Looks and sound- session 3

https://scratch.mit.edu/projects/editor/?tip_bar=home

In this module, you'll:

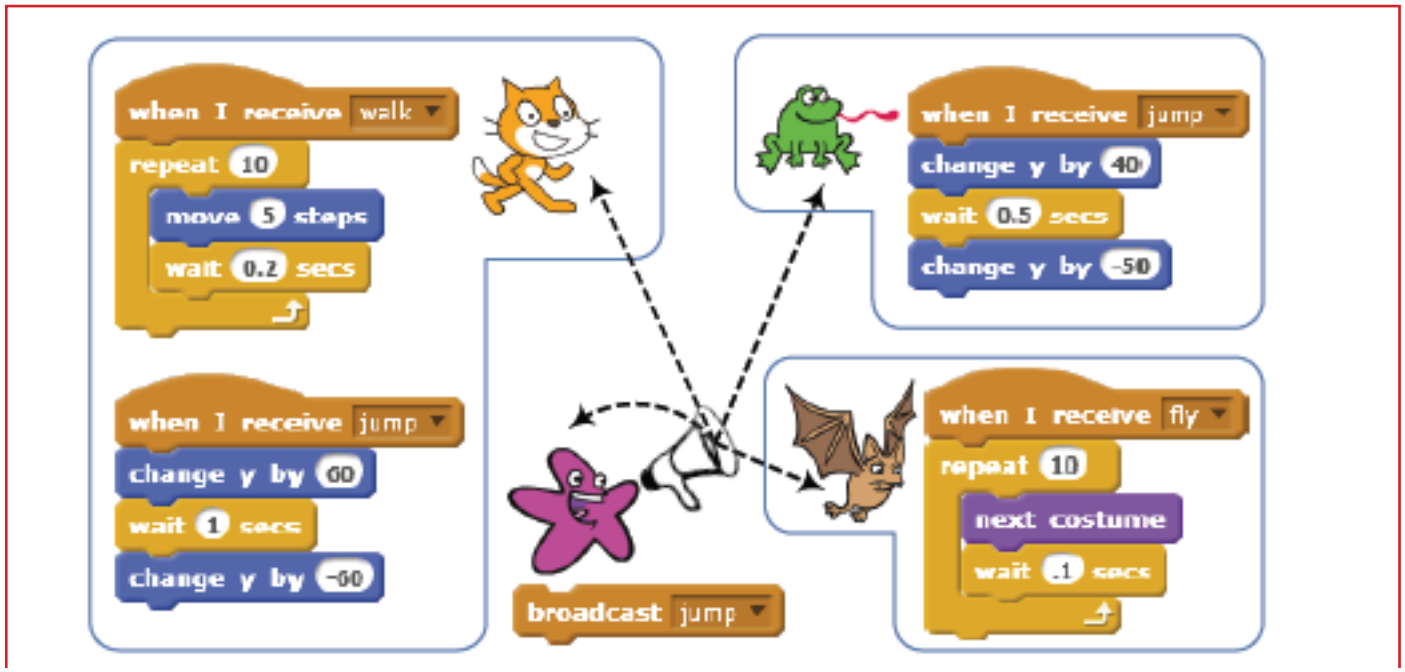
- learn how to use procedures
- Use message broadcasting to coordinate the behavior of many sprites
- Use message broadcasting to implement procedures
- Use the "build your own block" feature of Scratch 2
- Use structured programming techniques

Message Broadcasting

Broadcast

So how does the broadcast system in Scratch work in practice? Any sprite can broadcast a message (you can call this message anything you like) using the **broadcast** or **broadcast and wait** blocks (from the Events palette)

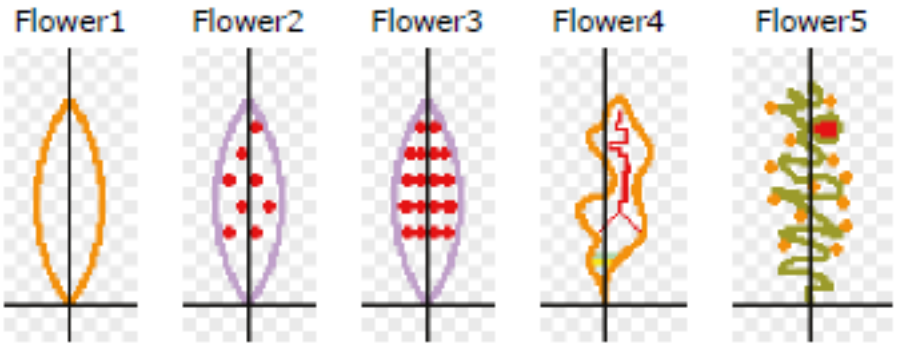
This broadcast triggers all scripts in all sprites (including the broadcasting sprite itself) that begin with a matching **when I receive** trigger block. All sprites hear the broadcast, but they'll only act on it if they have a corresponding **when I receive** block.



The picture above shows four sprites: starfish, cat, frog, and bat. The starfish broadcasts the jump message, and that broadcast is sent to all sprites, including itself. In response to this message, both the cat and the frog will execute their jump scripts. Notice how each sprite jumps in its own way, executing a different script. The bat also receives the jump message, but it does not act on it because it was not told what to do when it receives this message. The cat in this figure knows how to walk and jump, the frog can only jump, and the bat was taught only to fly. The broadcast and wait command works like the broadcast command, but it waits until all message recipients have finished executing their corresponding when I receive blocks before continuing.

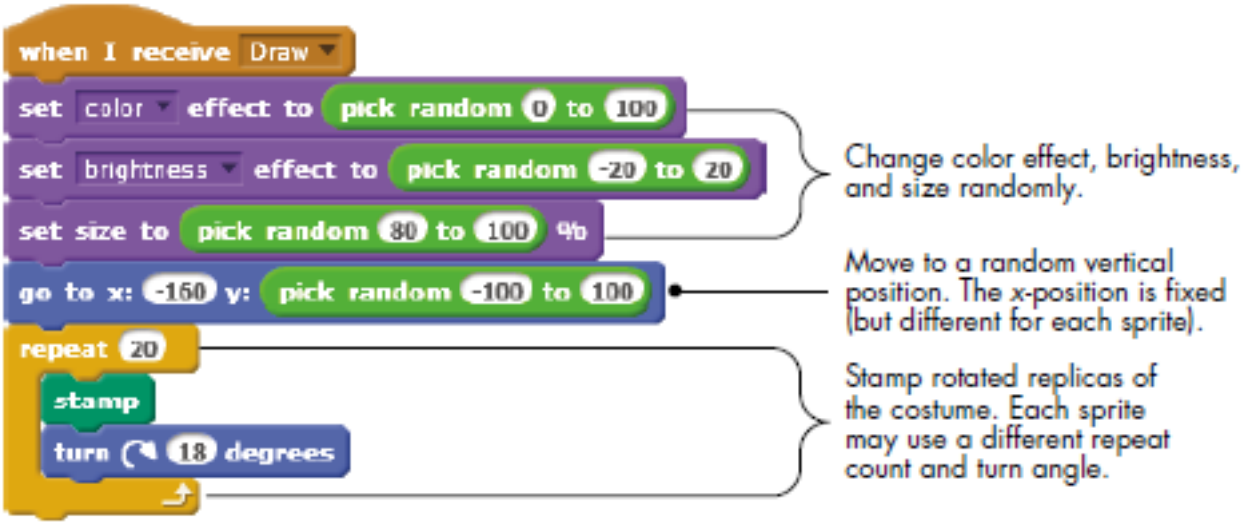
Message Broadcasting

OPEN PROJECT Flowers.sb2 FROM MODULE 4 http://tinyurl.com/scratch-modules Use File -> Upload from your computer to open it



To see multiple sprites respond to the same broadcast message, let's create an application that draws several flowers on the Stage in response to a mouse click. The Flowers application contains five sprites (named Flower1 through Flower5) that are responsible for drawing five flowers on the Stage. Each sprite has its own costume, as shown in picture above. Note how the background of each costume is transparent. Note also the location of the center of rotation for each costume (marked with the crossed lines).

When a sprite receives a message to draw its flower, it will stamp multiple rotated copies of its costume on the Stage. When you click the mouse on the Stage, the Stage detects the mouse click using the when this sprite clicked block. In response, it clears its background and broadcasts a message called Draw. All five sprites respond to this message by executing a script similar to the one shown below:



Open the application (named Flowers.sb2) and run it to see how it works. Despite its simplicity, its output is intriguing. Design different costumes to create different types of flowers. Change the costumes' centers to discover even more interesting flower designs.

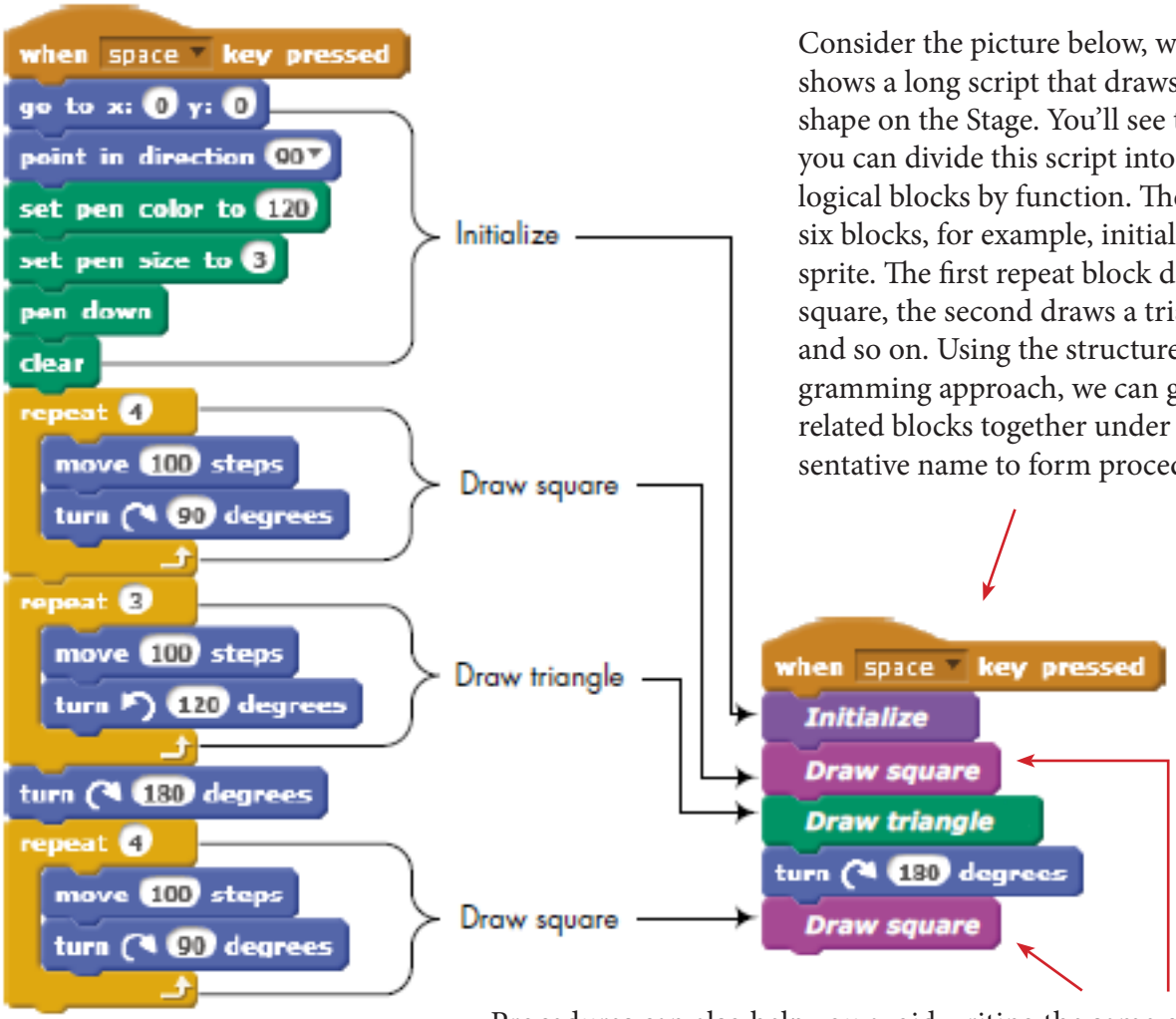
Procedures

The scripts that you've written up to this point are relatively short and simple. Eventually, you'll write longer, more complex scripts that contain hundreds of blocks, and understanding and maintaining them will become a real challenge.

An approach known as structured programming was developed in the mid- 1960s to simplify the process of writing, understanding, and maintaining computer programs. Instead of having you write a single large program, this approach calls for dividing the program into smaller pieces, each of which solves one part of the overall task.

Consider, for example, the process of baking a cake. You may not think about the individual steps as you bake, but the process follows a precise recipe that lists the necessary steps. The recipe might include instructions like (1) mix 4 eggs, 2 oz of flour, and 1 cup of water; (2) put the mixture in a pan; (3) put the pan in the oven; (4) bake for 1 hour at 350°F; and so on. In essence, the recipe breaks down the problem of baking a cake into distinct logical steps.

Similarly, when you design a solution for your programming problem, it helps to break the problem down into manageable, "mind-sized" bites. This approach helps you maintain a clear view of the whole program and the relationships between its component parts.

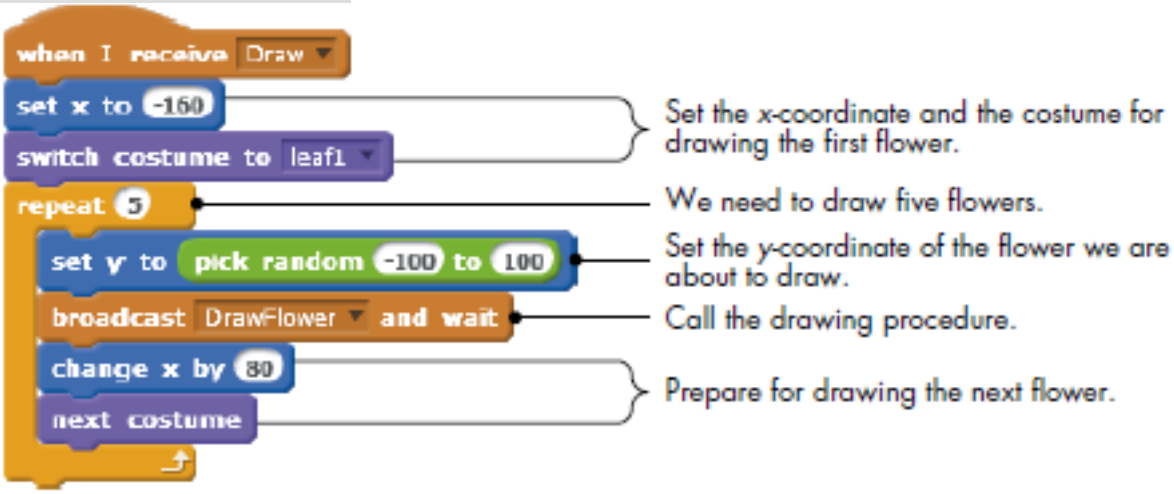


Consider the picture below, which shows a long script that draws a shape on the Stage. You'll see that you can divide this script into smaller logical blocks by function. The first six blocks, for example, initialize the sprite. The first repeat block draws a square, the second draws a triangle, and so on. Using the structured programming approach, we can group related blocks together under a representative name to form procedures.

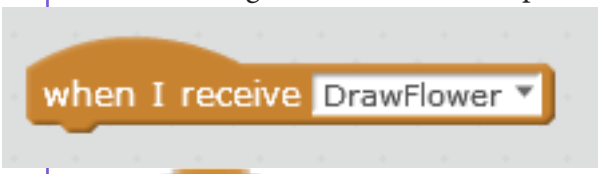
Procedures can also help you avoid writing the same code twice. If a set of commands is executed in several places in a program, you can write a procedure that performs these commands and use it instead. This strategy to avoid duplicating code is referred to as code reuse. Note, for example, how the Draw square procedure was reused in the code above

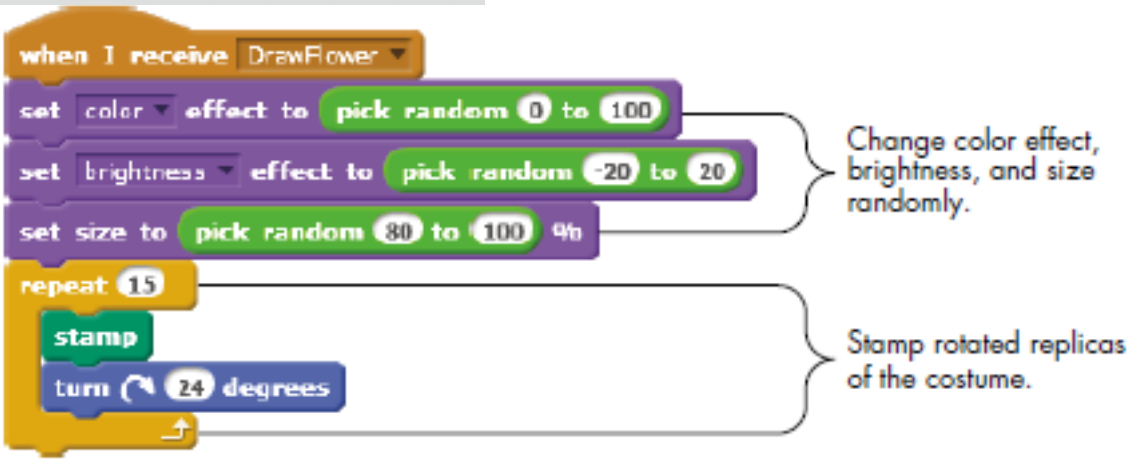
Procedures with message broadcasting

OPEN PROJECT Flowers2.sb2



The project contains the new version of the Flowers program. The script for the Stage is the same as before (the Stage broadcasts a Draw message when it detects a mouse click), but this time, our program uses only one sprite instead of five. This sprite has five costumes, leaf1 through leaf5, and will call a procedure to draw a flower for each costume. Since we have a single sprite, we only need one copy of the drawing code (not the five duplicate scripts we had in our first version). This makes the program smaller and the code easier to understand. When the sprite in this application receives the Draw message, it executes the script shown in the picture above

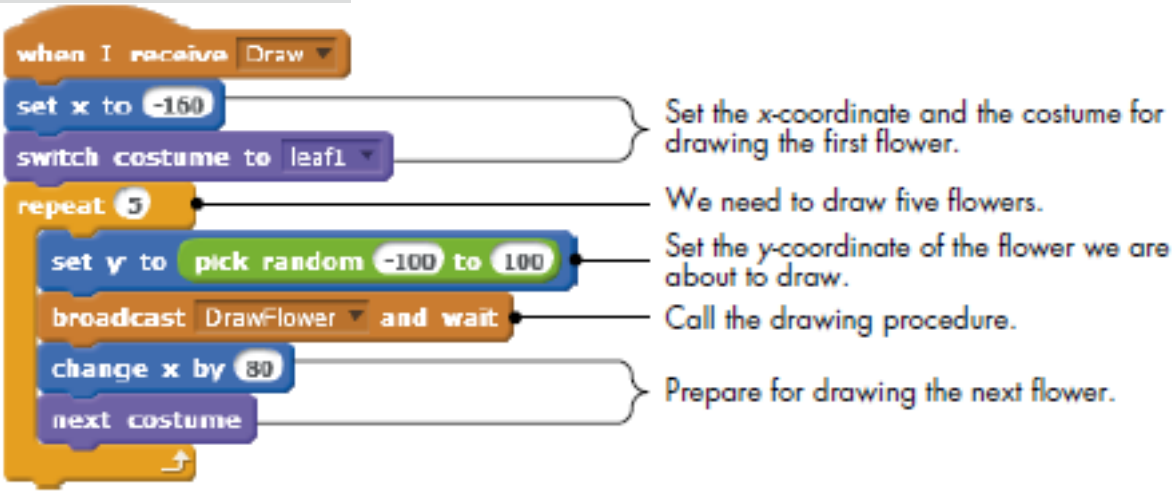




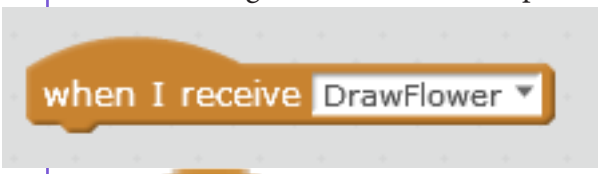
The DrawFlower procedure, shown above, sets random values for the color effect, brightness, and sprite size before stamping rotated versions of the current costume to draw a flower. While the first version of the program contained five sprites and five repeated scripts, the second version achieves the same result using a single sprite that calls one procedure for drawing all five flowers. Open Flowers.sb2 and Flowers2.sb2 in two tabs of your browser and compare them. Isn't the new version much simpler to follow? Using procedures lets you make smaller programs that are easier to understand and maintain. This will become more beneficial as you write programs to perform more complex tasks

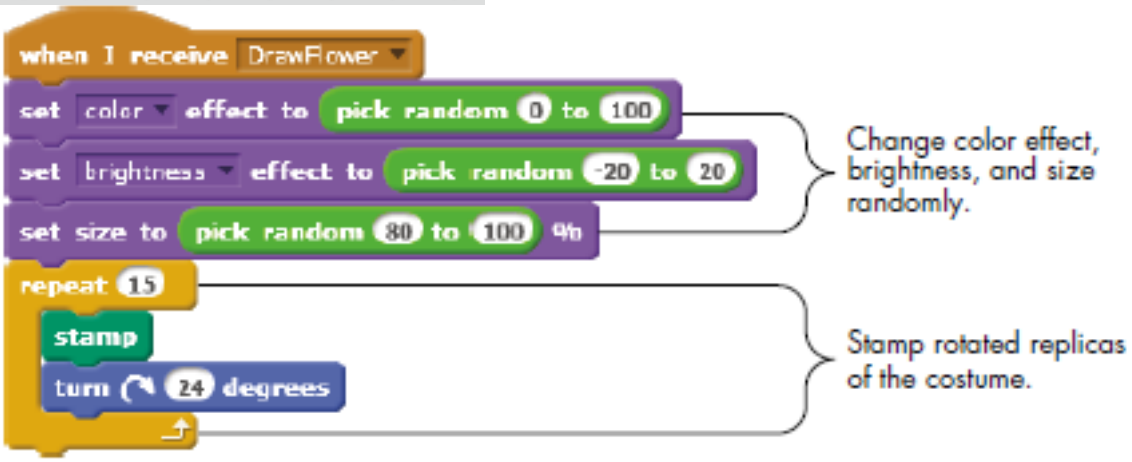
Procedures with message broadcasting

OPEN PROJECT **Flowers2.sb2**



The project contains the new version of the Flowers program. The script for the Stage is the same as before (the Stage broadcasts a Draw message when it detects a mouse click), but this time, our program uses only one sprite instead of five. This sprite has five costumes, leaf1 through leaf5, and will call a procedure to draw a flower for each costume. Since we have a single sprite, we only need one copy of the drawing code (not the five duplicate scripts we had in our first version). This makes the program smaller and the code easier to understand. When the sprite in this application receives the Draw message, it executes the script shown in the picture above





The DrawFlower procedure, shown above, sets random values for the color effect, brightness, and sprite size before stamping rotated versions of the current costume to draw a flower. While the first version of the program contained five sprites and five repeated scripts, the second version achieves the same result using a single sprite that calls one procedure for drawing all five flowers. Open Flowers.sb2 and Flowers2.sb2 in two tabs of your browser and compare them. Isn't the new version much simpler to follow? Using procedures lets you make smaller programs that are easier to understand and maintain. This will become more beneficial as you write programs to perform more complex tasks

Procedures with your own blocks

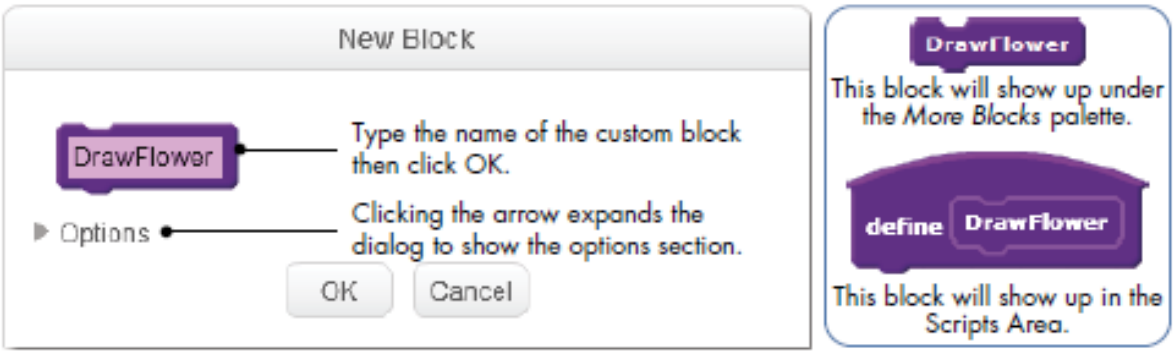
OPEN PROJECT **Flowers2.sb2**

As of Scratch 2, you can also create your own custom blocks. After you make a custom block, it should appear in the More Blocks palette, where you can use it as you would any other Scratch block. To show you how to create and use these blocks, we'll modify the Flowers2 program we discussed in the last section to use a custom block for the DrawFlower procedure. The following steps will guide you through creating this new version of the application.

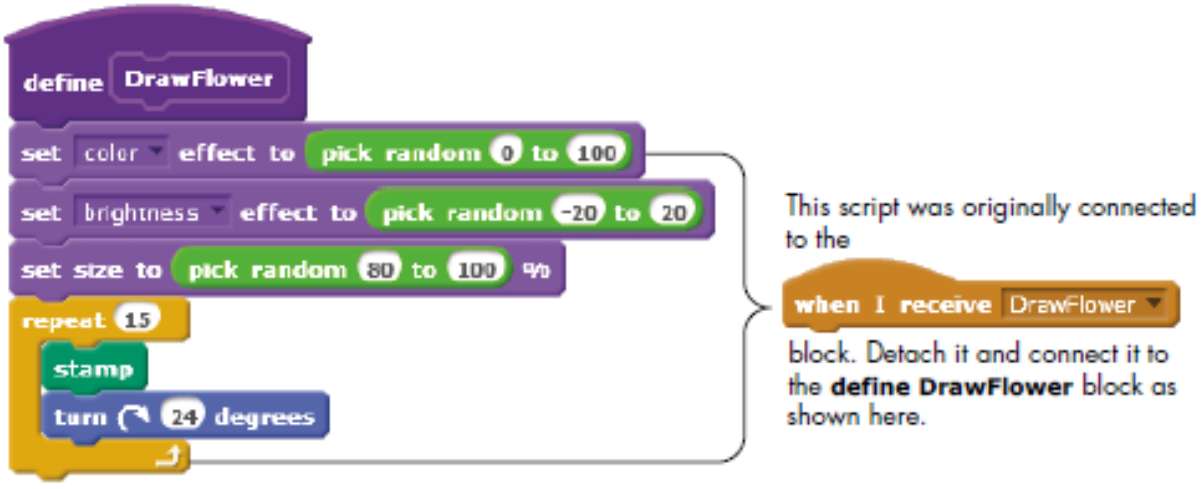
Step 1. Open the Flowers2.sb2 file that you looked at in the previous section.

Step 2. Click the thumbnail of the Flower sprite to select it.
Then select the More Blocks palette and click Make a Block. You should see the dialog shown below.

Type DrawFlower for the block's name and click OK. A new function block called DrawFlower should appear under the More Blocks palette, and a define DrawFlower block should appear in the Scripts Area as shown in the figure (right).

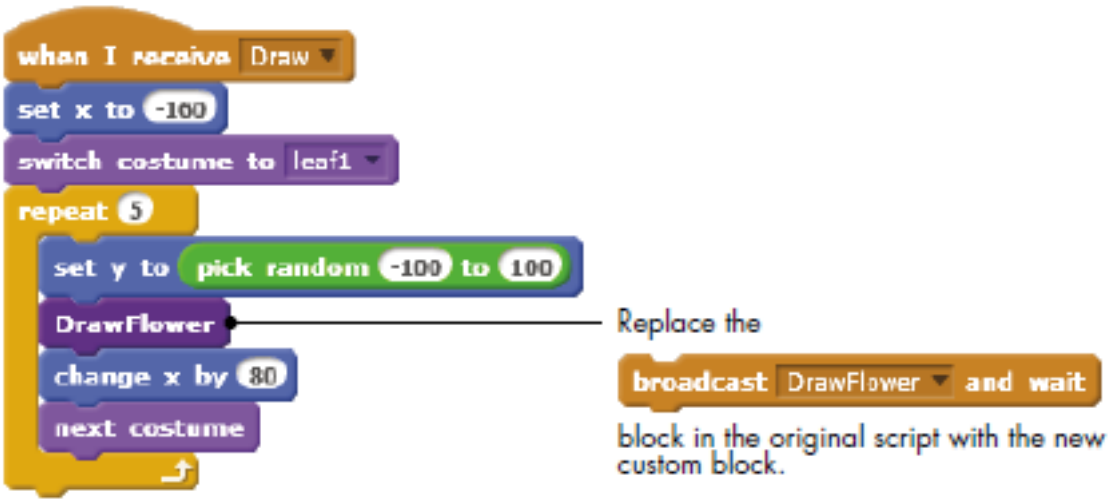


Step 3. Detach the script connected to the when I receive DrawFlower block and connect it to the define DrawFlower block, as shown below. This results in a new procedure, called DrawFlower, that is implemented as a custom block. Delete the when I receive DrawFlower block because it is no longer needed.



Procedures with your own blocks

Step 4. Now that we've created a DrawFlower procedure, we just need to call it from the Draw message handler. Modify the Draw message handler as shown in picture below. Note that we only replaced the broadcast DrawFlower and wait block with our new DrawFlower custom block.

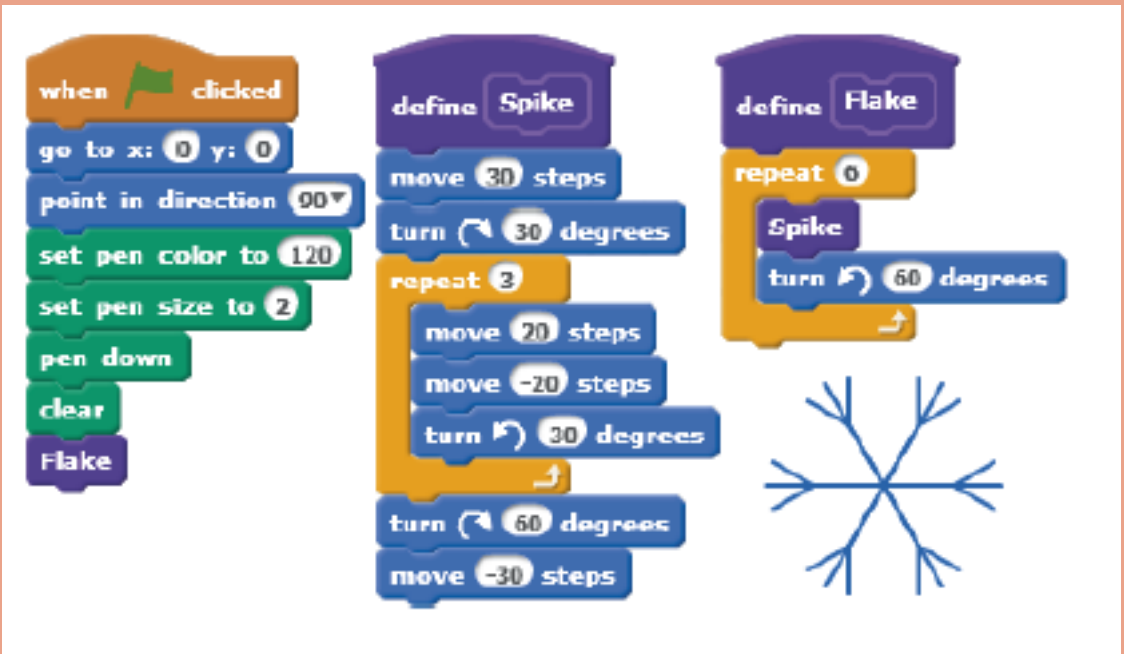


The program is now complete, and you can test it. Click the mouse on the Stage to verify that the program still works as before.

CHALLENGE TASKS

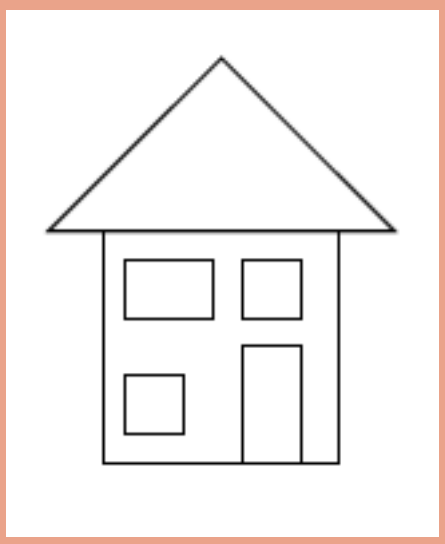
1. Write different procedures to draw each letter of your name. Name each procedure for the letter that it draws. Then write a script that calls these procedures so you can draw your name on the Stage.

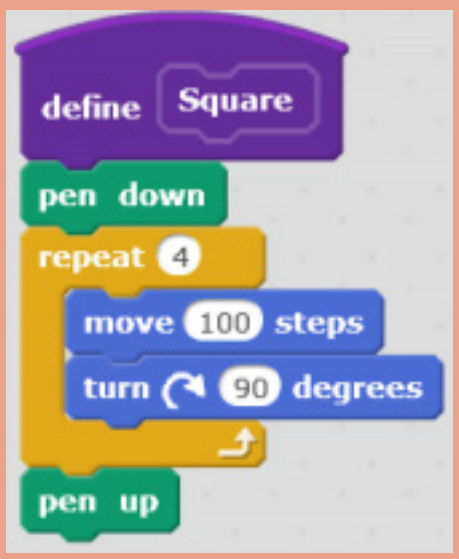
2. Create the program shown below, run it, and explain how it works.



CHALLENGE TASKS

3. Write a procedure to create the house shown below. Start by writing small procedures that draw small parts of the house (for example, door, roof, windows, and so on). Then combine these procedures to create the entire house.





If you have some remiaining time, pick a project from the following website https://scratch.mit.edu/starter_projects/ and make it your own.