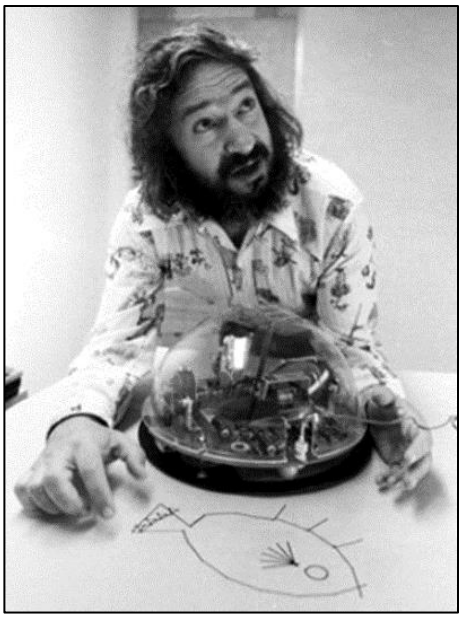
Chapter 1

Digital Designs and Artistic Patterns: Introduction to Turtle Graphics

Glen Bull and Joe Garofalo

Introduction

Robot Turtles



Seymore Papert with a Logo Turtle (credit: Cynthia Solomon, MIT)

Seymour Papert developed *Logo*, the first computing language designed for children, in the MIT Artificial Intelligence Laboratory in 1966. An MIT team led by Mitch Resnick subsequently developed a web-based successor to *Logo* named *Scratch*. With support from the MIT team, a version of *Scratch* named *Snap!* was designed for adults by a team at the University of California at Berkeley led by Brian Harvey working in collaboration with developer Jens Moenig.

Papert's vision of computers in schools embodied the idea that *the act of programming a computer can facilitate the process of learning subjects such as mathematics and language arts.* Papert believed that children can learn by teaching the computer through programming.

The robot turtle is the best-known feature of *Logo*. Papert described the *Logo* turtle as a "computational object-to-think-with." The early *Logo* turtles consisted of Plexiglas hemispheres that could be programmed to move about the floor. Later turtles included a pen that could be lowered to draw figures on rolls of paper.

Screen Turtles

The advent of video displays made it possible to create screen turtles in *Logo*. These turtles sometimes looked like an actual turtle (viewed from above) but often were represented by a triangle on the screen.

Examples of Screen Turtles

Securing a *Snap!* Account

A free account for *Snap!* can be obtained at the following web address:

https://snap.berkeley.edu/snap

Secure a *Snap!* account before continuing. Try each command as you read about it. For a comprehensive guide to *Snap!* commands, please refer to the online manual at the following web address:

https://snap.berkeley.edu/snap/help/SnapManual.pdf

Turtle Commands

The Command Palette

The types of commands available in *Snap!* are displayed in a *Command Palette* at the top of the left-hand side of the screen. For example, the *Motion* commands are currently highlighted in the palette below. Other categories of commands include *Looks*, *Sound*, *Pen*, *Control*, *Sensing*, *Operators*, and *Variables*. Note that each category is a different color. For example, *Motion* commands are purple.

Click on each of the categories in the *Command Palette* to get a sense of the types of commands that are found under each category. Use commands by dragging them into the script space to the right of the *Command Palette*.

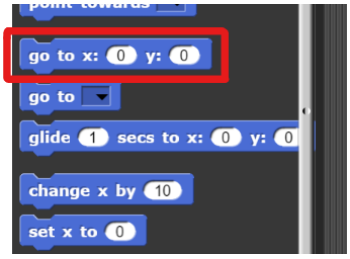
The Move Command

Drag the command **Move 10 Steps** the script space. Then click the code block. The turtle should move 10 steps forward (in the direction that the turtle is pointed) when this code block is clicked.

Try other values such as **100 Steps**. In this case, only one turtle is on the screen. However, it is possible to create multiple turtles. The term *Sprite* is also used as another name for a screen turtle.

Resetting the Turtle

If the turtle went off the screen in the last section, reset its position by clicking on the **Go To X_Y_** code block. You will use this frequently and may want to drag the command block into a corner of the command space for easy access.



The Turn Command

Then try the **Turn** command. Drag the **Turn Right 15 Degrees** code block into the script space. Enter the setting of 90 degrees into the code block.

Turn Right 90 Degrees

Click on the code block to execute the command. Watch the turtle rotate 90 degrees when the command is executed.

This example shows the *Turn Right* command. A *Turn Left* command is also available.

Combining Code Blocks

Combine the **Move** and **Turn** commands can by snapping the two code blocks together. Click any part of the combined code block to cause both commands to be executed in the order that they are listed (even though the order of the execution may not be obvious at this stage).

The name *Snap!* refers to the ability to snap code blocks together in the same manner as LEGO bricks. Practice snapping and unsnapping blocks together to become familiar with the mechanics of *Snap!* command blocks.

Pen Commands

The original floor turtle had a pen in its belly that could be raised and lowered. In a similar manner, **Pen Up** and **Pen Down** commands (found in the green *Pen* palette) enable the screen turtle to draw on the screen. Drag the **Pen Down** command into the script space. Click the **Pen Down** code block, and then click the **Move/Turn** block four times

Clearing the Command Space

Click the **Clear** command to erase any lines you have made. You will use **Clear** frequently and may want to drag the command block into a corner of the command space for easy access.

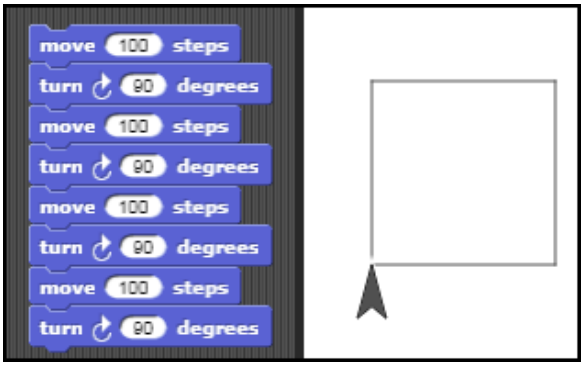
Repeating Commands

Repeating Commands to Draw a Square

Right clicking on any code block in the script space allows you to duplicate it. To duplicate a group of blocks, right click on the topmost block and click *Duplicate*.

Use the *Duplicate* option to create four copies of the **Move** and **Turn** commands.

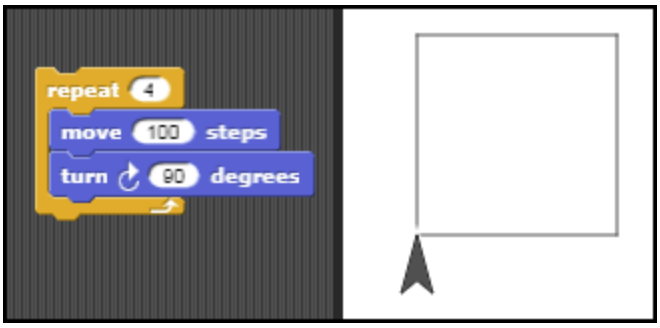
With the pen down, repeat the steps **Move 100 Steps** and **Turn 90 Degrees** four times to draw a square.



Then **Clear** the screen.

The Repeat Command

Use the **Repeat** statement to execute the commands with fewer lines of code. The **Repeat** command is found in the *Control* palette (highlighted in yellow). To use it, drag the command blocks you want to repeat into the empty space of the **Repeat** block and enter the number of times you want them to repeat.



The command **Repeat 4 [Move 100 Turn 90]** achieves the same result as duplicating the **Move** and **Turn** commands four times.

Defining New Commands

Teaching the Turtle a New Word

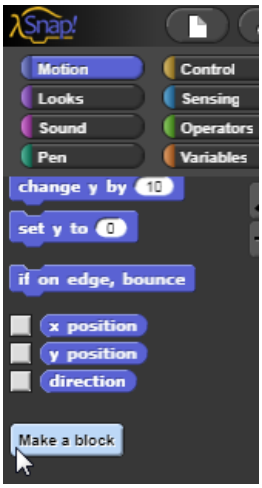
Papert's book, *Mindstorms: Children, Computers, and Powerful Ideas*, describes the way in which subjects such as mathematics can be introduced through turtle graphics:

> "The idea of programming is introduced through the metaphor of teaching the Turtle a new word. This is simply done, and children often begin their programming experience by programming the turtle to respond to new commands invented by the child …"

The family of programming languages that includes *Logo*, *Scratch*, and *Snap!* was designed for children to learn by adding their own extensions to the programming language.

The Make a Block Option

In *Snap!*, the **Make a Block** option is used to "teach the Turtle a new word." This option is found at the bottom of each palette of commands.



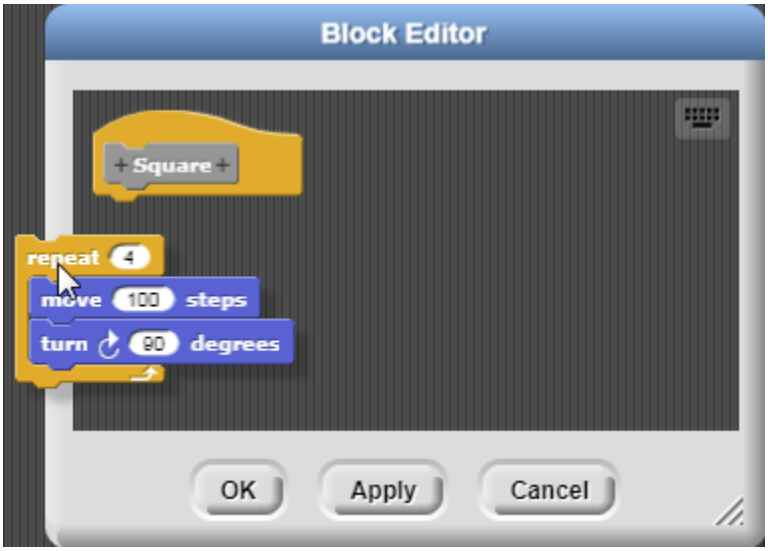
Click the *Make a Block* button to define a new command. Enter *Square* as the name of the new command. In most cases, the "for all sprites" option will be selected so that the new command will work with any sprite. Then click OK.

Defining a New Command - Square

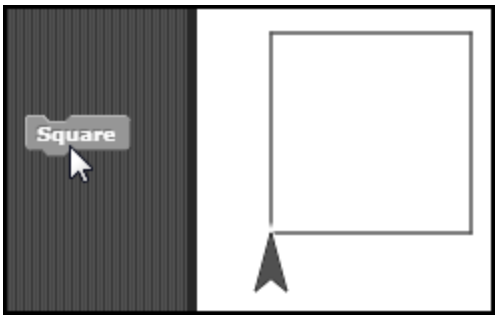
Next drag the previously developed block of code,

Repeat 4 [Move 100 Steps; Turn 90 Degrees]

into the *Block Editor* to define a new command named **Square**.



Click OK. A new command, **Square**, will appear at the bottom of the list of *Motion* blocks. It can now be used as though it were a built-in command.



Abstraction – A Key Computing Concept

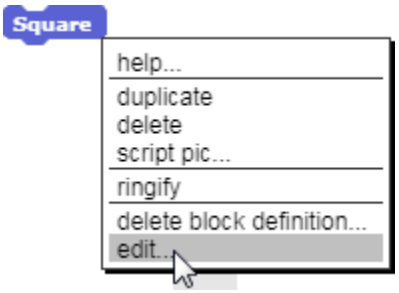
Creation of new commands is one of the most powerful concepts in computing. It makes it possible to work out complex sequences of commands and then assign a name to the code block. This process as known as *abstraction*. The underlying complexity of a long sequence of actions can then be hidden. Assuming the name of the new command is meaningful (and it is important to assign a name that accurately reflects the result that will occur), the programmer no longer has to deal with the underlying details.

The limited capacity of human memory imposes one of the key constraints on development of efficient code. Assigning a name to a complex process, thereby hiding the underlying details, makes it possible to create much more complex programs than otherwise would be possible.

Variables

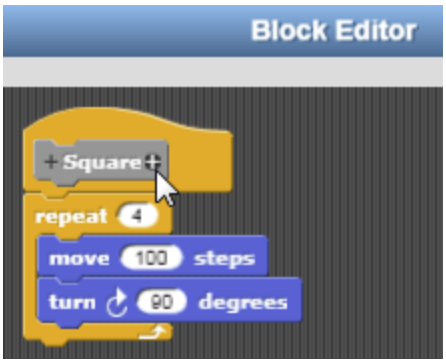
Editing the Square Procedure

The initial **Square** command is somewhat limited. It can only draw a square that is 100 steps on a side. Edit the **Square** command by right-clicking the name of the procedure. This produces a drop-down menu. Click the *Edit* option to edit the procedure.



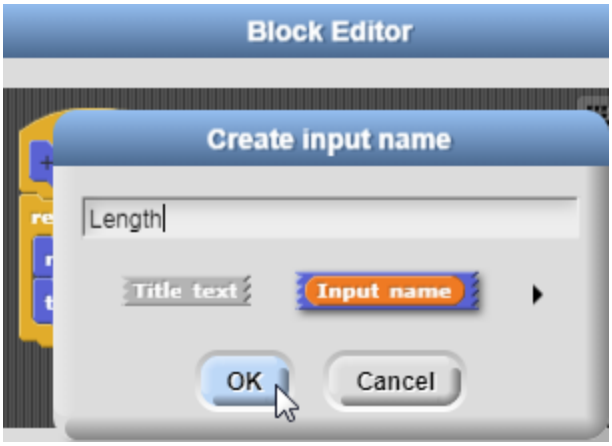
Adding an Input to the Procedure

Click the "Plus" sign (+) to the right of the title ***Square*** in the *Block Editor* to access another option, *Create Input Name*.



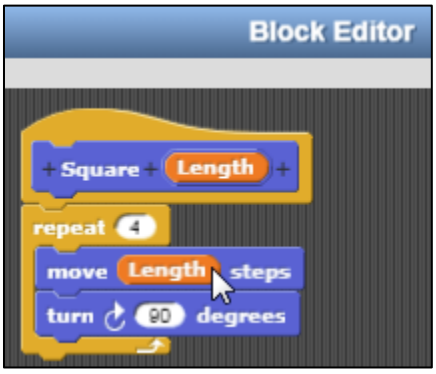
Naming the Variable - Size

Enter the name *Length* as the name of an input to the command. This name was chosen because we are going to use this variable to vary the length that the turtle travels as it completes each side of the square. Click OK to confirm this choice.

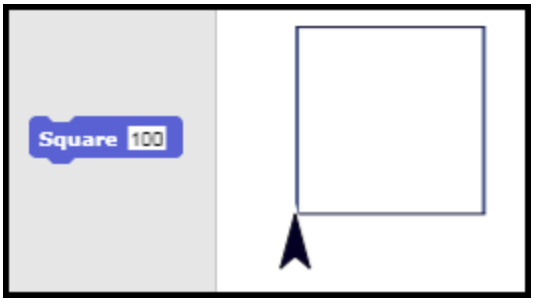


Using the Size Variable in the Square Procedure

Drag the orange oval labeled ***Length*** into the the *step* field of the **Move** command replacing the number.

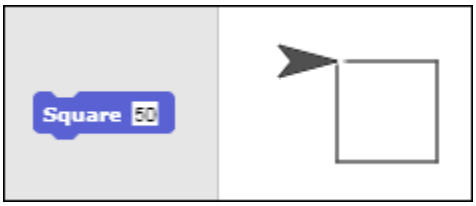


The input to the command **Square** can then be used to specify the length of each side of the square that will be drawn.



Drawing Squares of Varying Size

If 100 is entered as the input, a square that is 100 turtle steps on each side will be drawn. If 50 is entered as the input, a square that is 50 turtle steps on each side will be drawn.



Variables – A Key Computing Concept

The input to the command **Square** is known as a *variable* because it makes it possible to vary the size of the square drawn. Different values can be assigned to the variable known as *Length*. *Variables* are the second big idea in computing. Variables allow the actions of a procedure to be adjusted according to the needs of the situation.

Together, the ability to define new commands and create variables give computing languages much of their power. They enhance the ability of the programmer to create complex applications.

Global Variables

The Make a Variable Option

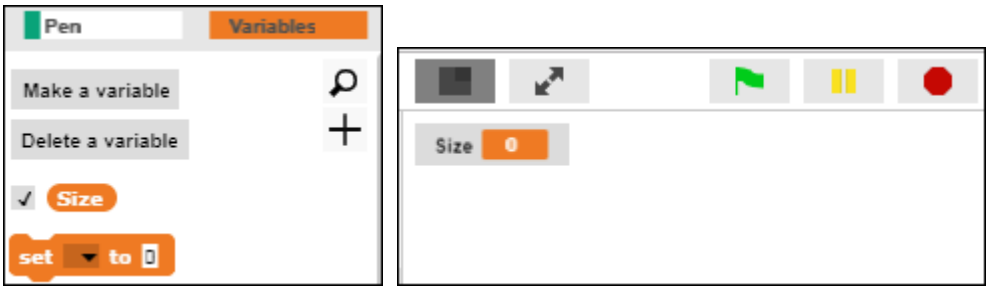
The variable ***Length*** only affects the block of code within the **Square** procedure. It is called a local variable because it *only* affects the local code within the **Square** procedure.

Global Variables can be used *across many* procedures and can also be created. To create a global variable, select the *Make a Variable* option found in the *Variable* command palette (orange). Variables created in this way can be used across multiple blocks of code. For example, a global variable could be used to specify the size of a square and a triangle. It could even be used to control creation of a series of squares of varying size.

The Variable Name Editor

Select the *Make a Variable* to access the *Variable Name* editor. Enter the name ***Size*** as the name of the new global variable. This variable will be used to vary the size of different polygons such as triangles and squares.

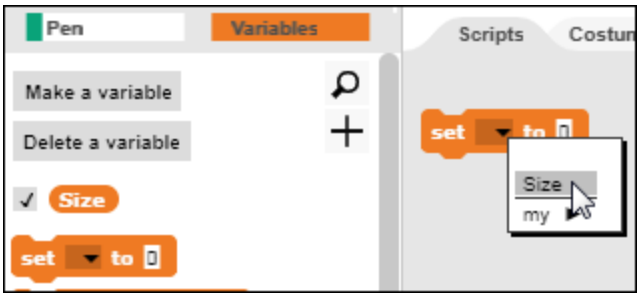
The new global variable, ***Size***, will appear at the top of the *Variables* palette



When the box beside the variable is checked, the value of the variable will be displayed on the stage (on the right-hand side of the screen). A default value of zero is assigned to new variables.

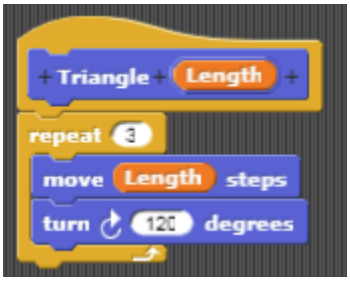
Assigning a Value to a Variable

Use the **Set** command to assign a value to a global variable. The **Set** command is found under the *Variables* palette.

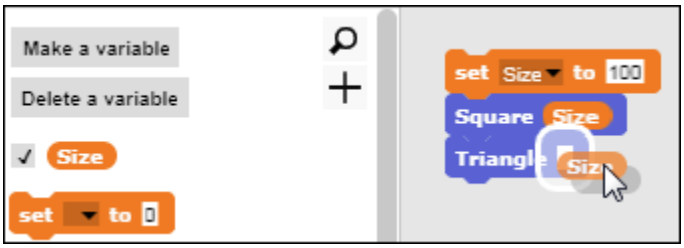


Enter a value of 100 for the global variable size.

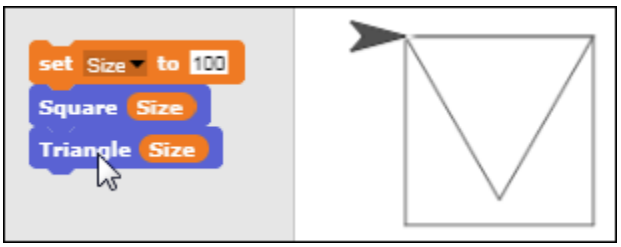
Define a **Triangle** procedure. Enter a value of 3 in the **Repeat** statement and a value of 120 degrees for the angle of each turn. Why does 120 degrees achieve the desired result?



Use the global variable *Size* to control the size of both a square and the triangle. Drag the orange oval labeled Size into the inputs of the **Square** and the **Triangle** procedures.

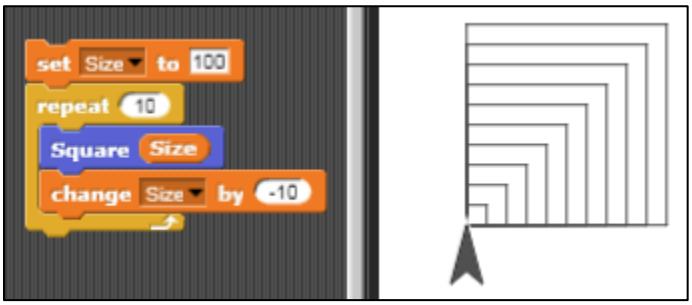


Then click the code block to draw a square and a triangle that are each 100 steps on a side. Experiment with other values for *Size*.



Repeating a Series of Squares

Combine the *Size* variable with a **Repeat** command and the **Square** procedure to create a series of squares. After each square is drawn, the value of the variable is decreased by 10 using the **Change** command. A series of squares decreasing in size results.



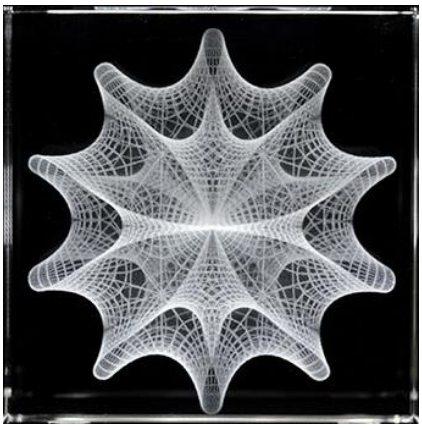
Algorithmic Thinking – A Key Computing Concept

Algorithmic thinking is the process of automating solutions through a series of ordered steps (sequencing) and conditional logic (flow of control), and is an essential characteristic of computing. Developing complex algorithms, such as the one above to draw a series of shapes, is an important skill to develop. *Snap!* provides a simple platform for constructing such algorithms by connecting and sequencing the command blocks. In a later section, we will explore conditional logic and controlling the flow of an algorithm in more depth.

Patterns in Art

Digital Artists

Artists like Bathsheba Grossman use mathematical patterns to create art. Their designs are etched in acrylic or glass using digital fabrication tools such as laser cutters.



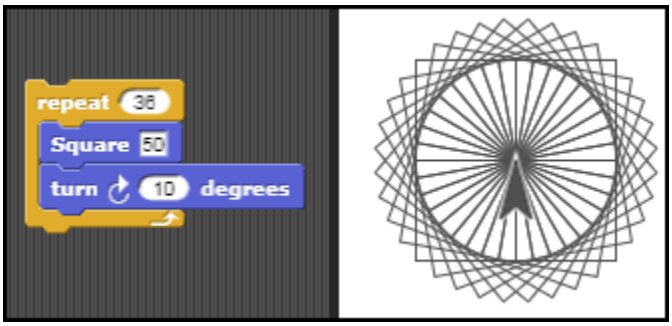
Bathsheba Sculpture Website

Spinning the Square to Create a Pattern

Create a pattern similar to those designed by digital artists. Begin by drawing a series of squares, turning slightly (10 degrees) before drawing each square in the series.

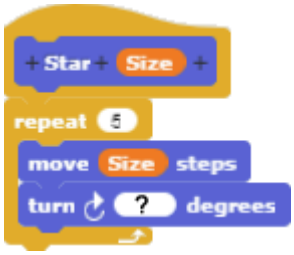
Repeat 36 [Square 50; Turn 10 Degrees]

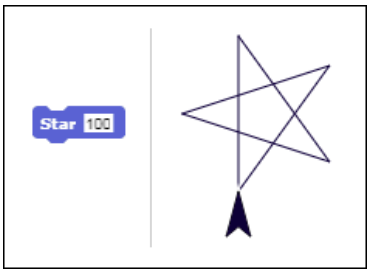
Rotating the turtle as it draws a series of squares results in the following pattern.



Drawing a Star

Use other geometric figures to create other patterns. Begin with a star. A five-pointed star requires a **Repeat** statement with five repetitions. Use experimentation or logic to determine the number of degrees that the turtle must turn.

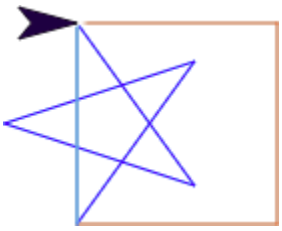




Try two or three different angles for the *Star* procedure. If you have not determined the correct angle after severl tries, consider the following.

Identifying the Correct Angle for the Star Procedure

The turtle turns 90 degrees to draw a square . It must be turned more than 90 degrees to create a star. But how much farther?

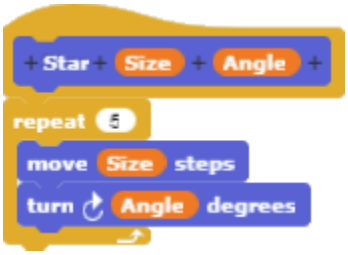


If the turtle does not turn far enough after drawing each line, the figure of the star will not be closed.

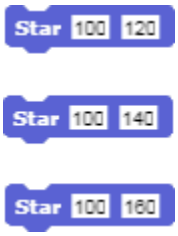
If the turtle turns too far after drawing each line, it will overshoot, creating a star that also is not well formed.

An Iterative Approach

Create a *Star* procedure with Size and Angle inputs.



Then try several angles such as 120 degrees, 140 degrees, and 160 degrees.



Which of the three values (120, 140 or 160 degrees) most nearly resembles a star? What is the most appropriate angle and why?

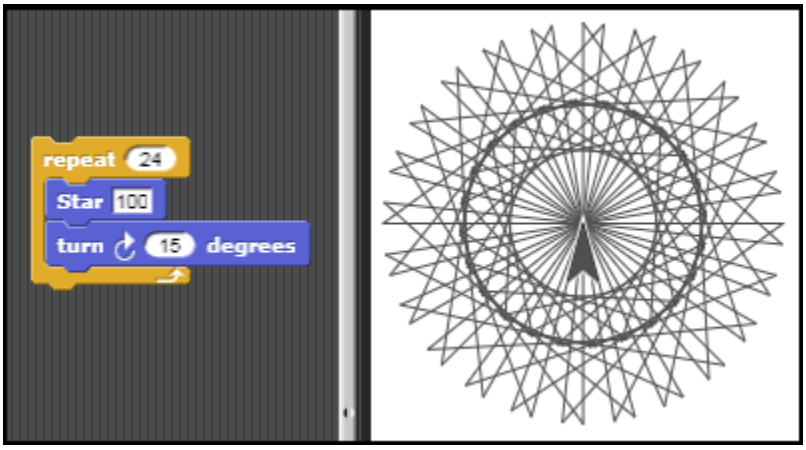
Debugging – A Key Computing Concept

Debugging is the process of identifying and correcting for mistakes, and is another essential to develop in computing. Just as a writer will find and correct grammatical mistakes in a rough draft, a coder will troubleshoot and debug their program. In the above example, we applied a bruteforce

approach to identifying the correct values to use in the **Star** command. Debugging is a process done frequently and occurs at almost every step in the development of a program.

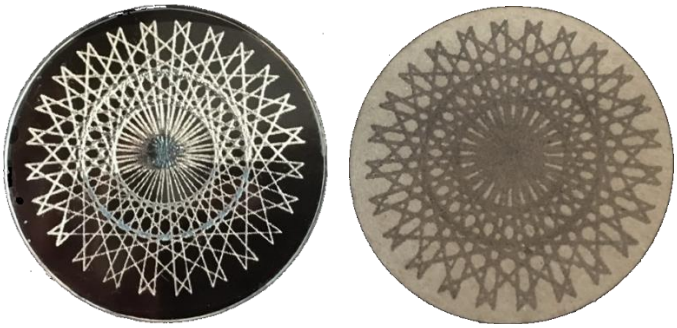
A Snowflake Pattern

Use a Star procedure to create a snowflake pattern. Draw a star; then turn the turtle a few degrees before drawing another star. Repeat until a pattern emerges.



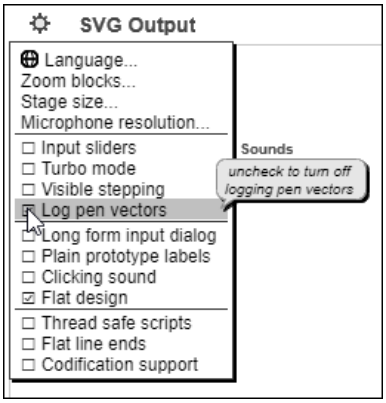
Fabricating Art with Patterns

Tools such as inkjet printers, laser cutters, and other fabrication tools can be used to translate digital designs into ornaments, jewelry, and sculptures in the same manner as digital artists.

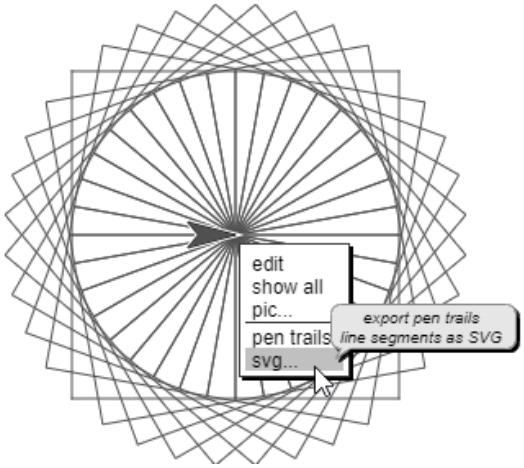


Print your design with an ink-jet printer. If you have access to a laser cutter or 3D printer, use the pattern to create a three-dimensional ornament.

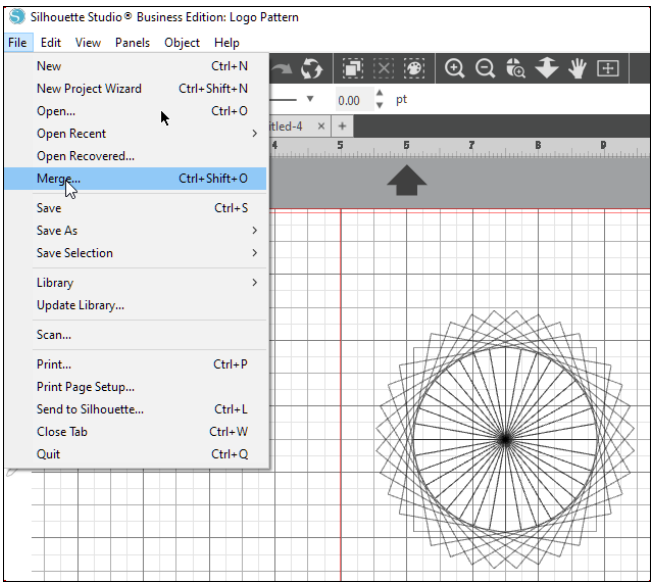
Graphics drawn with the turtle can be exported as Scalable Vector Graphics (SVG) files for higher resolution output. To access the *Scalable Vector Graphic* feature, turn on the *Log Pen Vectors* option in *Settings*.



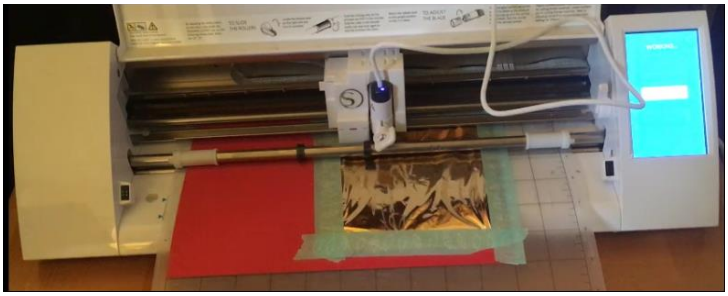
After a design has been drawn, place the mouse cursor on any part of the design and right-click to access the menu with the “SVG Export” option.



The SVG file can then be imported into other graphics programs such as Silhouette Studio. (Silhouette Studio Business Edition includes an option to import SVG files.)



Foil Quill is a third party option available for the Silhouette die cutter that can be used to emboss foil patterns onto materials such as card stock.



The result is an embossed foil pattern obtained by using an SVG vector pattern generated by *Snap!*.