Level Three: Recursion

NATIONAL
PEER REVIEWED
01802Y

Discovering Computer Science & Programming through Scratch

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 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

This guide was created as a part of the 4-H Computing Connections (CS4H) project funded by the University of Illinois Extension and Outreach Initiative.

Layout and Design:

Christina Tran, University of Illinois, College of LAS, Department of Mathematics, MSTE

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 his/her 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 nearly 20,000,000 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.

Notes:

Parameters

▶ Creating Blocks that Accept Input

Open a new tab**,** and then open the ***Parameter*** activity at:
https://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 ask if 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. 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 **Square__** block accepts an argument that tells the sprite how far to move to make the side of the square. Type a number (try 75 or 100) into the open circle of the **Square__** block. Then click on the block. Try other numbers. Make a really big square and a very small one.

This is an example of a script with a **parameter**. A parameter is a special variable. When we type a value, for example 75, into the **Square__** block, the parameter inside the **define Square__** gets the value 75. The parameter tells one script the value of the argument and when the script is run, it uses that value within the program.

We could have used a variable, **sideLength**, to draw a square. The script would have looked like this. So, how is a parameter different from a variable? A parameter can be used only by the script where it is defined. Whereas, a variable is available for use by any sprite or script in the program.

When would we want to use a parameter instead of a variable? In a more complicated project the value of a variable could be changed many times by many different scripts. Each script could set or change the variable to a different value using these blocks.

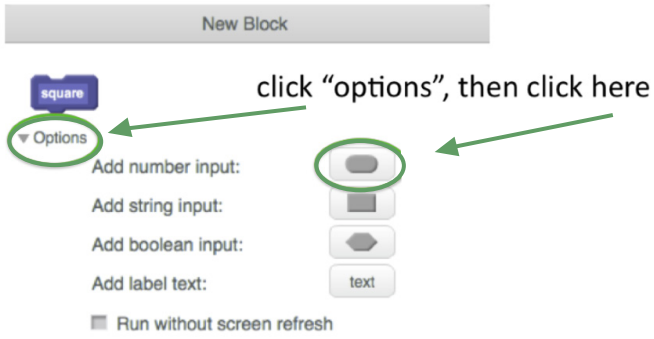
So we use a parameter when we want to associate a variable with a particular script.

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

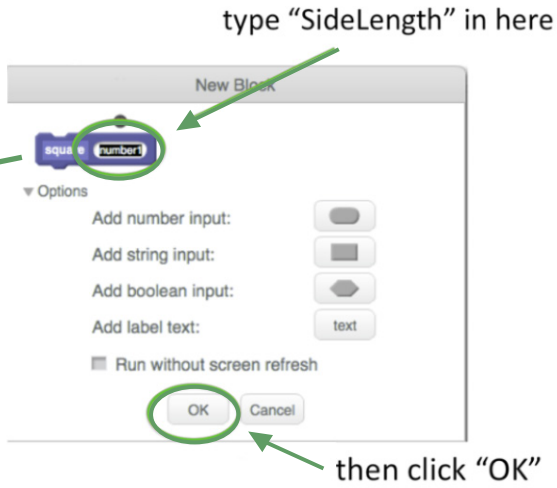
Open a new tab and a **new Scratch file**.

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

2. Click the oval to the right of **Add number input**.



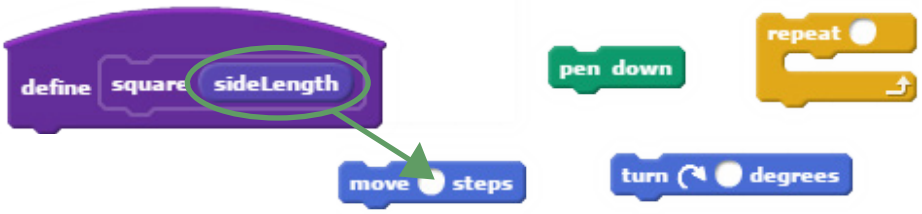
3. The **square** block expands so you can name the first parameter. Click inside the white oval to change the parameter name to **sideLength.**



4. Then, click **OK**.

Parameters are just like **variables** - except that they "belong" just to a specific script. Just like when you use variables, you should name a parameter with a name that clearly explains its meaning.

5. This **define** block will appear in the script area. Define the new **square** block by attaching a script that will create a square of any size. To do this, pull out these blocks.



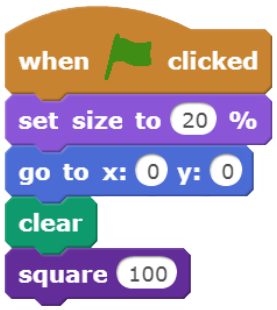
6. We want to be able to make a square of any size. Click and drag the **sideLength** parameter from the **define** block into the input circle in the **move** block.

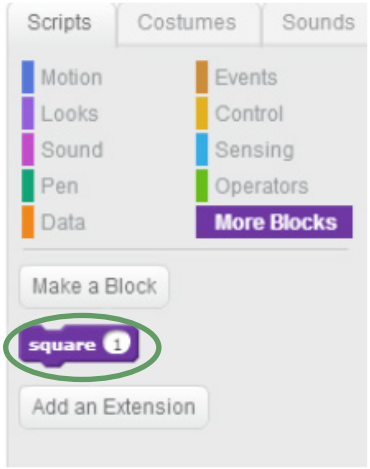
7. To draw a square the sprite moves and turns **90** degrees **4** times. Type these number inputs into the **turn** and **repeat** blocks.

8. Put the **move** and **turn** blocks inside the **repeat** block.

9. Attach the **pen down** block and the **repeat** group of blocks to the **define** block to create the script that defines our new square block.

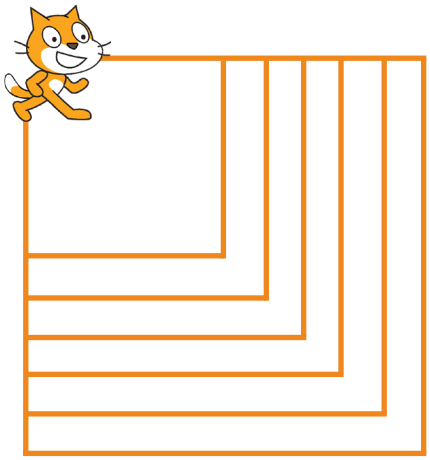
10. Use the new customized **square** block in a script. Pull the square block from the **More Blocks** category to use. Create this script:





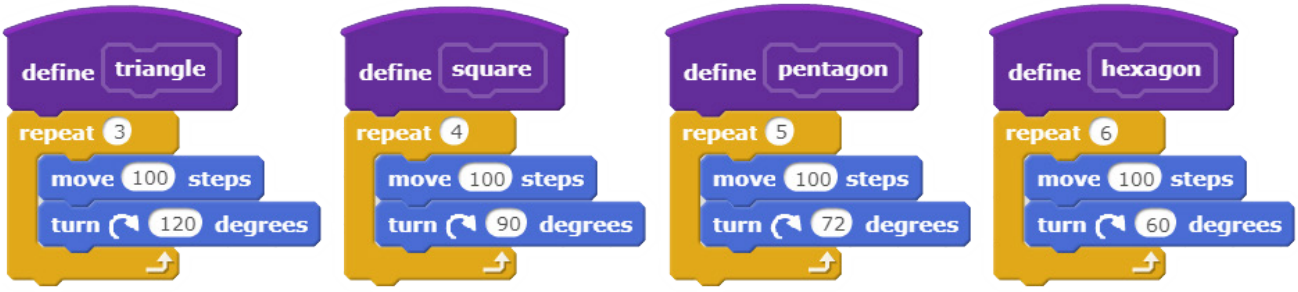
11. Change the input number to make a larger square. Make a square in the lower left corner of the stage.

12. Write a script that draws squares of several sizes to look like the squares below.



13. 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.

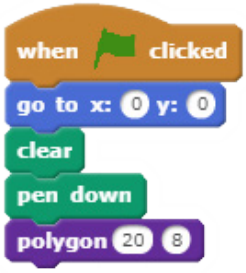
> Remember from the Level 1 book, the total number of turns a polygon needs to complete is a full circle turn of 360 degrees.
>
> A triangle makes 3 exterior angles with turns of 120° (3 * 120 = 360), a square makes 4 turns of 90° (4 * 90 = 360), and a octagon makes 8 turns of 45° (8 * 45 = 360).

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

2. Click the oval after **Add number input** twice and name the two parameters **sideLength** and **#ofSides**.

3. Create the script that defines the polygon block using these blocks. Add other blocks if you need them.

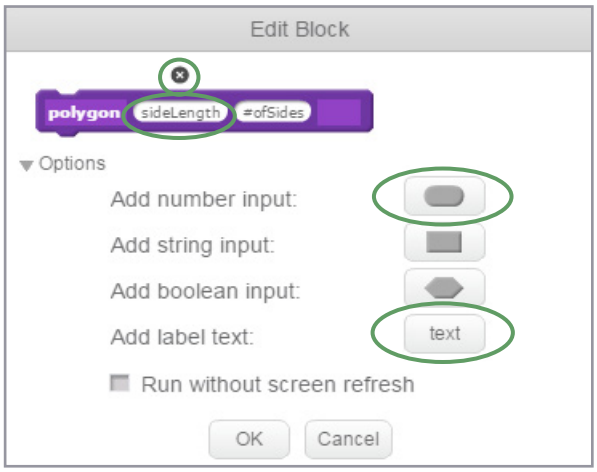
4. Predict what shape will be created when the green flag is clicked on this script. Why does this script use 20 and 8? Does it make a difference if the order of the numbers is reversed?



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

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

6. It makes it easier for the user if 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. Click **options**.

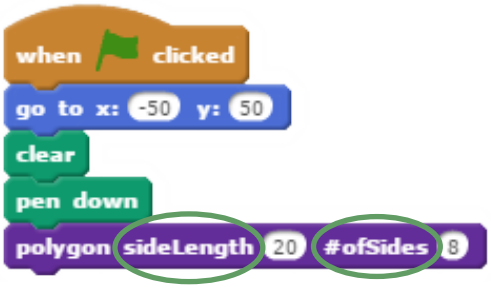


7. To delete both parameters, click on the white oval with the parameter name. Then click the x just above it.

8. Click on the word **text** to the right of **Add label text**. Type **sideLength** in the pink box. Click on the oval to add a number input and type **sideLength** again.

9. Click again on the word **text** to the right of **Add label text**. Type **#ofSides** in the pink box. Click on the oval to add a number input and type **#ofSides** again. 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: