Level Three: Recursion

NATIONAL
PEER REVIEWED
23003DD

Discovering Computer Science & Programming through Scratch

Updated for Scratch 3.0

About this Guide

This third volume in a collection of three (so far) books on Scratch programming focuses on an interesting, powerful, and important technique from computer science called recursion. A recursive program is one which "calls itself." In Scratch, this means using a block for the script within the very script that defines the block. Recursion makes solving many problems easier, and also allows one to do some things that would be very difficult to do without it, such as drawing intricate fractals.

While you need not have previously gone through the two earlier volumes, it will help to have done so. At the very least, we assume you are comfortable programming in the Scratch environment, and can use blocks from the motion category, from the control category, and that you know how to create and use variables. More generally, because recursion can be a confusing topic, it will help if you have enough programming experience in Scratch so that you can focus on the idea of recursion, and are not simultaneously learning to program.

We hope you find the projects in this book interesting, illuminating, and, most of all, fun! And, that the projects inspire you to go on to further study of topics in computer science.

Authors:

Lenny Pitt, Professor Emeritus of Computer Science, University of Illinois

Judy Rocke, Curriculum Development Specialist, Office for Mathematics, Science, and Technology Education (MSTE), University of Illinois

Jana Sebestik, Assistant Director STEM Curriculum Design, MSTE, University of Illinois

Support for this guide is provided by the 4-H Computing Connections (CS4H) project funded by the University of Illinois Extension and Outreach Initiative and also by the Department of Energy and the Department of Homeland Security under Award Number DE-OE0000780.

Layout and Design:

Christina Tran, Graphic Designer, MSTE, University of Illinois, iamchristinatran.com

Table of Contents

For the Facilitator

This is the third computer science manual in a collection of three (so far). The activities in this book assume that the user is comfortable programming in the Scratch environment and can use blocks from the motion and control categories and knows how to create and use variables. This book focuses on an interesting, powerful, and important technique from computer science called recursion. A recursive program "calls itself." In Scratch, this means using a block for the script within the very script that defines the block. Recursion makes solving many problems easier, and also allows one to do some things that would be very difficult to do without it, such as drawing intricate fractals.

This curriculum provides youth with a series of tutorials and challenges within the Scratch environment. Some of the activities are short and may take only thirty minutes, but others are more complex and offer opportunities to explore. Young people can work on the activities individually, with partners, or in a guided instructional setting. If students are working together, it is important to make sure that each student has equal time at the keyboard. It will also be helpful if each youth has their own guidebook.

As a facilitator of this project, encourage youth to talk about what they learn as they try new scripts and find new blocks. Youth will learn faster and more, when they discuss their projects with others. The Scratch community encourages users to share their projects on the Scratch website and to remix others' projects. Just be sure to give credit to the original project creator. There are millions of registered Scratch users sharing projects. Join the fun!

This curriculum was written for youth in Grades 5-12, but may be used and adapted for younger and older audiences, based on experience.

All three levels of Discovering Computer Science & Programming through Scratch were written using both the CSTA (Computer Science Teachers Association) K-12 Computer Science Standards and the ISTE (International Society for Technology in Education) Standards for Students as guidance. In addition to developing computer programming skills and providing computing practice, these learning materials offer opportunities for collaboration and stimulate computational thinking. Activities have been designed to promote creative design and encourage empowered learners. The global Scratch community emphasizes positive digital citizenship and responsible collaboration.

Additionally, Using Mathematics and Computational Thinking is one of the Next Generation Science Standards Scientific and Engineering Practices, while these lessons explicitly address the following Common Core State Standards for Mathematical Practice:

MP1: Make sense of problems and persevere in solving them.
MP2: Reason abstractly and quantitatively
MP6: Attend to precision
MP8: Look for and express regularity in repeated reasoning.

Parameters

▶ Creating Blocks that Accept Input

Open a new tab**,** and then open the ***Parameter*** activity at:
scratch.mit.edu/projects/87534863

Click on the **smallSquare** block. What do you see?
Click on the **largeSquare** block.
What do you notice about what the two blocks do? How are the scripts that define them ***alike***? How are they ***different***?

In computer science, a key concept is that of generalization. We try to find more general ways to solve problems, so that a single solution or program can be used to solve different versions of the same problem. Usually when you notice that you're writing scripts that are similar, it is a good time to consider whether there is a more general way to solve the problem so that you need only one script.

Now look at the **square __** block. We created this new block, which accepts a value, or an **argument**. The argument, which is typed into the white oval before the block is used, tells the sprite how long to make each side of the square. Type a number (try 75 or 100) into the open oval of the **square__** block. Then click on the block. Try other numbers. Make a really big square and a very small one.

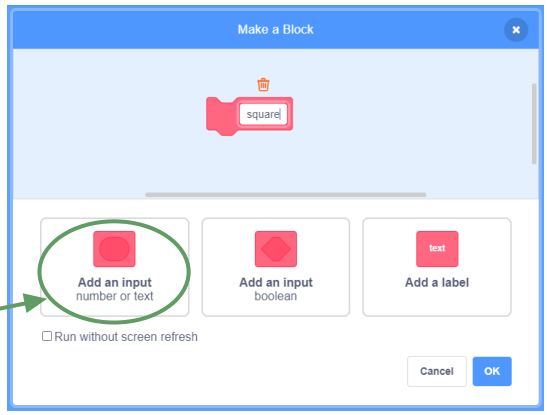
You have been using commands like this since you began using Scratch. For example, the **move** and **turn** blocks accept arguments to tell the sprite how far to move or turn. The **go to x:__ y:__** block accepts arguments to tell the sprite where to go to, and the **say__** block accepts an argument that tells the sprite what to say. Here, we've used an argument to tell the square block how big to draw the square.

But how does a script like **square__** tell the sprite how far to move, when the value of the argument (say, 75) isn't known until the script is run? We can't put in "move 75 steps", because a number other than 75 might be input when the block is used.

This is solved by using a special variable called a **parameter**. For example, we might have a parameter called **sideLength**, and then use the block **move sideLength steps**. The value of **sideLength** will be 75 if the user enters 75 into the **square__** block. Or it will be 100 if they entered 100. So, a parameter is used within the block as a placeholder for the argument that will be given by the user when the block is used.

Let's go over how to create the script **define square** with a parameter.

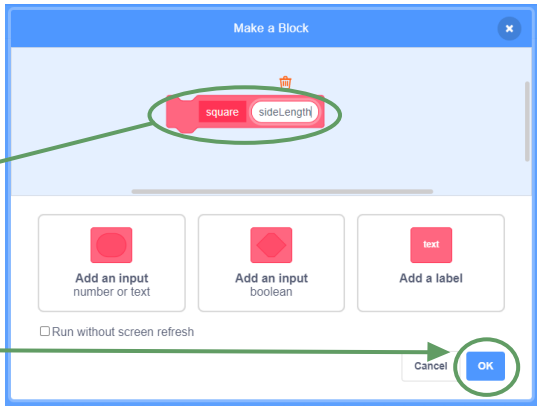
Open a new tab and a **new Scratch file**.

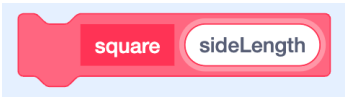


1. Click the **My Blocks** category. Then click **Make a Block**. Name the new block, **square**.

2. Click **Add an input number or text**.

3. The **square** block expands so you can name the first parameter. Click inside the white oval to change the parameter name to **sideLength**.

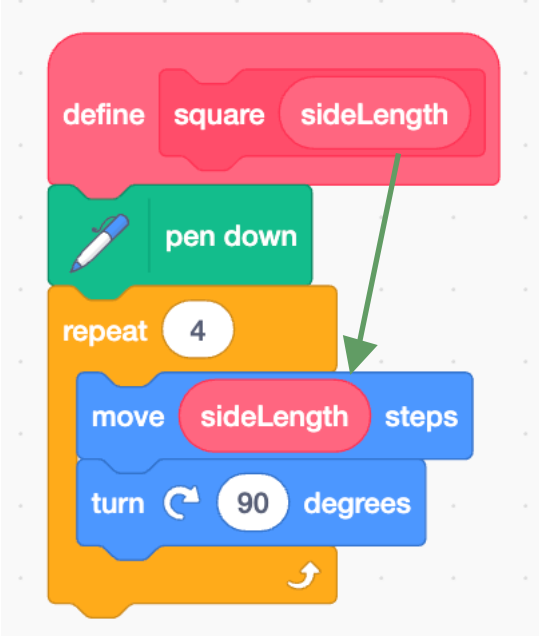




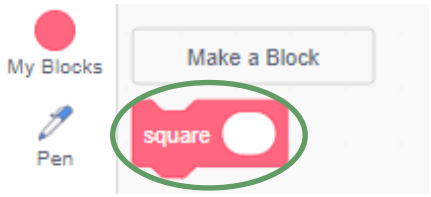
4. Then, click **OK**.

Give a parameter a name that clearly explains its meaning, just like when you name a variable.

5. This **define** block will appear in the script area. Program the new **square__** block as pictured so that it can create a square of any size.



6. What should we fill in for the **move__ steps** block? This is where the parameter comes in. When the **square___** block is run, a number will be input. We don't know now what that number will be. We use the parameter **sideLength** as a placeholder to represent that number. Drag **sideLength** from the **define** block at the top of the **define** script, and put it into the **move__ steps** white circle, as shown in the picture. This tells the script that when the script is run, the amount to be moved is whatever value was input into the **square___** block.

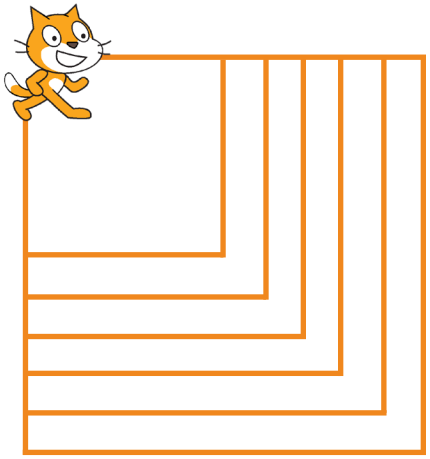


7 Use the new customized **square** block in a script. Pull the square block from the **My Blocks** category to use.

Create this script and then click the green flag.

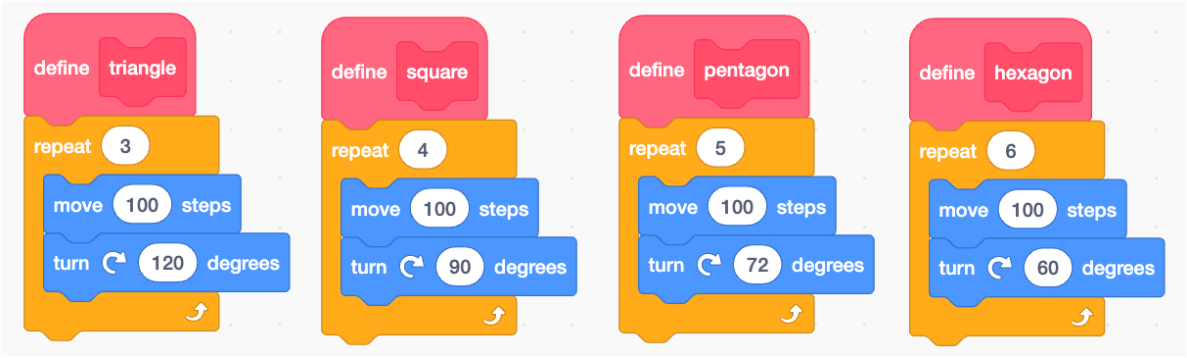
8 Change the input number to make a larger square. Make a square in the lower left corner of the stage.

9 Create a new script that uses the square block to draw squares of several sizes to look like the squares below.



10 Define a new block, and then create a script that draws a triangle with a parameter for the length of sides.

▶ Generalize Further



Notice the scripts shown above are very similar. Because of this, it should be no surprise that we can create a ***single*** script that draws ***any*** polygon with ***any*** side length.

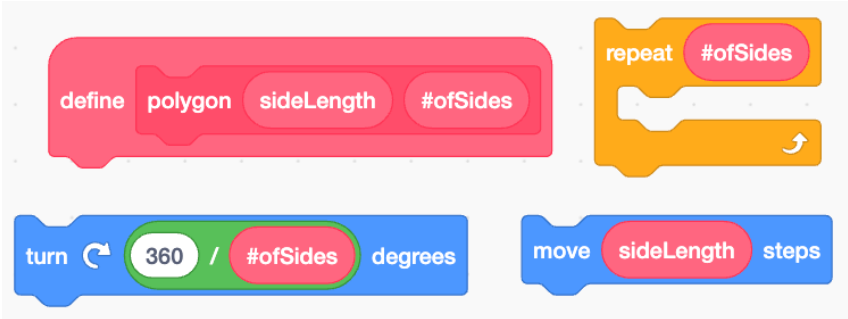
The script will have *two parameters*: one for the side length, and one for the number of sides.

1. Open a **new Scratch file**, click on the **My Blocks** category, and **Make a Block**. Name the new block **polygon**.

2. Click **Add an input number or text** twice and name the two parameters **sideLength** and **#ofSides**. Click **OK**.

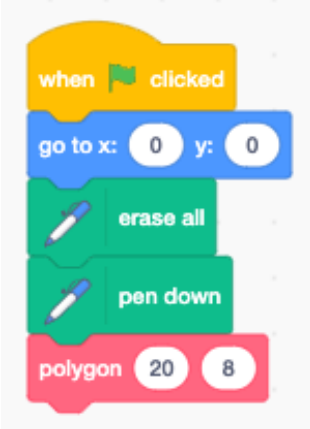
> Remember from the Level 1 book, the total number of turns a polygon needs to complete is a full circle turn of 360 degrees.
>
> A triangle makes 3 exterior angles with turns of 120° (3 * 120 = 360), a square makes 4 turns of 90° (4 * 90 = 360), and a octagon makes 8 turns of 45° (8 * 45 = 360).

3. Create the script that defines the polygon block using these blocks. Add other blocks if you need them. Drag the parameters to use as input in the repeat, turn, and move blocks.

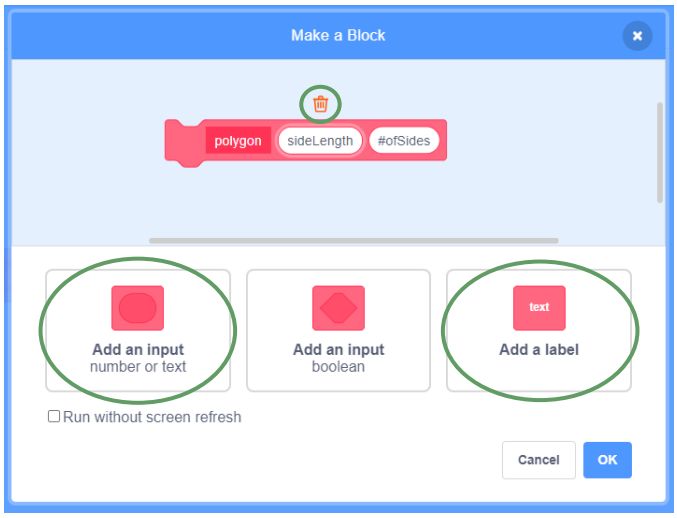


4. Predict what shape will be created when the green flag is clicked on this script. Does it make a difference if the order of the numbers is reversed?

   Create and test the script. Change the order of the numbers and test the script. What changes?



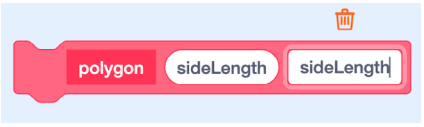
5 Change the input numbers in the **polygon** block to create a square, a pentagon, a hexagon, and other regular polygons.



6 It's easier for the user when we label the parameters so they know which numbers to put in each space. To do this, right click on the **define polygon** block and choose **edit** from the drop down menu.

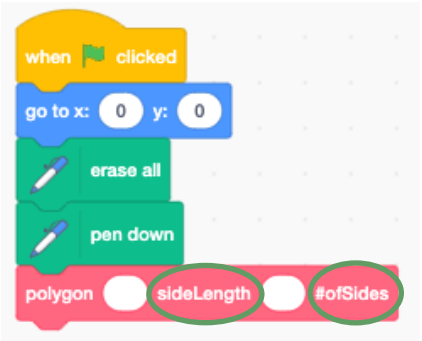
7 To delete both parameters, click on the white oval with the parameter name. Then click the trash can just above it.

8 Click **Add an input number or text**. Type **sideLength** in the pink box. Click **text Add a Label** and type **sideLength** again. Now the block has an input parameter and a label for the parameter.



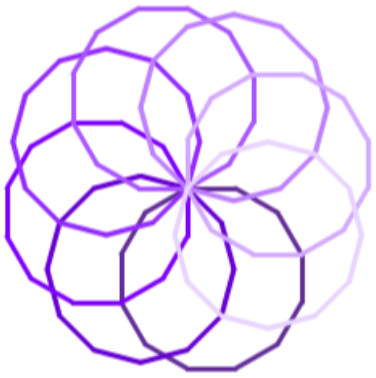
9 To add a second parameter and a label for **#ofSides**, click **Add an input number or text**. Type **#ofSides** in the pink box. Click **text Add a Label** and type **#ofSides** again. Then click **OK**.

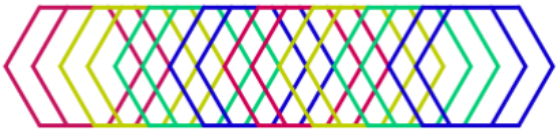
10 The **define polygon** block and the **polygon** block now both have labels for each of the parameters.



Challenge

- Write scripts using your new block to create multiple polygon pictures like these:





I pledge my head to clearer thinking,

my heart to greater loyalty,

my hands to larger service, and

my health to better living,

for my club, my community,
my country and my world.