. It will first set 'example' to 0, then change it by +1. *Keep in mind it is possible to use negative numbers with Scratch*. Then it will set 'example' equal to the current value multiplied by 5. This means that the final value will be 5. The green block is an **operator**, which are commonly used to interact with the data stored in variables.

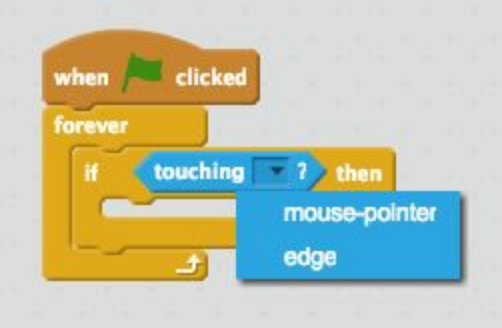
The code on the right is an *if* statement that checks the status of the variable named *example*: If example = 1 then set example equal to 0, *else* the code will set example to 1.

Random Number Generator

Random Number Generators are very useful and are straightforward to setup. All that is needed is a variable (in this case, named *example*) and to use the 'pick random' block.

This selects a number in the specified range. In the sample range above, any of the numbers 1 , 2 , 3 , 4 , 5, 6 , 7 , 8 , 9 , 10 can be assigned to the variable *example*. Each time the space key is pressed, a new number will be selected and assigned to the variable named *example*. Random Number Generators are useful for assigning random values to variables, creating random movement in sprites and many other uses.

Collision Detection



Collision detection is the term used to determine if two (or more) objects are touching.

This is a set of blocks used to detect if two objects are touching each other. A *forever loop* will run and continuously ask itself 'Is this sprite touching ___?'

1. touching_
2. touching color_
3. color_is touching_
4. distance to_

Add basic info about these- don't need too much because there is a lot of content in the Help files in Scratch. We need to show how to get to this point.

You can have any number of things happen when the if statement triggers. You can have a score increase, or hide the sprite, or have the sprites bounce off of each other.

Starting Over

While designing your game you may come across a situation where at the end of the game you want everything to disappear and to show a **WIN** screen. In this situation you

need to make sure that at the beginning of your game you need to have a show command to have the sprite pop back up again.

Inkscape and Scratch

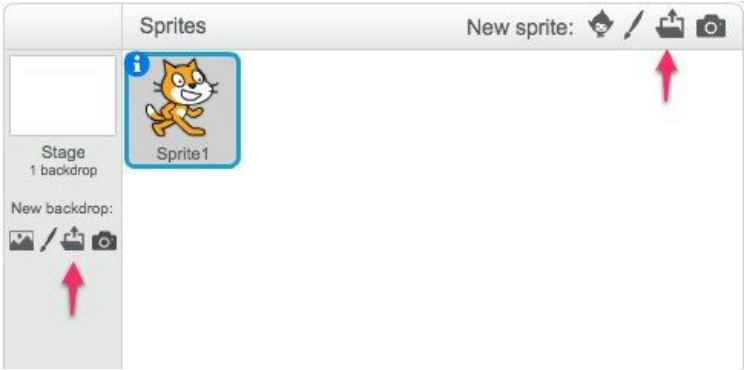
Inkscape is a great tool for creating characters, backdrops, menu/skill bars, and any other necessary sprites. Creating and importing from Inkscape allows for a much higher quality in design, and provides much more control and customization than Scratch's art editor.

One of the best aspects of knowing Inkscape is that you can import your creations into Scratch as **.svg** files. You're also able to import other file formats, such as **.jpg** and **.png**. I prefer to use the .**svg** format.

SVG files are **S**calable **V**ector **G**raphics. Anything that you import from Inkscape can be modified in Scratch's editor. For example, if your imported character is too large and you need to make a quick fix, you can scale it in the internal editor.

Steps for Importing Finished Images:

1. Save your file as a .**svg**. Refer back to the lesson in Learn if you need help remembering how to save in Inkscape.
    a. Save the file somewhere easy to access, such as the Desktop. Remember to upload any files you're using to Google Drive!
2. Find the **upload** icons in the bottom left corner of your Scratch program. You'll notice there is one for **backdrop** and one for **sprite**.



    a. Choose whichever is appropriate and you'll be prompted to locate the file.
3. Navigate to the file location (such as **Desktop**), select the file, and click **open**.

4. After you do this, the sprite will now appear with either your sprites or as a **Stage** (backdrop).

1.