Broadcast

A **broadcast** is a message that is sent through the Scratch program, activating scripts with the matching hat blocks. Broadcasts are sent with the blocks Broadcast () and Broadcast () and Wait, and are received by the hat block When I Receive ().

Broadcasts allow scripts to continue into other sprites, as some values are different to different sprites. They can also be used to branch a single sending script into many receiving scripts, or to close many sending scripts into a single receiving script.

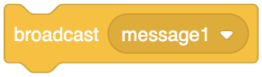
Broadcasts are useful in games and animations, as they trigger specific scripts. They are similar to events, which are scripts triggered when certain actions, like mouse moves or key presses, are performed.

Examples

All the broadcast blocks can be found in the Events Block section. In Scratch 1.4 and earlier, they were in the Control Block section.

Broadcast ()

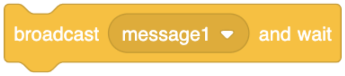
Main article: Broadcast () (block)

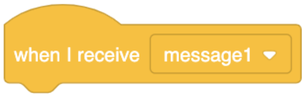


This block broadcasts the specified message and has no further effect.

Broadcast () and Wait

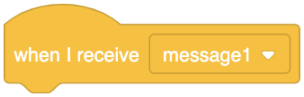
Main article: Broadcast () and Wait (block)



This block broadcasts the specified message and blocks its script until all scripts under a  block have finished.

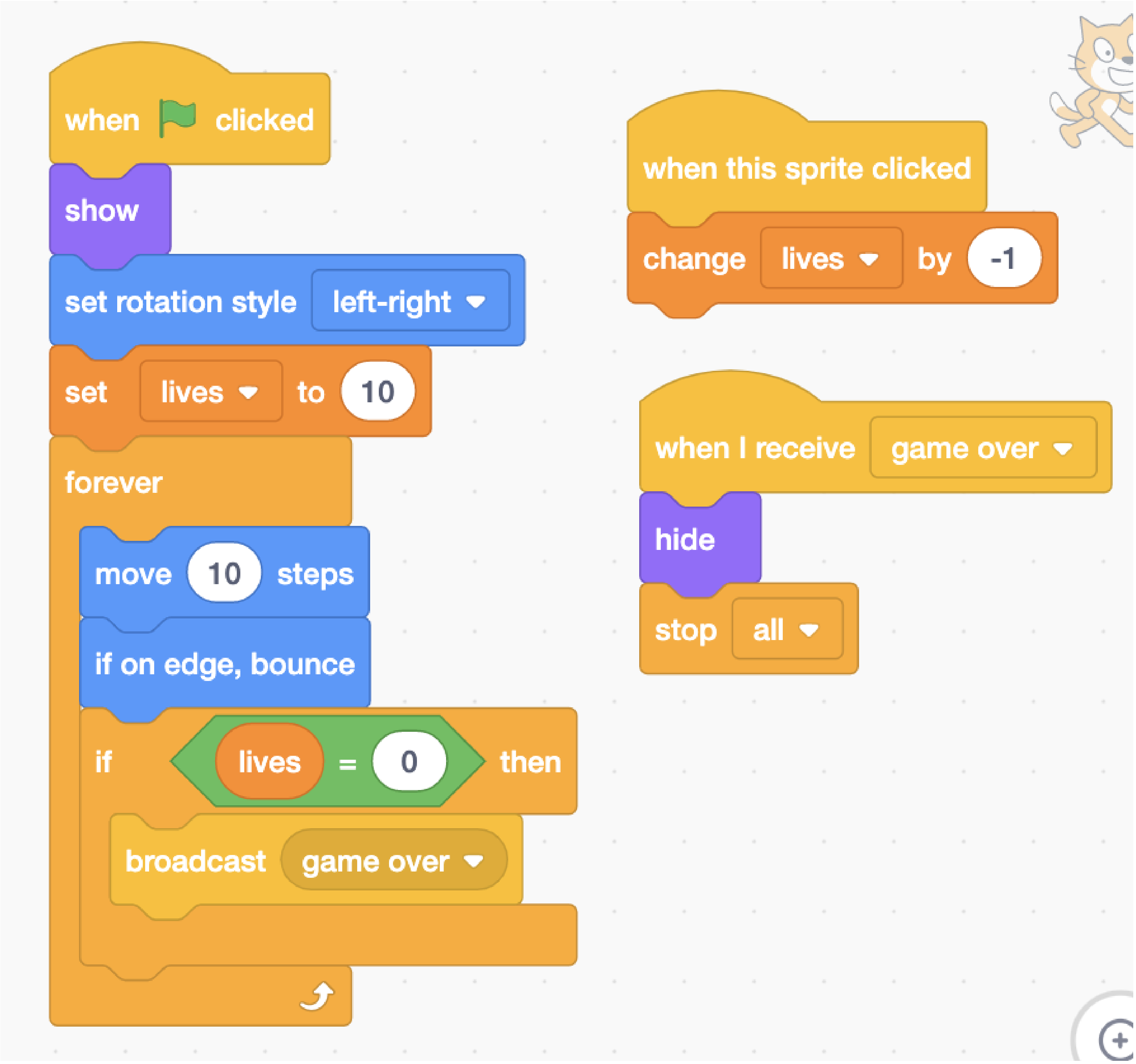
When I Receive ()

Main article: When I Receive () (block)



This block will stay inactive until it receives the specified broadcast. Once it has been received, the script goes into action and ends once it has finished, but it can be started more than once.

Try it!

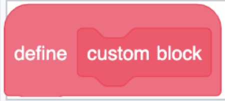


```
when ⚑ clicked
show
set rotation style [left-right ▾]
set [lives ▾] to (10)
forever
    move (10) steps
    if on edge, bounce
    if <(lives) = (0)> then
        broadcast [game over ▾]
```

```
when this sprite clicked
change [lives ▾] by (-1)
```

```
when I receive [game over ▾]
hide
stop [all ▾]
```

My Blocks

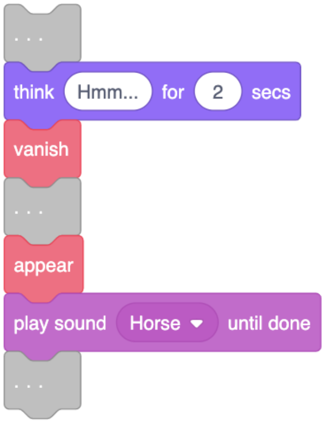


The Current Define block

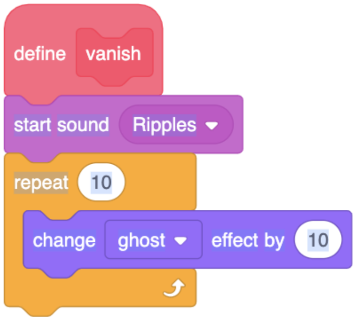
My Blocks (known as **More Blocks** in Scratch 2.0) is one of the ten categories of Scratch blocks. It holds **procedures** for the selected sprite. They are color-coded pink. Before any blocks are created, it is empty, except for a "Make a Block" button.

Clicking Make a Block brings up a dialogue allowing the user to make a procedure. Once OK is pressed, the new block appears in the palette and an empty *definition* appears in the code area. When the procedure runs, Scratch will run the blocks below the corresponding Define block.

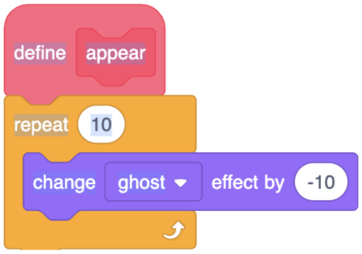
Apart from variables, procedures are the main kind of abstraction used in procedural programming; they let Scratchers write scripts without knowing or thinking about the details of what each script does. This is also true of Scratch's built-in blocks, like . For example, suppose one wants a sprite to vanish and reappear. To vanish, the sprite should play a sound and repeatedly increase the ghost effect; to reappear, it should repeatedly decrease the ghost effect. This should happen at several points in the project. Without procedures, the Scratcher would have to duplicate the same sequence of blocks at each point, but by making "vanish" and "appear" blocks, they can write scripts in a way which matches how they think of them:



```
. . .
think (Hmm...) for (2) secs
vanish
. . .
appear
play sound (Horse ▾) until done
. . .
```



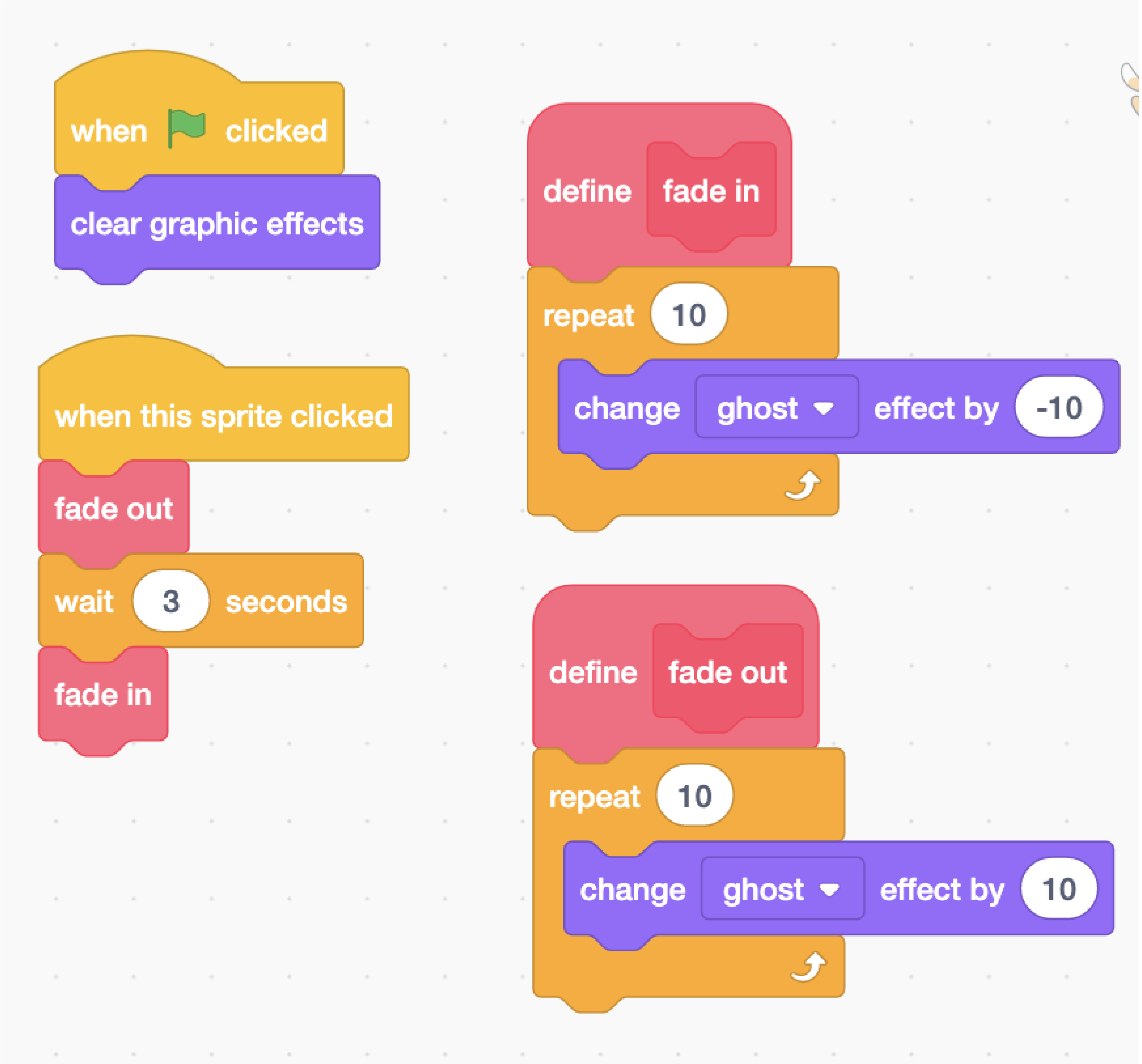
```
define vanish
start sound (Ripples ▾)
repeat (10)
    change [ghost ▾] effect by (10)
```



```
define appear
repeat (10)
    change [ghost ▾] effect by (-10)
```

Try it!

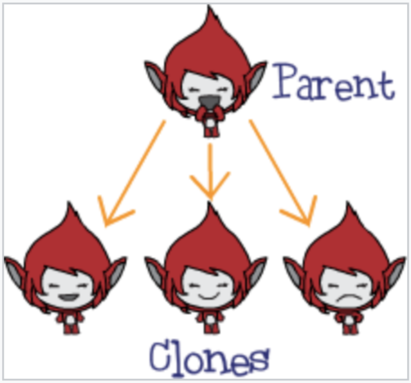
Create the example blow! When a sprite is clicked, it vanishes for 3 seconds then appears.



```
when ⚑ clicked
clear graphic effects

when this sprite clicked
fade out
wait (3) seconds
fade in

define fade in
repeat (10)
    change [ghost ▾] effect by (-10)
⤴

define fade out
repeat (10)
    change [ghost ▾] effect by (10)
⤴
```

See how the blocks we defined are separate? This makes our program easier to read and fix!

Cloning



A diagram showing how cloning works. The clones of a parent sprite can be different, as the facial expressions show.

Cloning is a feature that allows a sprite to create a copy of itself while the project is running. Each clone has the same costumes, sounds, scripts, and variables as the original but is otherwise independent.

Cloning is different from stamping in that stamping only produces an image of the sprite on the stage, whereas clones are individual sprites that are able to interpret scripts and run seperately. Stamping will also by default stamp a bitmap image of the sprite. It is also different from the "duplicate" option in the sprite menu in that ordinary duplicates are permanent and appear in the sprite pane, whereas clones do not appear in the sprite pane, disappear when the stop sign is pressed, and can be sensed by other sprites using the Touching ()? block with the original sprite as its input.

There may only be 300 clones at once to prevent excessive lagging or crashes,[1] but 301 could be made in Scratch 2.0.[2]

Usage

Cloning is commonly used when a project has many similar sprites doing similar things. Because clones are created by the project rather than the user, cloning prevents the user from needing to make the same changes to each of many sprites. Some common uses of cloning are:

- Tower Defense games
- Many arcade-style games
- Particles such as fireworks and snow
- RNG-based projects[*clarify*]
- Mouse trails
- Any project needing many repetitive sprites

Blocks

There are three blocks related to cloning, all of which are found in the control palette:

- create clone of (myself ▾) — clones the sprite selected
- when I start as a clone — when a sprite is cloned, this hat block runs in the newly made clone
- delete this clone — deletes the clone it runs in and stops all of its scripts

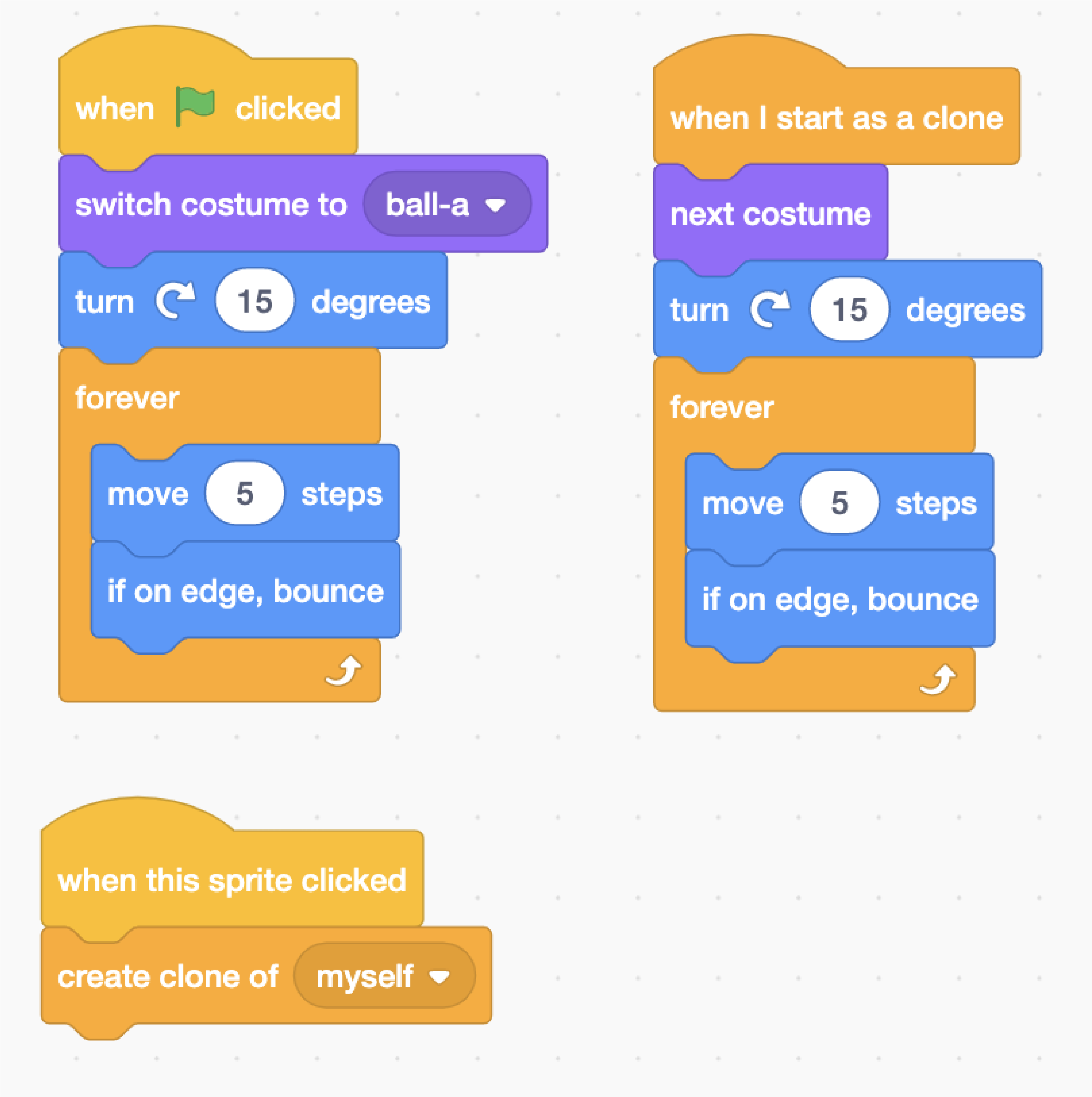
Clones can also create clones, which is often used for recursion.

Variable Usage

If a variable is marked as "for this sprite only", each clone of the sprite will have its own value for the variable that is separate from the original sprite and all other clones. This can be used to store information for individual clones, such as health for enemies in some Game Projects.

Try it!

-Create a ball sprite, add these blocks!

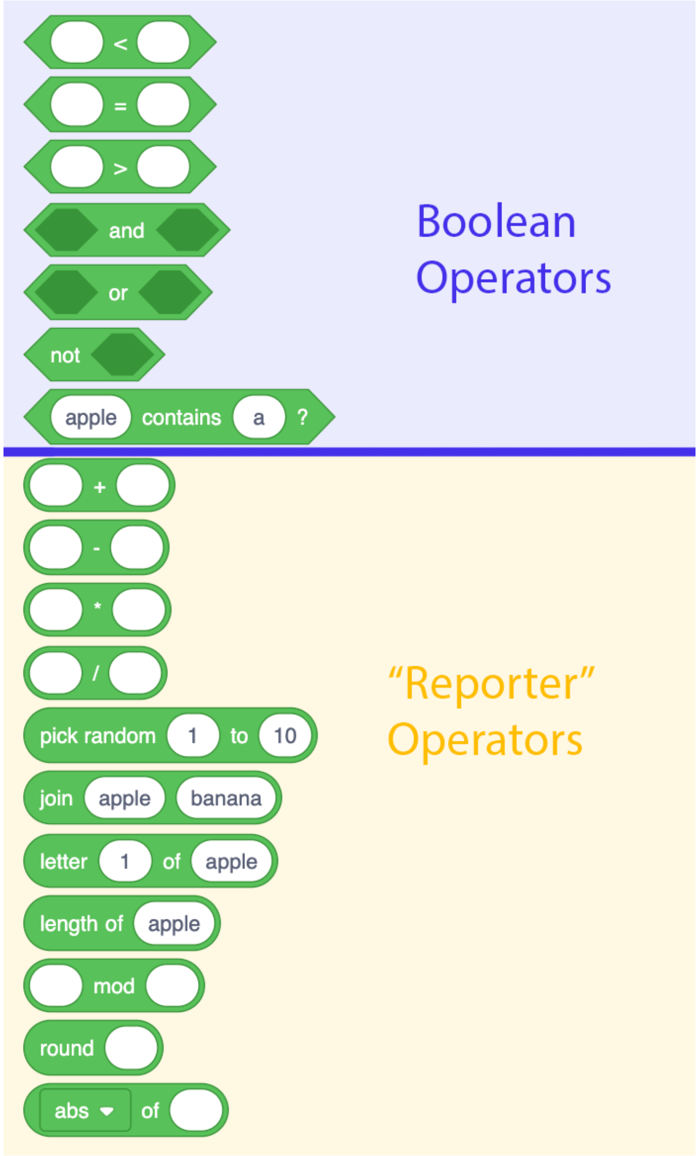


```
when ⚑ clicked
switch costume to (ball-a ▾)
turn ↻ (15) degrees
forever
    move (5) steps
    if on edge, bounce
end
```

```
when I start as a clone
next costume
turn ↻ (15) degrees
forever
    move (5) steps
    if on edge, bounce
end
```

```
when this sprite clicked
create clone of (myself ▾)
```

Operators



Boolean Operators will return a true/valse value. Duh!

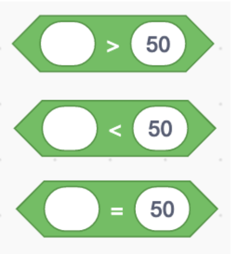
We can use these in conditional blocks like if/else and repeat until.

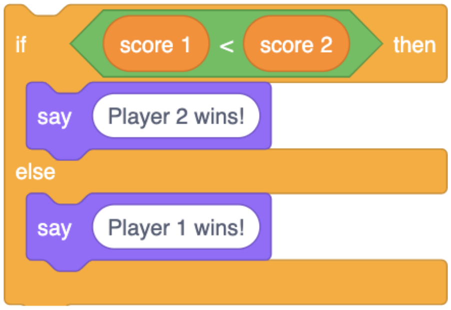
A reporter block is a block that reports a value. These can be anything, from numbers to strings.

We can use these to do some math or change text.

Comparison Operators

Like in math, the comparison operators can be used to compare two numbers. They will return a boolean (true or false) value.



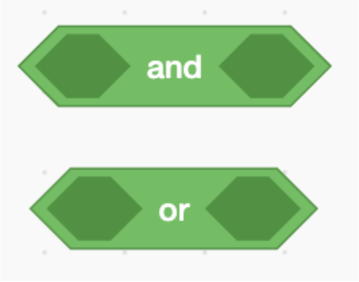


```
if score 1 < score 2 then
    say Player 2 wins!
else
    say Player 1 wins!
```

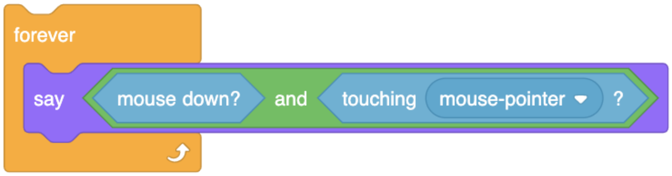
AND and OR operators

AND will be true if both boolean values are true.

OR will be true if either boolean value is true.



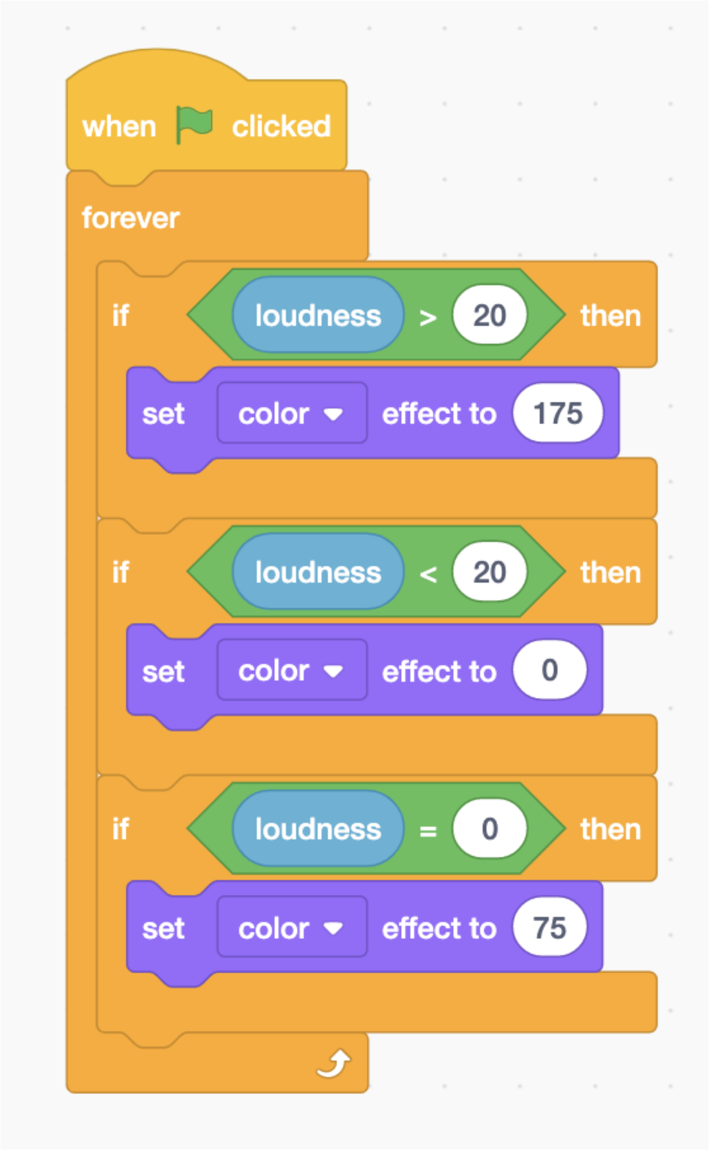
```
< and >
< or >
```



```
forever
    say <mouse down? and touching mouse-pointer ?>
```

Try it!

Create a loudness meter…



```
when 🏁 clicked
forever
    if loudness > 20 then
        set color effect to 175
    if loudness < 20 then
        set color effect to 0
    if loudness = 0 then
        set color effect to 75
```