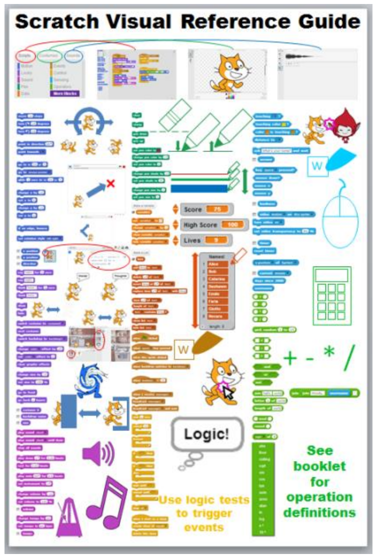
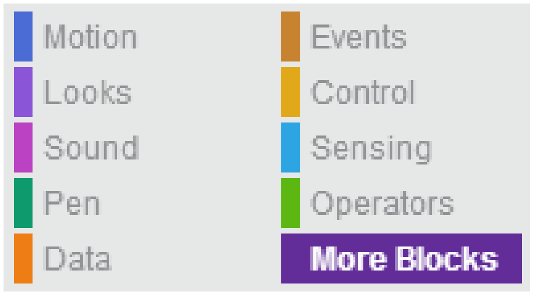
Scratch Documentation Booklet

How to use this booklet:

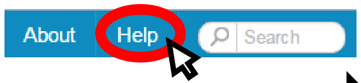

Scratch Visual Reference Guide
Logic!
Use logic tests to trigger events
See booklet for operation definitions

If you know what you want to do, but are unsure how to code it, then look at the Visual Reference Guide (pictured at left). If you know what you are trying to code, but are unsure how precisely the code works, then flip to the color-coded section for documentation on how the different tiles work.

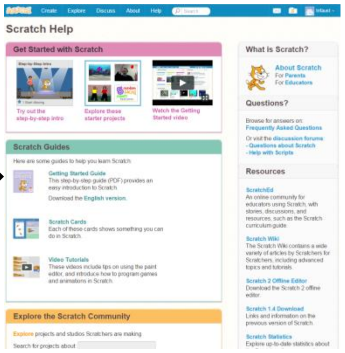


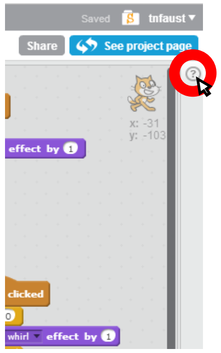
Sections are divided by block type, as shown at left (see tabs below). Each page defines what the blocks do on top, with tips below. There is an index at the back.

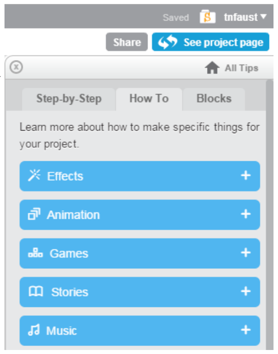
For more information, see Scratch's online Help section, the on-board help menu, or the wiki:



https://scratch.mit.edu/help/







http://wiki.scratch.mit.edu/wiki/

Motion Blocks

Block Definitions

`move (10) steps` Move the sprite a number of pixels in the direction it is facing.

`turn ↻ (15) degrees` Turn the sprite a number of degrees, clockwise.

`turn ↺ (15) degrees` Turn the sprite a number of degrees, counter-clockwise.

`point in direction (90▾)` Points the sprite in an absolute direction (0 is up, 180 is down, etc.).

`point towards [▾]` Points the sprite toward another sprite or the mouse pointer.

`go to x: (-55) y: (-37)` Teleports the sprite to x, y coordinates.

`go to [mouse-pointer ▾]` Teleports the sprite to the mouse pointer.

`glide (1) secs to x: (-55) y: (-37)` Moves the sprite smoothly to x, y coordinates over z seconds.

`change x by (10)`
`set x to (0)`
`change y by (10)`
`set y to (0)`

These blocks change the x value of a sprite’s position, relative to where it is, or set it to an absolute value. The possible range of pixel coordinates is x -240 to x +240, and y -180 to y +180. Sprites do not pass between regions on their way; that is, they “teleport” instead of “walk” (see below).

`if on edge, bounce` This block checks if a sprite is on the edge, and then bounces it off.

`set rotation style [left-right ▾]` This block controls how the sprite rotates (see below).

☐ `(x position)`
☐ `(y position)`
☐ `(direction)`

These blocks can be inserted into other blocks to check and/or change the x/y coordinates of a sprite, as well as their direction (see below).

Motion Blocks
Tips and Examples

Motion and Time:

In reality, objects moving to a different spot need to pass through all spots in between. Sprites don't need to do this, and can instantly teleport to wherever they need to go. This is the main difference between the **move** block and the **glide** block:

```
move (10) steps
```

```
glide (1) secs to x: (-180) y: (0)
```

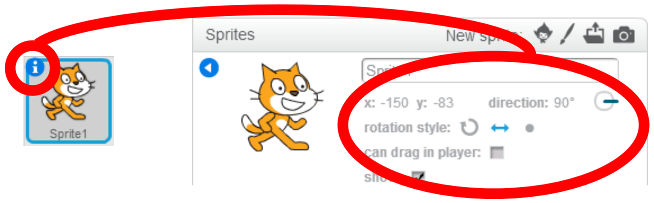
Most **Motion** blocks work instantly, but the **glide** block will move a sprite through in-between points.

The tricky thing about the **glide** block is that it works over time, not at a specific speed. But you can turn time into speed with the **distance to** block (in the **Sensing** palette). Distance is measured in pixels, which are tiny, so you can divide them by a constant number (using **Operators**) to change speed. Try using the blocks below to make one sprite **glide** to another at constant speed, no matter where it starts:

```
distance to [Cat1 ▾]      ( ) / (100)

glide (10) secs to x: (10) y: (10)

[x position ▾] of [Cat1 ▾]      [y position ▾] of [Cat1 ▾]
```

Rotation Style:

Rotation style is a sprite attribute that can be changed in a sprite's info:



Sprite1

Sprites

New sprite:

x: -150 y: -83 direction: 90°

rotation style:

can drag in player:

Build the sample script at right to see how the different rotation styles work in action when you use the **set rotation style** block:

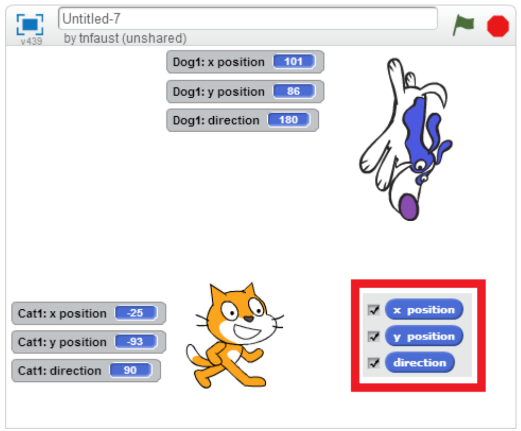
Notice that, inside the **repeat** block, everything is the same. That is, the sprite is doing exactly the same thing, but it looks different because of the **set rotation style** block.

```
when green flag clicked
point in direction (90 ▾)
set rotation style [all around ▾]
say [This is "all around" rotation] for (2) secs
repeat (24)
    turn ↻ (15) degrees
    wait (.25) secs
end
set rotation style [left-right ▾]
say [This is "left-right" rotation] for (2) secs
repeat (24)
    turn ↻ (15) degrees
    wait (0.25) secs
end
set rotation style [don't rotate ▾]
say [This is "don't rotate" rotation] for (2) secs
repeat (24)
    turn ↻ (15) degrees
    wait (0.25) secs
end
```

Position & Direction:

Checking the boxes next to these attributes will show them on the screen (see right). You can use these with the **glide** block and some **Operators** to make a sprite glide smoothly to a relative position, like in the below example:

```
glide (1) secs to x: ((x position) + (50)) y: ((y position) + (50))
```



Untitled-7

by tnfaust (unshared)

v439

Dog1: x position 101

Dog1: y position 86

Dog1: direction 180

Cat1: x position -25

Cat1: y position -93

Cat1: direction 90

☑ x position

☑ y position

☑ direction

Looks Blocks

Block Definitions

say Hello! for 2 secs This block tells a sprite to say something for a number of seconds.

say Hello! This block tells a sprite to say something until it's told to say something else.

think Hmm... for 2 secs The "think" blocks work exactly like the "say" blocks, but they use
think Hmm... thought bubbles instead of speech bubbles.

show
hide These blocks tell a sprite whether it should show itself or hide.

switch costume to costume2 This block tells a sprite to use a specific costume (by name).

next costume This block tells a sprite to go to its next costume, whichever it's on.

switch backdrop to backdrop1 This block tells the Stage to switch to a specific backdrop.

change color effect by 25 This block changes a graphic effect by a relative value.

set color effect to 0 This block sets a graphic effect to an absolute value.

clear graphic effects This block clears all graphic effects on a sprite.

change size by 10 This block changes a sprite's size by a relative value.

set size to 100 % This block sets a sprite's size to an absolute value.

go to front This block sends a sprite to the front of the stage, whatever's in front of it.

go back 1 layers This block sends a sprite back a number of layers behind it.

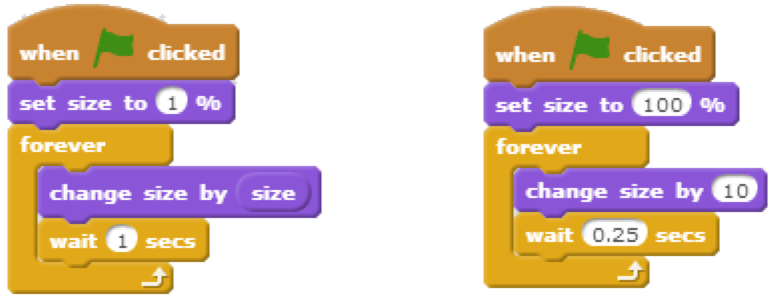
costume # These blocks detect sprite/backdrop attributes, and can be inserted into
backdrop name other blocks as logical tests (see Control/Sensing/Operators blocks).
size

Looks Blocks

Tips and Examples

Build these sample scripts to compare and contrast the effects of different **Looks** blocks.
Don't use them all at the same time (otherwise they'll all go at once and make a big mess). Maybe use different keys to trigger them, instead of the green flag. Can you think of times you might want one over the other?

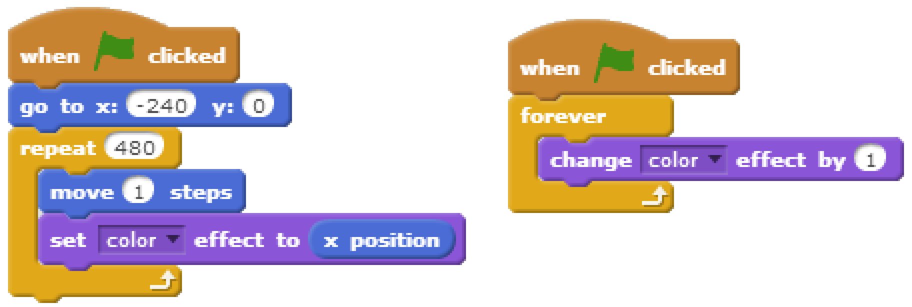
Check the box by the **size** variable to see its value.



```
when green flag clicked
set size to (1) %
forever
    change size by (size)
    wait (1) secs
```

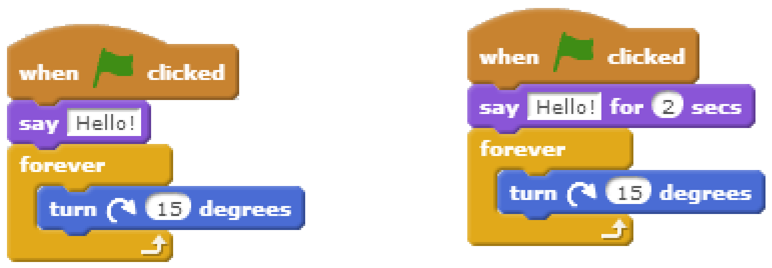
```
when green flag clicked
set size to (100) %
forever
    change size by (10)
    wait (0.25) secs
```

When might you want to change the color overlay of a sprite?



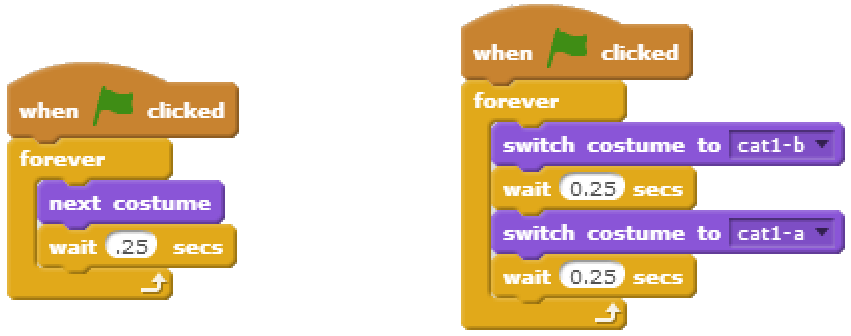
```
when green flag clicked
go to x: (-240) y: (0)
repeat (480)
    move (1) steps
    set [color] effect to (x position)
```

```
when green flag clicked
forever
    change [color] effect by (1)
```



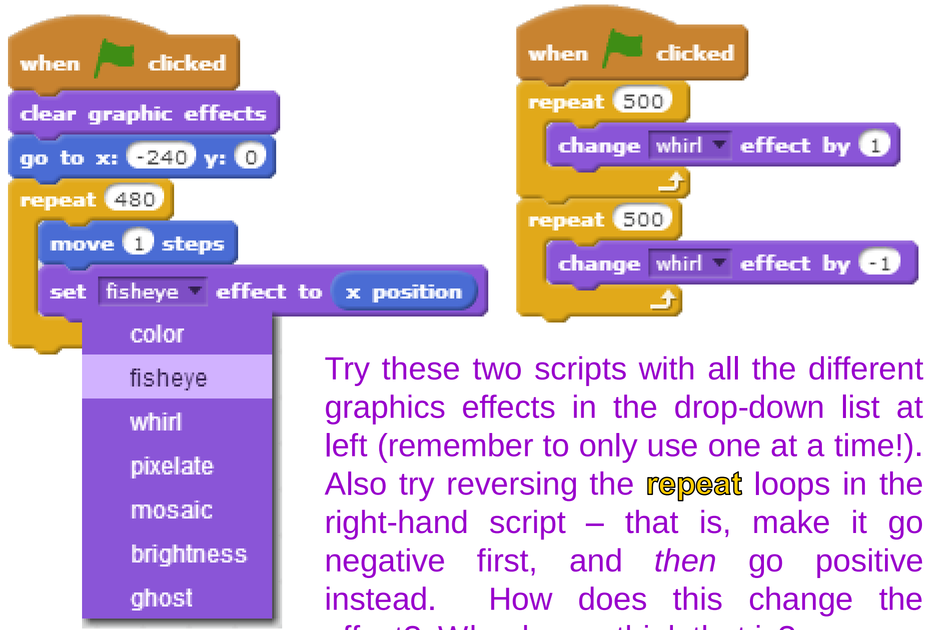
```
when green flag clicked
say [Hello!]
forever
    turn ↻ (15) degrees
```

```
when green flag clicked
say [Hello!] for (2) secs
forever
    turn ↻ (15) degrees
```



```
when green flag clicked
forever
    next costume
    wait (.25) secs
```

```
when green flag clicked
forever
    switch costume to [cat1-b]
    wait (0.25) secs
    switch costume to [cat1-a]
    wait (0.25) secs
```



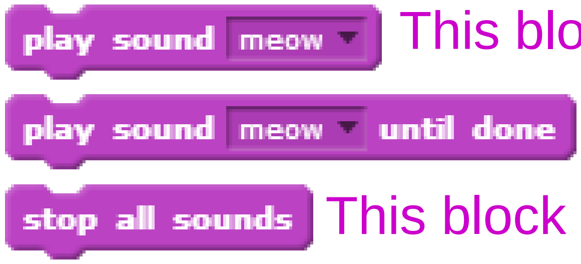
```
when green flag clicked
clear graphic effects
go to x: (-240) y: (0)
repeat (480)
    move (1) steps
    set [fisheye] effect to (x position)
```

```
when green flag clicked
repeat (500)
    change [whirl] effect by (1)
repeat (500)
    change [whirl] effect by (-1)
```

Try these two scripts with all the different graphics effects in the drop-down list at left (remember to only use one at a time!). Also try reversing the repeat loops in the right-hand script – that is, make it go negative first, and *then* go positive instead. How does this change the effect? Why do you think that is?

Sounds Blocks

Block Definitions

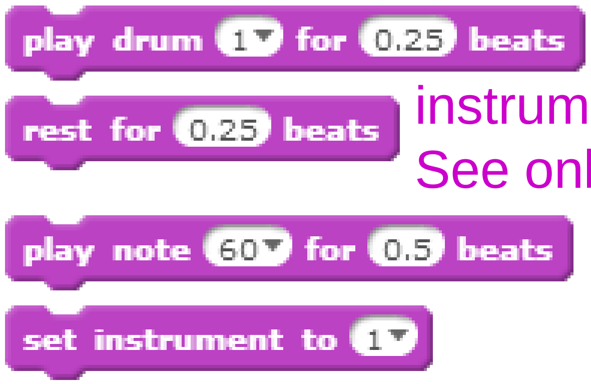


```
play sound meow
play sound meow until done
stop all sounds
```

This block starts playing a sound, then moves to the next block.

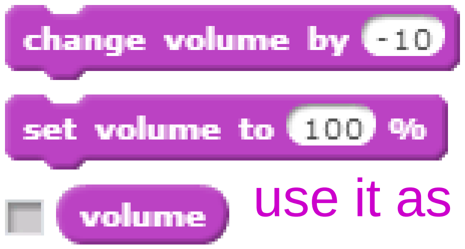
This block plays a sound completely, then moves on.

This block stops all sounds in the program.



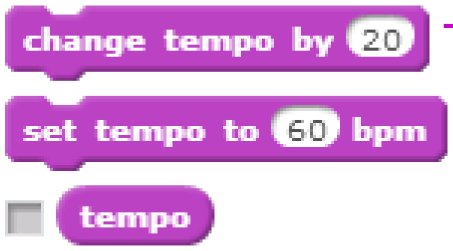
```
play drum 1 for 0.25 beats
rest for 0.25 beats
play note 60 for 0.5 beats
set instrument to 1
```

These blocks are able to make music with different instruments (including drums), and are also capable of rests. See online tutorials for in-depth information on how to make your own musical compositions in Scratch.



```
change volume by -10
set volume to 100 %
volume
```

These blocks change volume by a relative value or set it to an absolute value. The **volume** block detects volume and can use it as a variable (see **Operators** blocks).



```
change tempo by 20
set tempo to 60 bpm
tempo
```

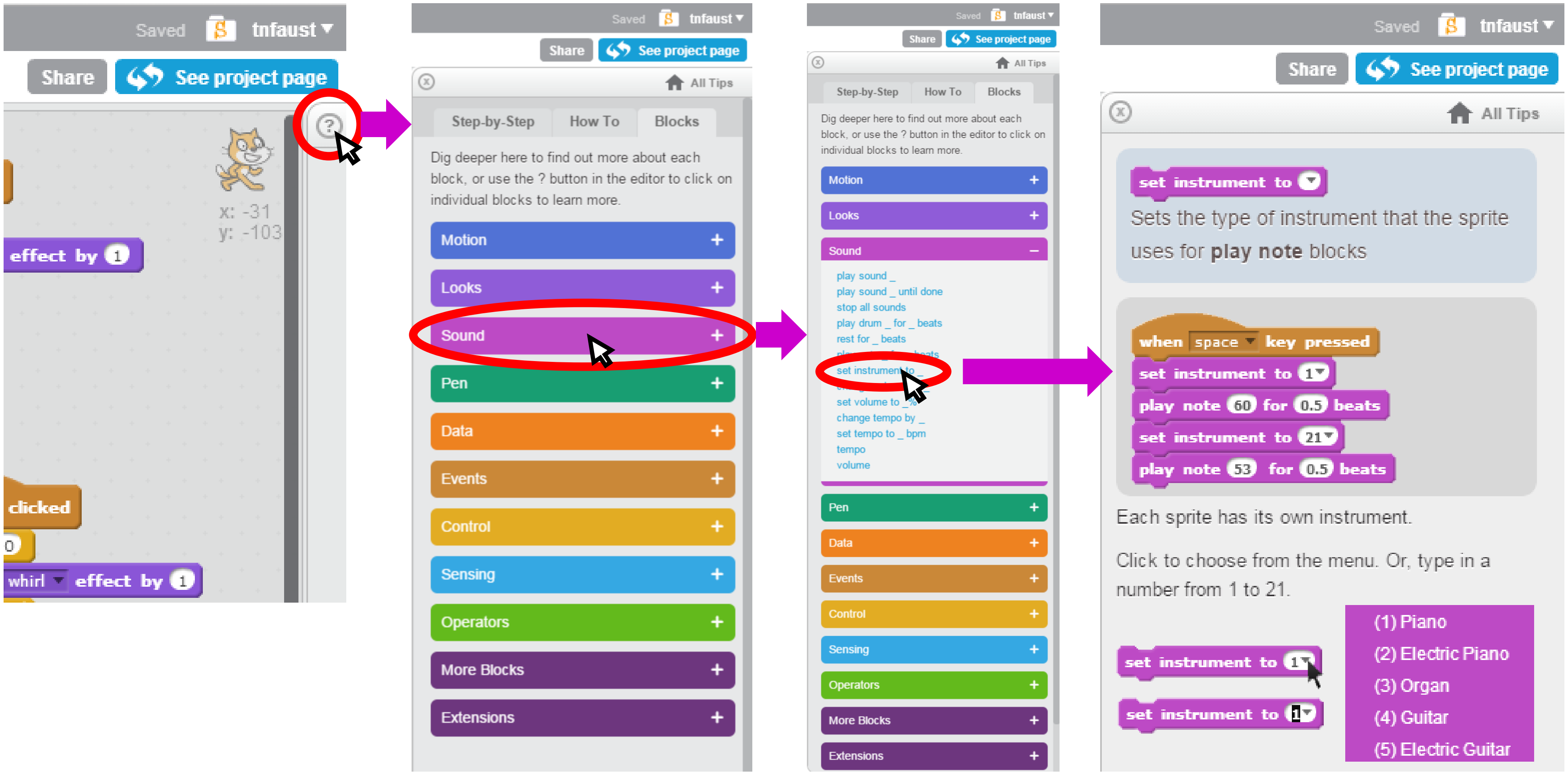
These blocks change the tempo of scripted music by a relative value, or set it to an absolute value. See online tutorials for in-depth information on how to compose music in Scratch.

Sounds Blocks

Tips and Examples

At top-right, you can find Scratch's on-board help menu. This very helpful set of tips can give you step-by-step tutorials for some basic projects, how-to guides for specific project features, and also in-depth information on every block that exists. Below is one example, showing how to see information on the **set instrument to** block: click on the question-mark to open the menu, click on **Sound** to see those blocks, click on **set instrument to** for its documentation, and now you can see which instrument number sounds like what instrument. You can click on the X at top-left of the menu to close it at any time, or click "All Tips" to go back to the menu's home.

Lots of information that cannot be covered in this general reference can be found in the help menu, and don't forget the wiki!



Dig deeper here to find out more about each block, or use the ? button in the editor to click on individual blocks to learn more.

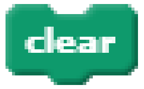
Sets the type of instrument that the sprite uses for **play note** blocks

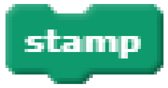
Each sprite has its own instrument.

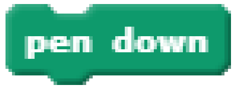
Click to choose from the menu. Or, type in a number from 1 to 21.

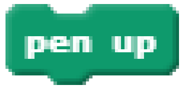
Pen Blocks

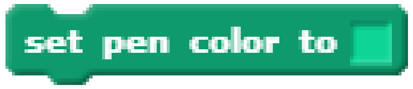
Block Definitions

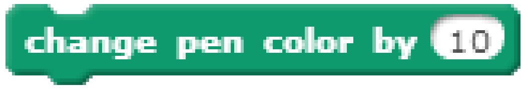

`clear` This block erases all pen drawings from the stage.

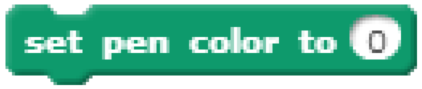

`stamp` This block makes a “stamp” of the sprite, as it looks now, on the stage.

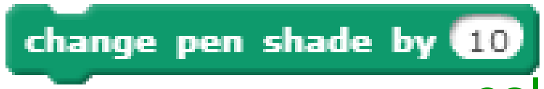

`pen down` This block places the pen down into the “draw” position.

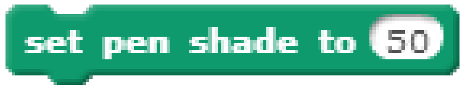

`pen up` This block lifts the pen up from the paper, so it will not draw on the stage.


`set pen color to 🟩` This block sets the color of the pen to a color you choose.

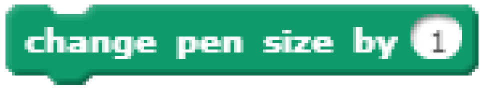

`change pen color by (10)` This block changes the pen’s color by a relative amount.

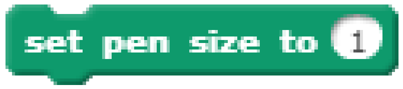

`set pen color to (0)` This block sets the pen’s color to an absolute value.


`change pen shade by (10)`


`set pen shade to (50)`

These blocks change or set the darkness of the pen’s color. Experiment with it to see how it works; the graphical effects of the pen are difficult to judge without experience.


`change pen size by (1)`


`set pen size to (1)`

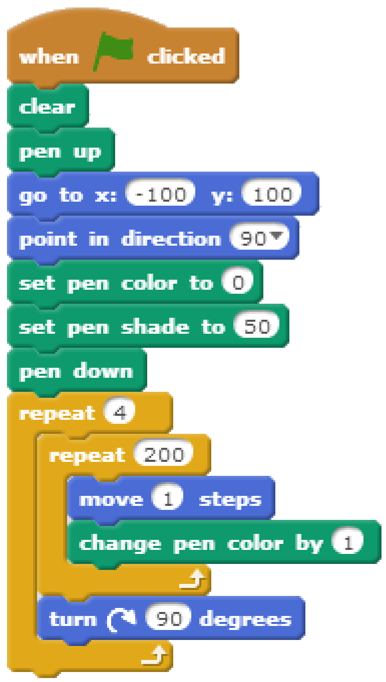
These blocks control the size of the pen, in pixels. You can change the size by a relative value, or set it to an absolute value.

Pen Blocks

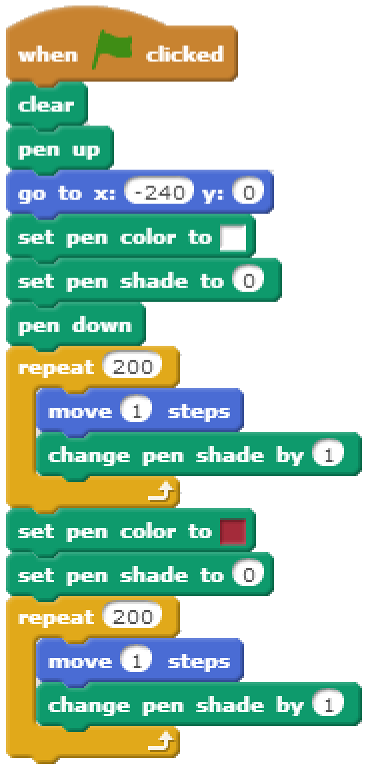
Tips and Examples

The pen can be tricky at first. Try building the below scripts to see how they work. The one at left will show you the basics of the **change pen color by** block – see if you can guess what it will look like before you run it. The one in the middle will show you how the **change pen shade by** block works – try to describe what it does after you run it, and try some experiments.

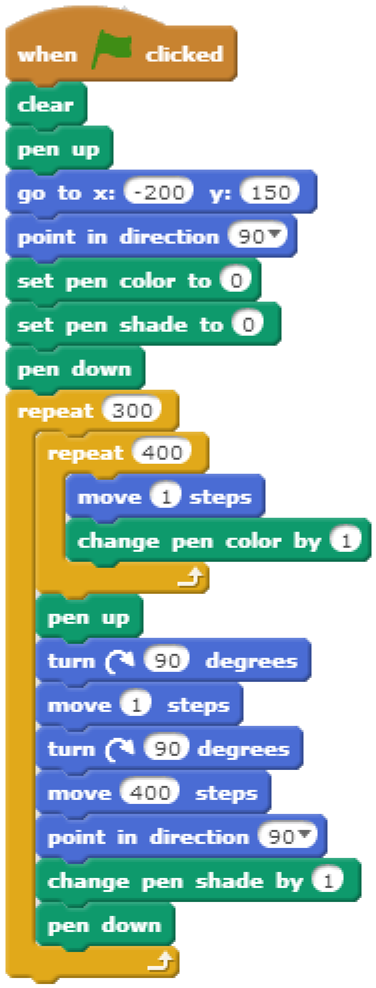
The last script is a bit of a doozy. Before you run it, hold the Shift key while clicking the green flag to activate “Turbo Mode” (it will say “Turbo Mode” by the green flag when Turbo Mode is active – you can hold Shift and click the green flag again to deactivate it). If you don’t use Turbo Mode, the far right script will take over an hour to run. (For bonus points, see if you can figure out why it looks a little different every time you run it.)



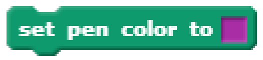
```
when ⚑ clicked
clear
pen up
go to x: -100 y: 100
point in direction 90▾
set pen color to 0
set pen shade to 50
pen down
repeat 4
    repeat 200
        move 1 steps
        change pen color by 1
    turn ↻ 90 degrees
```



```
when ⚑ clicked
clear
pen up
go to x: -240 y: 0
set pen color to ⬜
set pen shade to 0
pen down
repeat 200
    move 1 steps
    change pen shade by 1
set pen color to 🟥
set pen shade to 0
repeat 200
    move 1 steps
    change pen shade by 1
```



```
when ⚑ clicked
clear
pen up
go to x: -200 y: 150
point in direction 90▾
set pen color to 0
set pen shade to 0
pen down
repeat 300
    repeat 400
        move 1 steps
        change pen color by 1
    pen up
    turn ↻ 90 degrees
    move 1 steps
    turn ↻ 90 degrees
    move 400 steps
    point in direction 90▾
    change pen shade by 1
    pen down
```



```
set pen color to 🟪
```

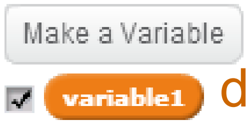
Set Pen Color to [color]:

This block in particular can be tricky at first, because you don’t enter data into it like most other blocks (such as **move** or **set pen shade to** blocks). For this block, first click in the colored square, then click anywhere on the screen to set the color to that pixel. This means that if you want a specific color, it has to be on the screen somewhere in order to click it.

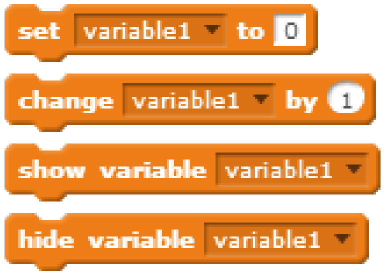
There are a few ways to do this. If you are having trouble getting the *exact* color you want because there are lots of other colors around it, try thinking of ways to make it easier (for instance, by making your target bigger to pick the color, and then smaller again when you’re done).

Data Blocks

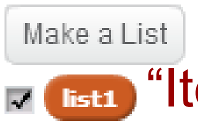
Block Definitions



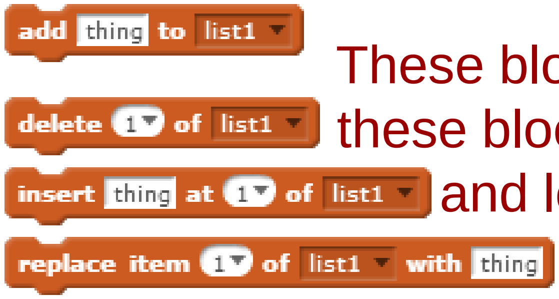
Use this button to create a variable. You should name it something descriptive, after what you're using it for (e.g. "score," "lives," etc.).



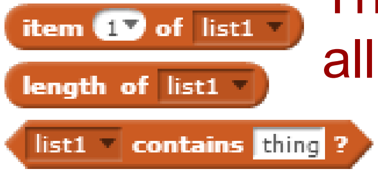
These blocks set a variable to an absolute value, change it by a a relative value, and control whether it is shown or hidden. Variables like a score or number of lives should be shown, but other variables can be hidden (for "achievements" or other).



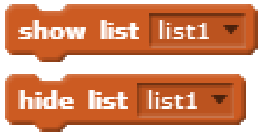
Use this button to create a list. You should name it descriptively, as in "Items" or "Names," depending on what you're using it for.


add thing to list1
delete 1 of list1
insert thing at 1 of list1
replace item 1 of list1 with thing

These blocks add, remove, and change list items. You can use these blocks for a wide variety of functions – experiment with them, and look up tutorials on the internet for advanced functions.



These blocks can be inserted into other blocks (see **Operators**) as all kinds of data. See the "Number Guesser" project (at https://scratch.mit.edu/projects/22179026/#editor) for ideas. Get creative with it, and experiment to learn the finer points.

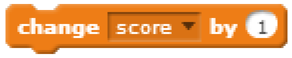


You will rarely want to show/hide lists, but exceptions are exceptional.

Data Blocks

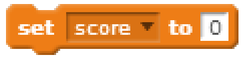
Tips and Examples

Open up a starter project on Scratch, such as the Pong Starter (at https://scratch.mit.edu/projects/10128515/#editor), that could use a "score" and "high score" feature. In the **Data** blocks, click "Make a Variable" to create a variable called "score" and another one called "high score." Both variables will be shown by default when you create them, and there's no reason to change this, since we'll always want to see them.



Now that you've created your variables, you'll see more data blocks become available. Use the **change [variable] by** block to increase the score by 1 each time the ball touches the paddle. Look through the scripts (there aren't many) to figure out where this should go (you don't need a new script, you can put this block into an existing script).

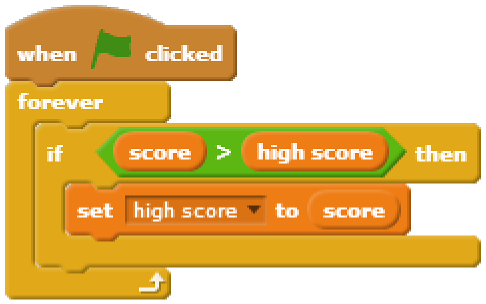
Play the game a couple of times (if you're good, just die intentionally after racking up a few points), and see if you can identify the problem with how the score works. You can fix this problem by putting the **set [variable] to** block into a different script – try to figure out where you should put it.



Now the score should set to zero when the game starts (making sure things start out right is called "initializing," and is an important programming concept). However, our high score still doesn't do anything. We can write one simple script to fix this, and we can even say how it should work in plain English:

When the green flag is clicked, forever: if score is greater than high score, then set high score to score.

With the blocks all clicked together, the script should look like this:

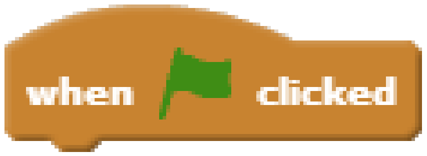


Now the score should set to zero every time the game starts, but the high score should only change when you score more than the old high score, and the high score should never totally reset to zero.

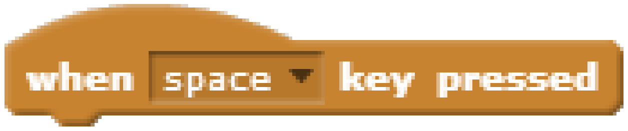
What if you wanted to show, say, the top three or five high scores? Try using a list to accomplish this. Do you want the list showing all the time, or only when the game ends? A short list, like only three, might be reasonable to show all the time – but not ten! You wouldn't be able to see what you were doing in the game!

Events Blocks

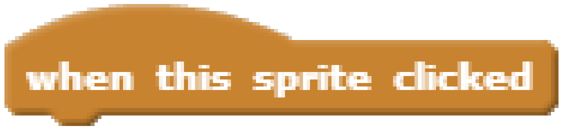
Block Definitions



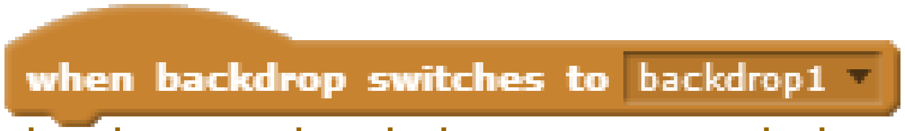
This block starts a script of blocks, and will start whenever the the green flag is clicked. You will probably use this block a lot.



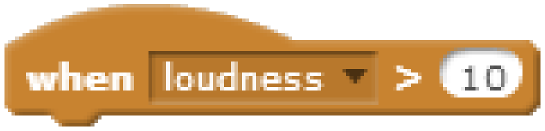
This block starts a script of blocks when a key is pressed. You can use A-Z, 0-9, space, and arrow keys.



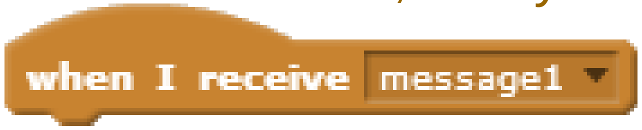
This block starts a script when the sprite is clicked.

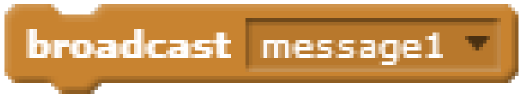


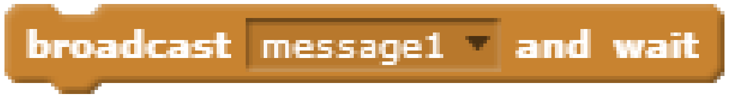
This block starts a script when the stage's background switches to a certain backdrop (you can define it by name).



This block starts a script of blocks when the microphone receives a certain loudness. This requires a microphone in order to work; it only detects volume; look up tutorials for more information.







These blocks use messages to communicate "behind the scenes" information between sprites. Look up tutorials for more information, or see https://scratch.mit.edu/projects/33044982/#editor for an example.

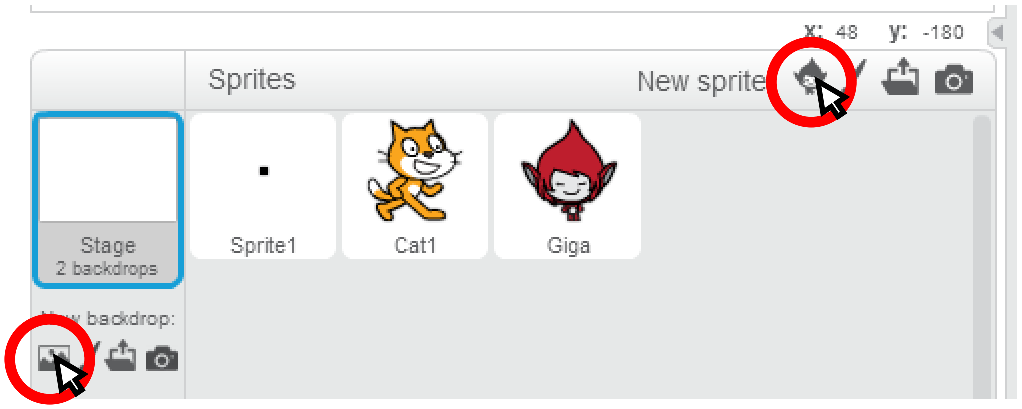
Events Blocks

Tips and Examples

Events are some of the most useful blocks in Scratch, as they are the only blocks that can start scripts (aside from **when I start as a clone**). Every script that you want to run has to start with an **Events** block (again, except for clones). **Events** are how you run scripts off the green flag, clicking sprites, pressing keys, or broadcast messages.

Broadcast Messages:

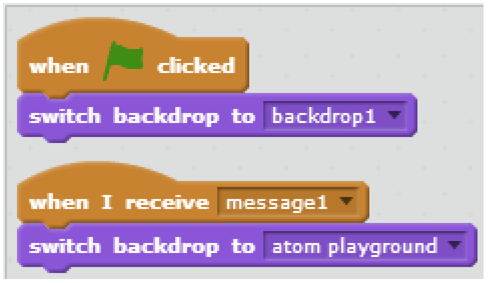
Broadcast messages are "behind the scenes" cues that allow sprites to communicate directly with each other and the Stage. These communications are invisible and silent – only the computer knows they're happening, and you can only tell when they happen by looking at the code of a program. Here is a simple example of broadcast messages in action:



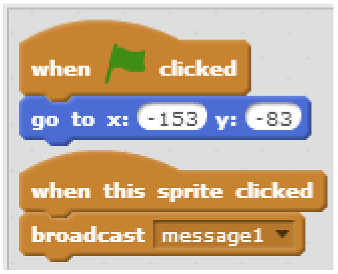
Create a new project, and add a backdrop (such as the Atom Playground) by clicking the "Choose backdrop from library" button (shown at near right). Also, add a sprite (such as Giga) by clicking the "Choose sprite from library" button (shown at far right).

Below are shown three sets of two scripts each. Give the left pair of scripts to the Stage (notice that the Stage *cannot* use **motion** blocks – things move on the Stage, but the Stage itself never moves).

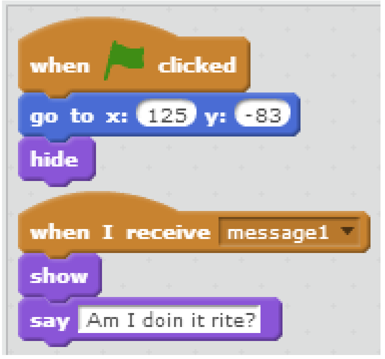
Give the middle pair of scripts to the cat (who you should have started with), and give the right pair of scripts to Giga (or whoever is your second sprite). Now click the green flag, and then click the cat to see what happens! Notice that several things can run off of the same broadcast message, just like the green flag. What other things could you use broadcast messages for?



```
when 🏴 clicked
switch backdrop to backdrop1

when I receive message1
switch backdrop to atom playground
```



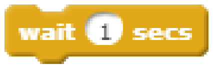
```
when 🏴 clicked
go to x: -153 y: -83

when this sprite clicked
broadcast message1
```



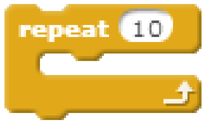
```
when 🏴 clicked
go to x: 125 y: -83
hide

when I receive message1
show
say Am I doin it rite?
```

Control Blocks

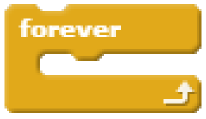
Block Definitions



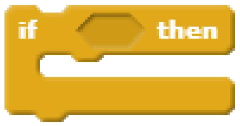
This block tells a sprite to wait for some time (1 sec, .25 sec, etc.).



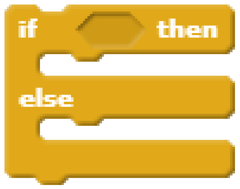
This block tells a sprite to execute the code inside a number of times.



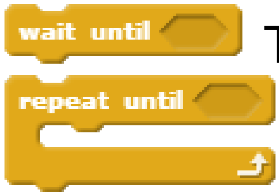
This block tells a sprite to execute the code inside forever (all the time).



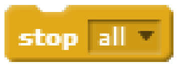
This block checks a logical test, and if the test succeeds, then it will execute the code inside. If not, the code inside is skipped.



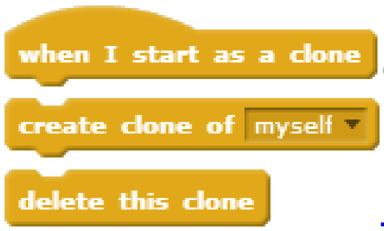
This block checks a logical test, and if the test succeeds, then it will execute the code inside the first block. If not, then it will execute the code inside the second block (the "else" part).



The **wait until** block waits until a logical test succeeds, then executes the code inside. The **repeat until** block will repeatedly execute the code inside until a logical test succeeds; then it will stop.



This block stops all scripts in your program. Think of it as "Quit" or "Exit."



Clones are copies of a sprite with separate scripts of their own. Look up online tutorials to see how they work, or see the wiki: http://wiki.scratch.mit.edu/wiki/Cloning for more information.

Control Blocks

Tips and Examples

Language and Ambiguity

Logic control can be confusing to new Scratchers, and for good reason: human language uses a lot of *ambiguity*, meaning that the same set of words can be used to mean different things. For example, how do you pronounce the word "read"? It depends on the rest of the sentence ("I'll read it tomorrow" vs. "I already read that"). Even the word "is" can have different meanings: two plus two *is* four in a different way than a T-shirt *is* black. What's more, entire sentences can change their meanings, based on your tone and the rest of the conversation. How many different ways could you answer the question, "Could you put that in different words?" You could answer it in a lot of different ways, *even if you're only re-phrasing one statement*. Or consider the following snippet of conversation:

Person A: Now do you get it?

Person B: Oh, yes. I get it now.

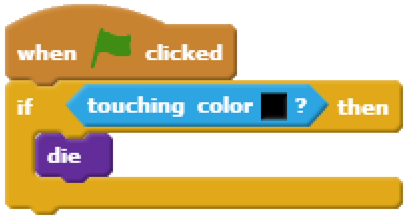
How differently would each person say their line if they were talking about a joke (happy)? Or a betrayal (angry or sad)? Or a mystery movie (confused)? Or a really hard math problem (frustrated)?

Four Parts to an Instruction

Computers *cannot* tolerate ambiguity. All instructions to a computer must have one *exact* meaning, and *only* one meaning. Each instruction to a computer can be thought of as having the following four parts that together make up the whole instruction:

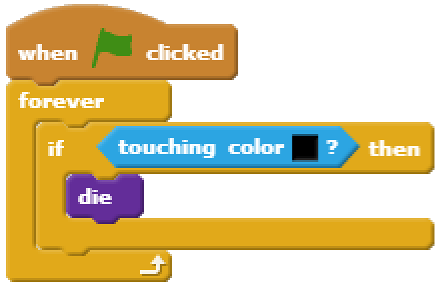
1. *When do you want it to start?*
2. *What do you want it to do?*
3. *How much do you want it to do it?*
4. *When do you want it to stop?*

The simpler these parts are, the shorter the script will be. But be careful! If you're making a jumping game with black pits of death, you might imagine saying to a person, "OK, **when the green flag is clicked**, **if** the player is **touching** blackness, **then die**." If you script that in Scratch, it might look like this:

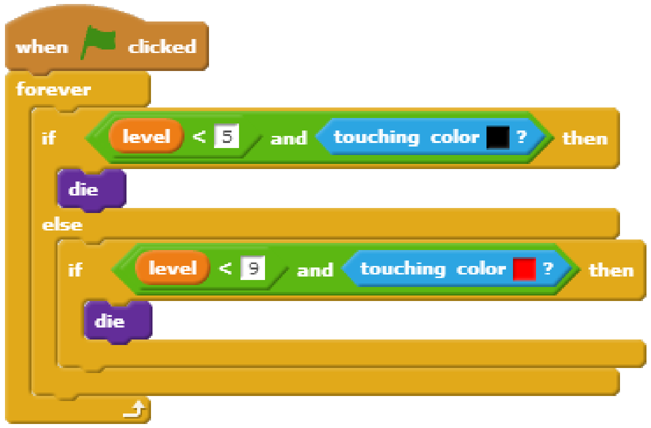


A human knows to do that whenever you touch the black pit, because a human knows you're making a jumping game.

But a computer will only make the check *once*, because you didn't tell it to do it more than once. Really, you want it making the check *all the time*, which means you need a **forever** block:

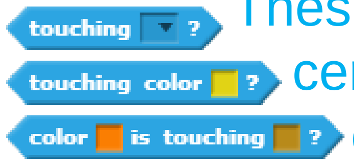


But you might want it to make an exception – maybe you have underground levels with a black background, so you want to use lava instead of blackness for the pits. If the first four levels are above ground, and the next four are underground, your script might look like this:

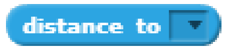

when clicked
forever
if level < 5 and touching color ? then
die
else
if level < 9 and touching color ? then
die

Sensing Blocks

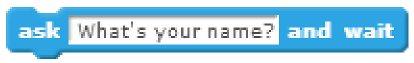
Block Definitions

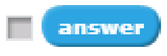

touching ?
touching color ?
color is touching ?

These blocks check if a sprite or clone is touching another sprite or a certain color, or if one color is touching another color. Insert them into other blocks (usually **Control** blocks) to serve as logical tests.

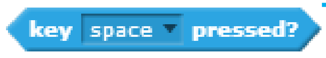

distance to

This block measures the distance between sprites (in pixels).

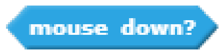

ask What's your name? and wait


answer

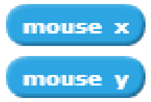
This block waits for user input. You can store the answer with the **answer** block as a variable or string of text for later use.


key space pressed?

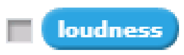
This block checks if a key (A-Z, 0-9, space, or arrows) is pressed.


mouse down?

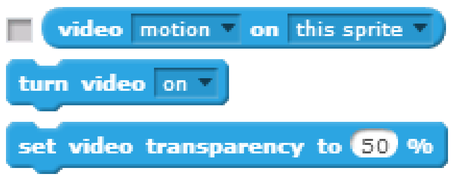
This block checks if the left mouse button is clicked.


mouse x
mouse y

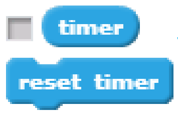
These blocks measure the x and y coordinates of the mouse pointer.


loudness

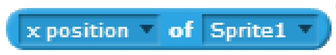
This block measures the absolute volume from the microphone input.


video motion on this sprite
turn video on
set video transparency to 50 %

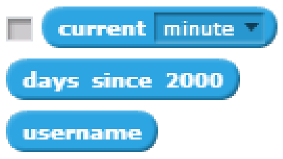
Scratch is able to use the webcam (if you have one), but it is complicated. Look up an online tutorial or see the wiki for more information.


timer
reset timer

Scratch has an in-game timer, but it is often quirky. In most cases, you are better off making your own timer using **Data** blocks and the **wait** block.


x position of Sprite1

This block can measure various attributes of a sprite.

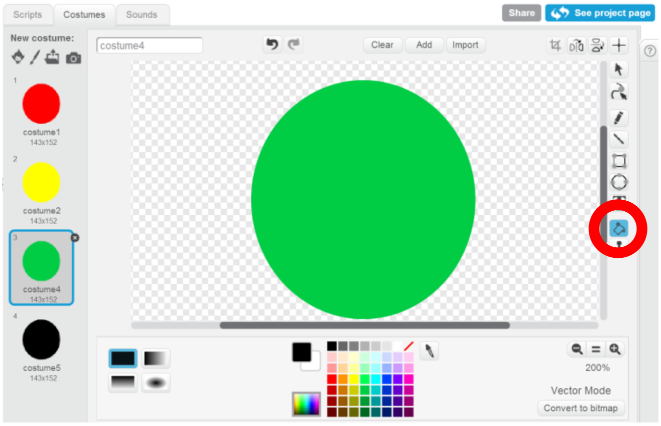

current minute
days since 2000
username

These blocks are useful for cloud variables. See the wiki for more info.

Sensing Blocks
Tips and Examples

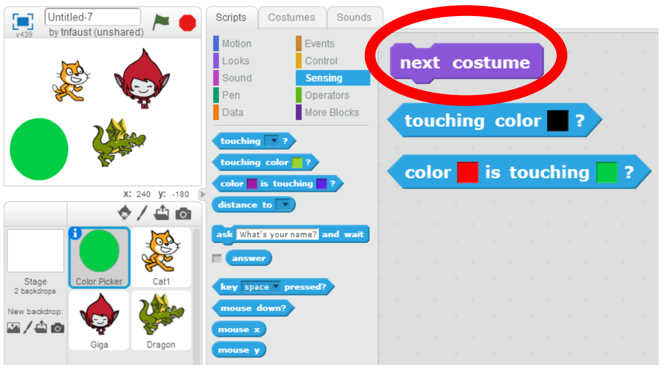
Dithering, or: "Why won't this color work?"

The **Sensing** blocks can be quite finicky, as their colors work with exact RGB values and Scratch uses some tricks that mess with those exact RGB values to make things look nicer. One of these tricks is called "dithering," which means "making a small area of color *not exactly that color* so that the bigger picture looks better. You can use your own tricks to help you out, if you're running into problems like this.



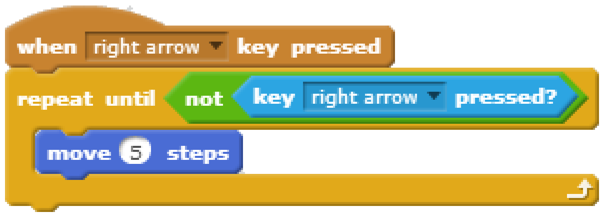
Working Smarter, not Harder

If you're trying to use the **touching color** block to tell when a sprite is touching a color, but you're having trouble picking the color, you can make a sprite just to help you with this. Maybe call it "color picker," so you know what it's for, and you can click a **hide** block whenever you want it to go away. Just make a big shape, and copy the costume a bunch, then use the paint bucket tool to change what color the different costumes are (as shown at upper right). Then you can click the **next costume** block to cycle between colors easily, pick them from a nice big spot, and then drag your **Sensing** blocks to the sprites you want to actually use them. It's a few extra steps, but each step is simple and easy, which is way better than trying to click on *that one exact pixel* (especially if Scratch is dithering it). And the more times you want to pick a color, the more your extra color picker sprite pays off!



Better Motion

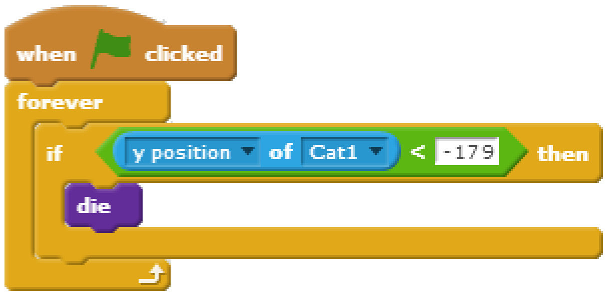
Try out this sample script in the Maze Starter project (https://scratch.mit.edu/projects/10128431/#editor). Make sure to deactivate the old script! Which is better?



```
when [right arrow] key pressed
repeat until <not <key [right arrow] pressed?>>
    move (5) steps
```

Better Death

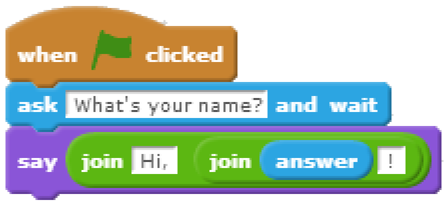
How might you use this script instead of the **touching color** block (see **Control**)?



```
when flag clicked
forever
    if <([y position] of [Cat1]) < (-179)> then
        die
```

Joining Forces

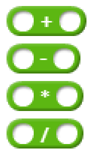
When using the **ask** and **answer** blocks, you will often want to use **Operators** as well. Build this sample script to try it out! You can also use numbers to do math!



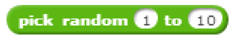
```
when flag clicked
ask [What's your name?] and wait
say (join [Hi, ] (join (answer) [!]))
```

Operators Blocks

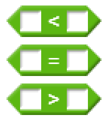
Block Definitions



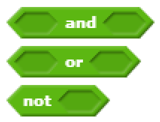
These blocks will perform arithmetical operations on numbers and/or variables. You can put these blocks inside other blocks, and inside each other.



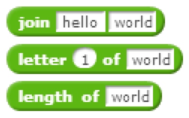
This block picks a random number from a user-defined range.



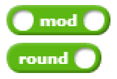
These blocks test for equivalence or relative value (equals, greater than, or less than). They can be used with **Control** blocks to make logic tests.



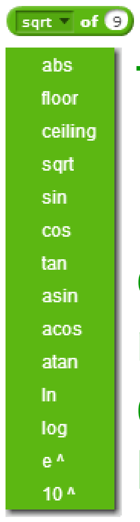
These blocks are used for logical tests, mostly with **Control** blocks.



These blocks are used to do things to “strings” of text, but can also be used with numbers. Check the wiki for more information on this topic.



The “modulo” function reports the remainder of a division operation.



The “round” function removes the remainder from a division operation.
The “sqrt” function reports the square root of a number.

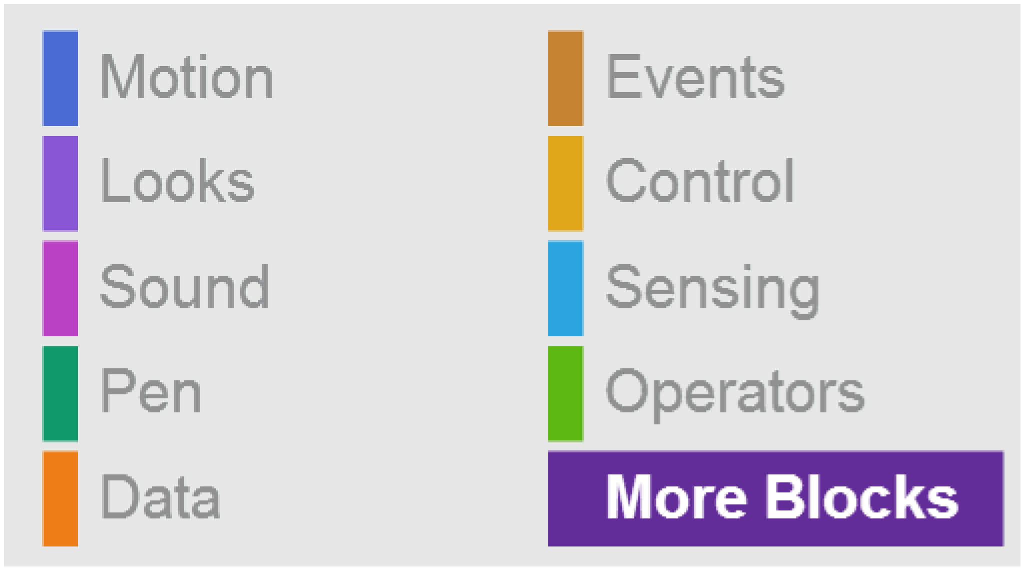
For the other operations (shown at left), you may wish to look up them up on the wiki or in a high school algebra/trigonometry textbook. They have many uses (for example, distance as reported in the **Sensing** blocks, or direction as shown in the **Motion** blocks), but are complicated if you don’t know what they are. Brief definitions are given below.

Operators Blocks

Tips and Examples

Index of Blocks

[for making tabs]


Motion
Looks
Sound
Pen
Data
Events
Control
Sensing
Operators
More Blocks