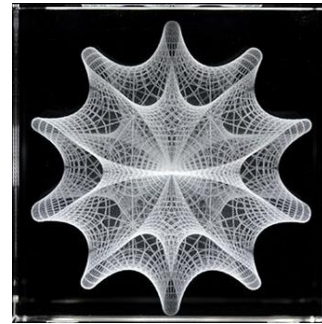


Creating Artistic Patterns through Computer Graphics

Glen Bull, Jo Watts, and Alexis Kellam

Many artists now use computer graphics to create artistic patterns. Bathsheba Grossman is representative of a new generation of artists who use computers to extend the type of art that they create. The design below was created by Bathsheba Grossman (<https://bathsheba.com/crystal/calabiyau/>) using a computer-generated pattern.

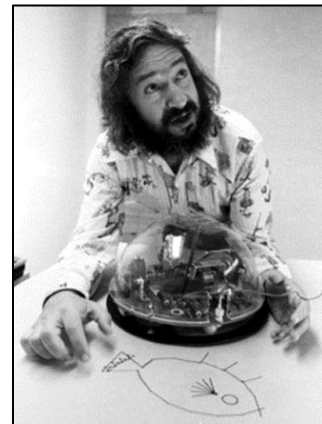


A laser-etched sculpture created by Bathsheba Grossman using an algorithm. (Photo used with permission of the artist.)

Topic 1.1 Algorithmic Thinking – Finding a Pattern

The first robotic turtles were developed at M.I.T. These robots consisted of Plexiglas hemispheres that rolled about the floor. They were called “turtles” because the Plexiglas hemispheres resembled the shell of a turtle. A pen in the belly of the turtle could be lowered to draw patterns on paper placed on the floor.

Robotic turtles are still used to draw patterns on paper today. However, a turtle on the screen of the computer can be used to create a design more quickly. Screen turtles typically consist of a triangle on the screen of the computer. The **Pen Up** and **Pen Down** code blocks are used to raise and lower the pen in the belly of the screen turtle.



A robot turtle developed at MIT. (Photo by Cynthia Solomon.)



The explorations that follow begin with simple patterns. These simple patterns are then used to create more complex patterns.

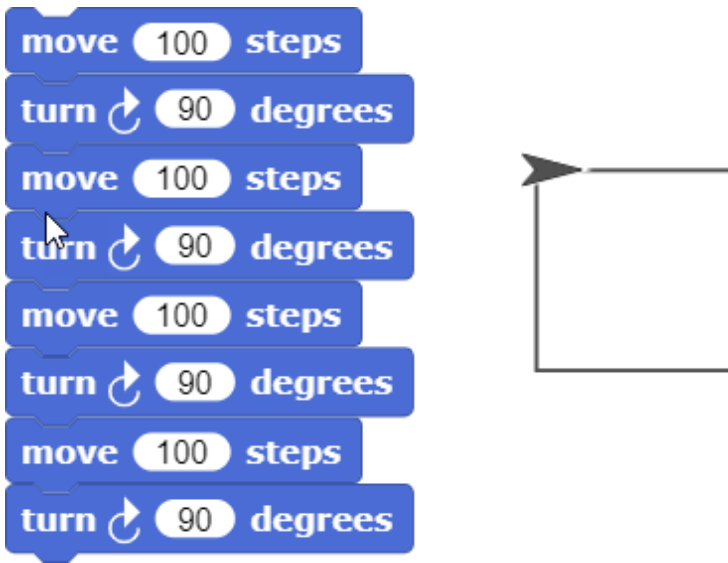
A square, for example, is one of the most basic patterns that can be created. A square is created by drawing a line and then turning 90 degrees. For example, if the turtle begins in the position shown below (pointing right):



then executing the code blocks **Move 100 Steps** followed by **Turn 90 Degrees** causes the turtle to draw a line and then turn right.



Repeating the **Move** and **Turn** code blocks four times causes the turtle to draw a square.



However, repeating the pattern four times achieves the same result more efficiently. Computers are efficient at repeating a pattern many times. However, the human coder must recognize the pattern in order to take advantage of the computer's capabilities in this way.



Computing Concepts

The example above illustrates the process of repeating a series of steps to draw a square. This can be done most efficiently by identifying the pattern involved in drawing the square. The pattern in this case consists of a *Move* command followed by a *Turn* command: **Move 100; Turn 90 Degrees**.

This pattern provides a recipe for drawing a square. In computer science, the term *algorithm* is used to describe a pattern that can be used as a recipe for creating the steps of a program.

The process of repeating the steps of the pattern several times is known as iteration. In the case of the square, the steps of the pattern (Move Forward and Turn) are repeated four times (i.e., four iterations).

An algorithm can be written in multiple ways and still accomplish the same task. A square can be drawn in eight steps; the same pattern can be drawn more efficiently by using a Repeat code block in combination with one Move code block and one Turn code block

Exploration 1.1 Algorithmic Thinking – Finding a Pattern

Identify the pattern used to create other shapes such as triangles (three-sided figures) and polygons (five sided figures). Assemble and run code blocks that draw shapes such as triangles and polygons.

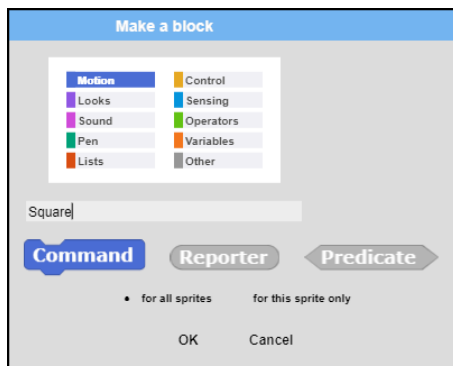
Topic 1.2 Abstraction – Making a Custom Code Block

The limits of human memory place limits on creation of computer code. A human can only retain a limited number of code blocks in memory at one time. Experienced coders circumvent these limits through abstraction. Once code has been developed and tested, it can be abstracted through creation of a custom code block.

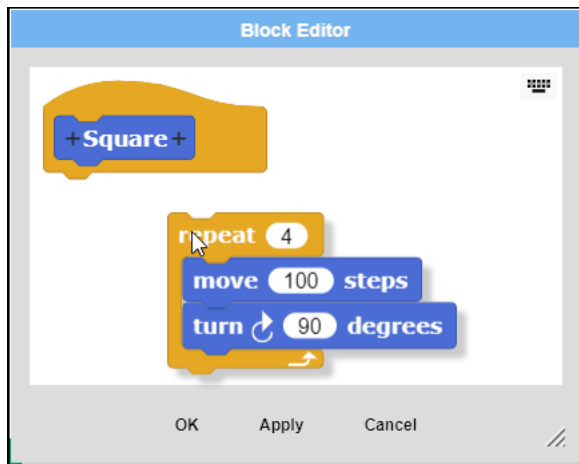
Clicking the *plus* sign near the top of the command palette invokes the option to make a custom block.



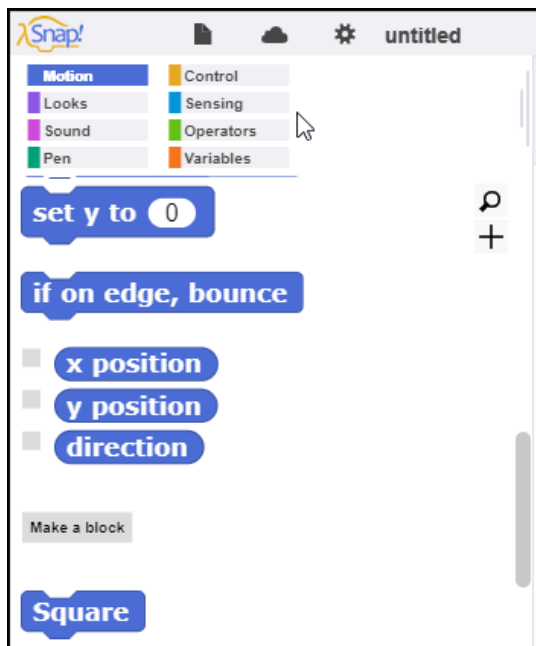
A dialog box appears when the *Make a Block* option is invoked. The dialog box includes a space to enter the name of a new custom block. (Note: in most instances the default selection of “for all sprites” is used.)



When the entered name is confirmed by clicking “OK,” a block editor appears. The working code previously developed and tested can then be dragged into the block editor and snapped into place beneath the header block at the top of the block editor.



Once this choice is confirmed by clicking “OK,” the newly defined custom block appears at the bottom of the other commands in the command palette.



The new custom block can then be dragged into the script area and used in the same way as any other code block.

Creating a custom block reduces the burden on the coder’s memory. Instead of remembering the correct order of an entire sequence of code blocks, the coder now only needs to remember the name of a single custom block. For this reason, it is important to create a meaningful name for the custom block so that its purpose is clear.

The introduction to a famous computer science textbook, the *Structure and Interpretation of Computer Programs*, asserts in the preface that “Programs must be written for people to read, and only incidentally for machines to execute” (Abelson & Sussman). Elegance in programming is a balance between efficiency (writing code in as few lines as possible) and readability (creating custom code blocks with meaningful names). Readability ensures that code can readily be maintained and re-used in other programs.

Computing Concepts

A procedure is a group of program instructions that have been given a name. In the example above, the instructions for drawing a square are assigned the name “*Square*”.

In Snap! a custom code block is created by assigning a name to a group of instructions. Custom code blocks can be used in Snap! in the same way that built-in code blocks are used.

The term *abstraction* refers to the process of naming a concept without explaining the details of the concept each time. The process of assigning a name to a group of instructions to create a procedure is one form of *abstraction*. Once a code is tested and encapsulated in the form of a procedure, the programmer no longer needs to remember the specific details of the procedure. This enables programmers to focus on other program details and create much more complicated programs than otherwise would be possible.

In the example above, a custom code block named Square replaced a longer sequence of instructions. Replacing an entire sequence of instructions with a named procedure yields several benefits.

1. It makes the top level of the program more concise and easier to read.
2. If meaningful names are used, it makes the program easier to follow.
3. The cognitive load on the programmer is reduced by replacing a longer sequence of code with a procedure name. This enables the programmer to devote attention to other details of the program.

Exploration 1.2 Abstraction – Making a Custom Code Block

Create custom code blocks for other shapes such as *Triangle* and *Pentagon*.

Topic 1. 3. Generalization – Creating Inputs for Custom Code Blocks

One of the goals of effective programming is creation of re-usable code. Inputs to custom code blocks make these code blocks more generalizable.

The previously created **Square** code block made a square that was 100 turtle steps on each side. In other circumstances, a square that was 50 steps on a side or 200 steps on side might be needed. One approach to addressing this need consists of creating a new custom block – such as **Small Square** and **Big Square** – for each new application.

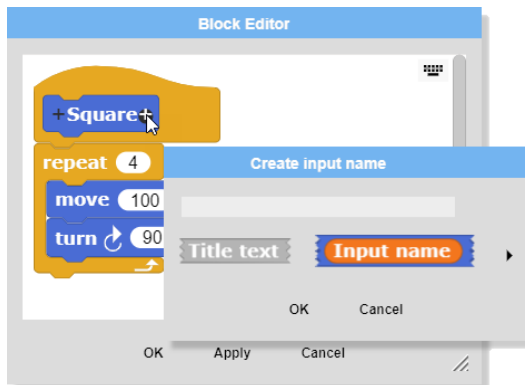


However, a more effective approach would be to incorporate an input into the procedure that made it possible to use a single code block to draw a square of any size.

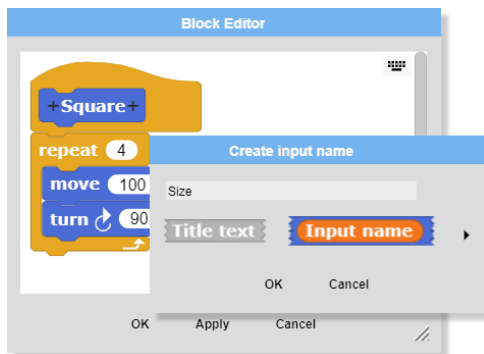
Right-clicking the **Square** code block reveals a drop-down menu that offers the option of editing the code block.



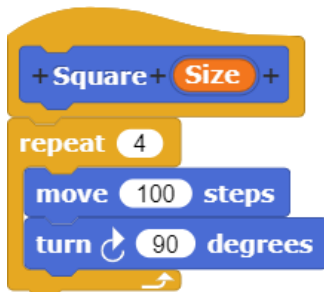
Selection of the *Edit* option accesses the *Block Editor*. If the plus sign (“+”) beside the name *Square* is clicked, another dialog box appears that offers the option of entering an input name.



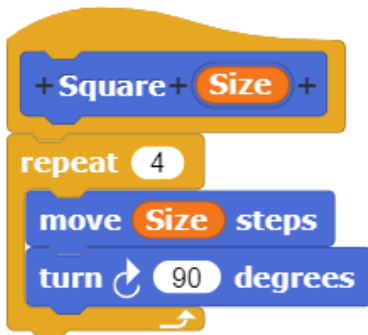
Once an input name is selected (*Size* in this instance), the choice can be confirmed by clicking “OK.”



Once this choice is confirmed, the input name (*Size*) will appear to the right of the block name (*Square*).



This orange oval representing the input *Size* can then be dragged into the input slot of the **Move** code block.



Once this revision of the custom code block is confirmed, an input slot will appear to the right of the name of the code block.



The input slot makes it possible to use the same code block to create squares of three different sizes, rather than creating three different code blocks for the same purpose.



Use of inputs to create more generalizable procedures greatly increases the efficiency of coding.

Computing Concepts

Procedures are implemented in the form of custom code blocks in Snap!. In Snap!, a code block can be executed (i.e., called) by clicking it.

Procedures can have inputs. For example, an input to the **Square** procedure can be used to specify the size of the square. In computer science, the term *parameter* can be used to describe an input such as size that can be varied in this manner.

Use of an input such as size allows the procedure to be generalized. This enables the procedure to be re-used to draw squares with a wide range of different sizes.

In the example above, the numbers “50”, “100”, and “200” are used as inputs to the procedure. The *size* parameter allows the procedure to be generalized, enabling **Square** to be used to draw squares of different sizes.

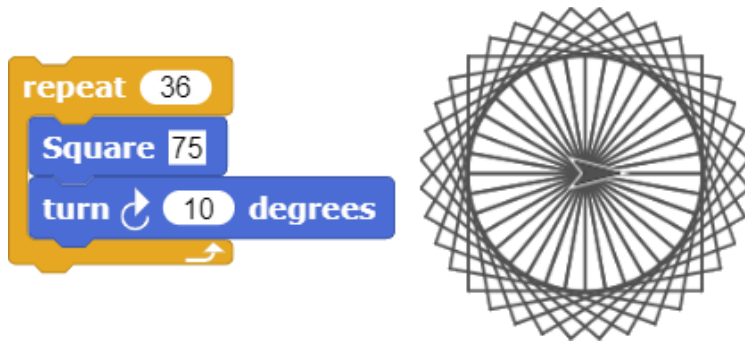
Exploration 1.3 Generalization – Creating Inputs for Custom Code Blocks

Create inputs for other shapes such as the Triangle and Pentagon code blocks.

Topic 1.4 Building Complex Code Block by Block

Effective coding begins with construction of basic building blocks that are used to create more complex procedures.

For example, if a series of squares is drawn, turning a bit after each square is drawn, the following pattern results.



In this manner, the Square code block becomes a building block for construction of another custom block, **Spin Square**.



Surprisingly complex patterns can be generated through construction of a library of custom code blocks in this manner.

Computing Concepts

An effective coding strategy breaks a larger problem into a series of smaller sub-problems. Procedures are created to solve each of the smaller sub-problems. The larger problem is then solved by combining the procedures that were developed to solve each of the sub-problems.

In the example above, in order to create the **Spin Square** procedure, the larger problem was separated into two smaller problems: (1) drawing the square and (2) turning the square. The **Spin Square** code block is based on the procedure **Square**. Development of the **Square** procedure makes the code easier to understand when it is incorporated into a more complex program in the future.

Exploration 1.4 – Building Complex Code Block by Block

Incorporate the **Triangle** and **Pentagon** code blocks into other code blocks to create complex patterns.

Topic 1.5 Using Loops to Vary a Parameter

The **Repeat** block represents one type of loop in which the same action is repeated over and over again. A **For** block is another type of loop. One advantage of a **For** loop is that the variable i changes each time that the loop is executed.

In the example below, the variable i has a value of “1” the first time the loop is executed. The value “1” is multiplied by 10 in the input to the **Square** code block. Consequently, a square with a size of 10 steps on a side is drawn the first time the loop is executed.



The second time that the loop is executed, a square is drawn that is 20 steps on a side. Since $i = 2$ during the second execution of the loop, the input to the **Square** block becomes 2×10 (i.e., 20).

The last time that the loop is executed, a square that is 100 steps on a side is drawn. In this manner, a series of nested blocks is drawn using just two code blocks, the **For** loop combined with the **Square** code block.

The ability to vary the input of a code block each time that the loop is executed can therefore yield some powerful effects in an efficient way. Almost every program of any complexity will make use of one or more loops in this manner.

Computing Concepts

A variable is a name that can be assigned a value. For example, the name *Size* might be assigned a value of "40". In Snap!, the **Set** command can be used to assign a value to a variable.

The *Operators* palette in Snap! provides mathematical and logical operators that can be used to evaluate an expression such as " $2 + 2$ " (which equals "4") or " $\text{Size} = 40$ " (which returns a value of "True" if the variable *Size* has been set to "40").

In the example above, the multiplication operator is used to multiply the variable "*i*" by "10" to draw a series of squares with increasingly larger sizes.

The term *incrementing* is used to describe the process of changing the value of a variable during each iteration of a loop.

Exploration 1.5 – Using Loops to Vary a Parameter

Explore creation of nested shapes such as triangles and pentagons through use of loops.

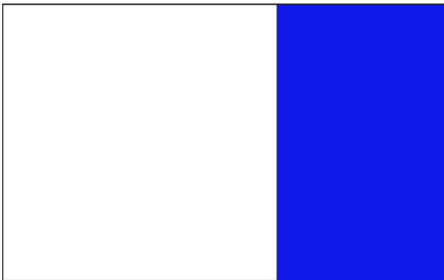
Topic 1.6 Computational Thinking

Computational Thinking can be described as the process of formulating the solution to a problem in a manner that can be solved by a computer. The most efficient way solving a problem by computer can often be different from the strategy that might be used to solve a problem by hand.

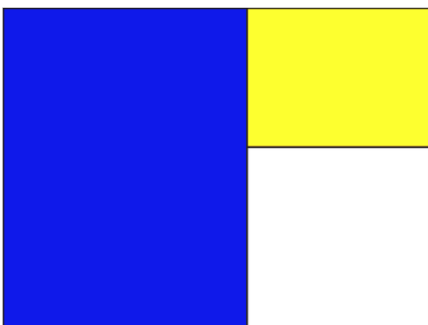
In the following example, the process of constructing a golden ratio by hand is contrasted with the process of constructing a golden ratio with a computer. The golden ratio has been used in art and design for centuries. Leonardo da Vinci used golden rectangles in many of his paintings, including the *Last Supper*.



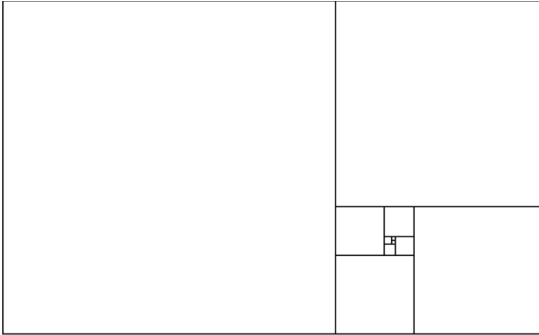
A square drawn within a golden rectangle produces another smaller golden rectangle. In the illustration below, the larger golden rectangle has been divided into two parts by drawing a square inside it. This results in a second golden rectangle shown in blue.



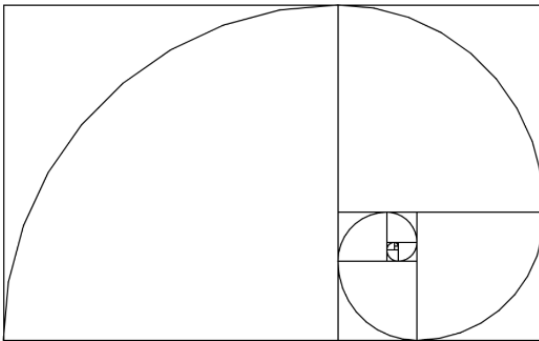
If a square (shown in white) is drawn inside the second golden rectangle, a third golden rectangle (shown in yellow) results.



This process can be repeated indefinitely, limited only by the resolution of the medium on which the rectangles are drawn.



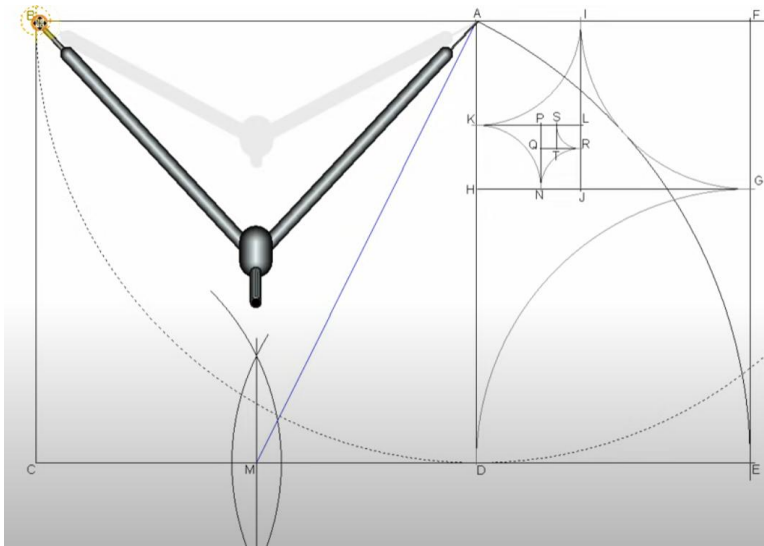
If a quarter-circle is drawn inside each of the squares created in this way, a golden spiral results.



There are many examples of golden spirals in nature ranging from nautilus shells to spiral galaxies.



In traditional geometry, golden rectangles are constructed using a compass and a ruler. A series of arcs are created with a compass that provide benchmarks for aligning the ruler.



The process begins by constructing an initial golden rectangle, subdividing it to create a square, and repeating the process. When presented with the task of constructing a series of golden rectangles, a computing novice often will replicate the hand-drawn sequence:

draw a golden rectangle,
subdivide with a square, and
repeat by drawing another square within the next subdivided rectangle.

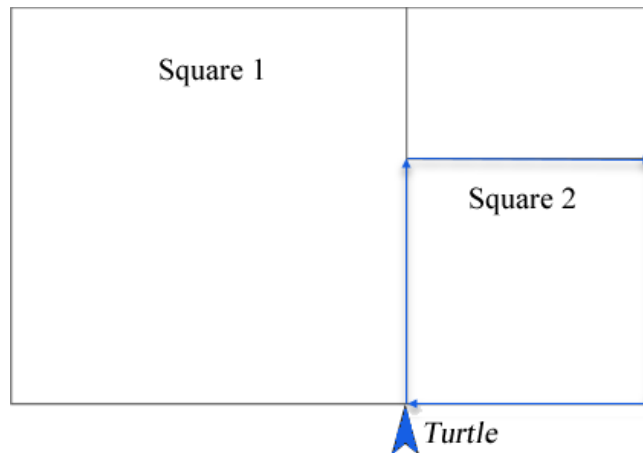
The ratio of the shorter and longer sides of a golden rectangle is approximately 1.618, represented by the Greek letter ϕ (phi) in geometry. Construction of the initial golden rectangle can therefore be accomplished by drawing a rectangle whose width is the height times the constant *phi*:



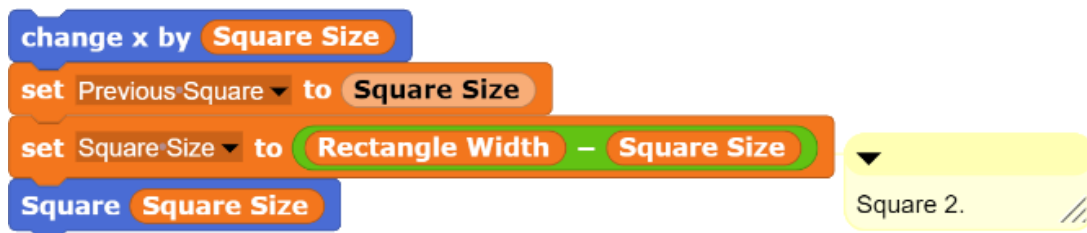
Once the initial rectangle has been constructed, a square with sides that match the height of the rectangle can be drawn within the rectangle.



In the illustration, the turtle's horizontal (X) position has been shifted by the width of the initial square. A second square is then drawn within the second rectangle.

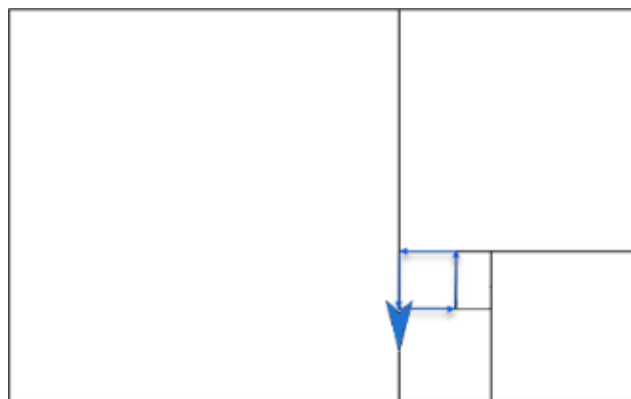


The code shown below shifts the turtle to the correct horizontal position and draws the second square.

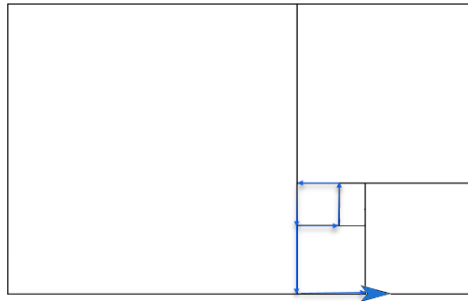


This process can then be repeated to create additional rectangles subdivided by squares. However, this strategy of drawing rectangles subdivided into squares was guided by methods possible with a compass and ruler.

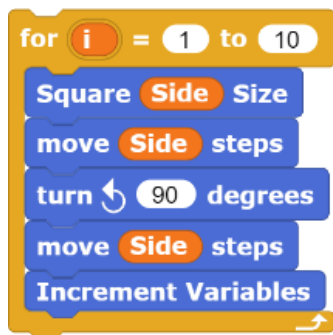
A digital computer makes other strategies possible. In the example below, the figure is constructed by drawing a series of squares beginning from the inside out. The innermost square is drawn first.



The turtle is then positioned to draw a second square. The process is repeated as a series of squares is drawn, from the innermost square to the largest square.



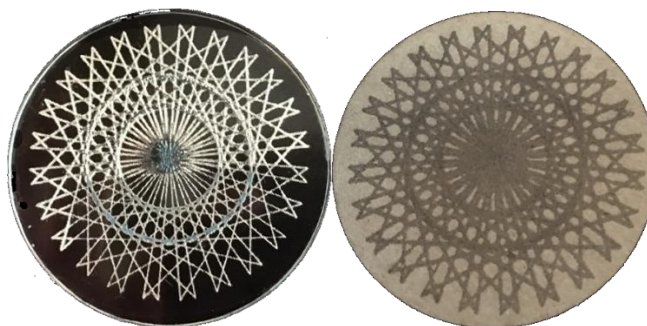
In the code below, the initial command **Square Size** draws a square. The next three code blocks (**Move, Turn, Move**) move the turtle so that it is positioned to draw the next square. After variables such as the size of the next square are updated, the process is repeated.



This strategy is more compact and easier to understand than the method used in the initial example. In this instance, the method of constructing a series of golden rectangles by hand is not necessarily the best strategy for accomplishing the same task with a computer. Hence the term *computational thinking* – the process of formulating the most effective approach to completing a task with a computer.

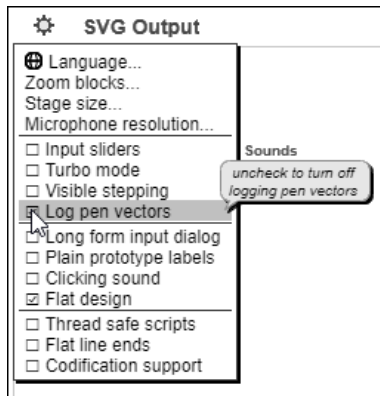
Topic 1.7 Fabricating Art with Patterns

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.

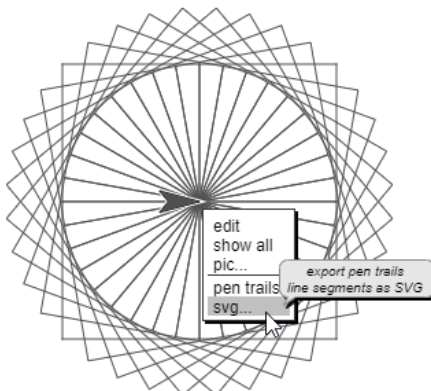


Print your design with an ink-jet printer. If you have access to a laser cutter or 3D printer, use the pattern to create a three-dimensional ornament.

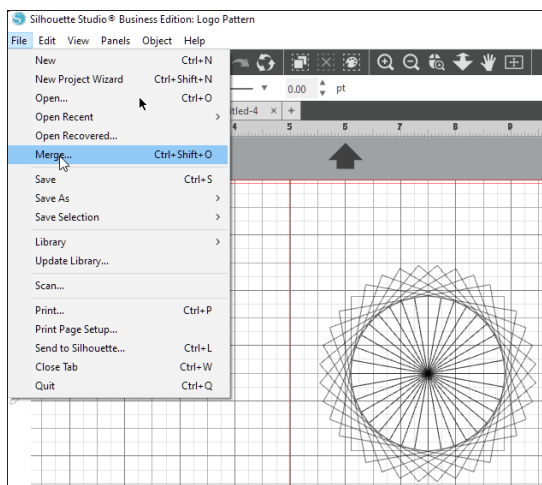
Graphics drawn with the turtle can be exported as Scalable Vector Graphics (SVG) files for higher resolution output. To access the *Scalable Vector Graphic* feature, turn on the *Log Pen Vectors* option in *Settings*.



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



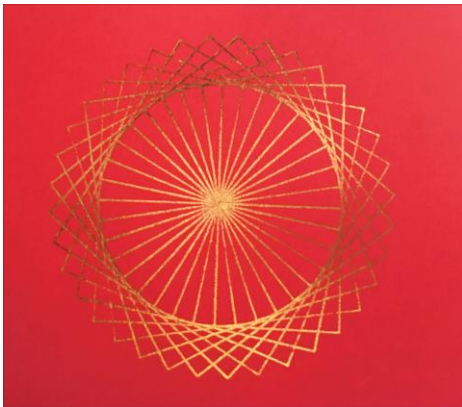
The SVG file can then be imported into other graphics programs such as Silhouette Studio. (Silhouette Studio Business Edition includes an option to import SVG files.)



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!.



Exploration 1.7 – Fabricating Art with Patterns

Design a card with your own unique artistic design.

Appendix A

Code Blocks introduced

A. Motion Palette

Move and Turn

The **Move** code block is used to move the turtle forward. (If a negative number is used as an input, the turtle moves backward.)



The **Turn** code block is used to rotate the heading of the turtle.

B. Pen Palette

Pen Up and Pen Down

The **Pen Up** and **Pen Down** code blocks are used to raise and lower the pen in the belly of the screen turtle.



C. Control Palette

Repeat

The **Repeat** code block consists of a “C-shaped” block. Other code blocks placed within the **Repeat** command are repeated the specified number of times.



For Loop

The **For** code block consists of a “C-shaped” block that can be used to repeat a sequence of actions while changing the value of a variable (such as “*i*” in the example below) during each iteration of the loop.



Appendix B

New APCS Principles

This appendix lists *Advanced Placement Computing Standards* (APCS) associated with each topic.

Topic 1.1 Algorithmic Thinking – Finding a Pattern

The section illustrates the process of repeating a series of steps to draw a square.

Algorithms

The term algorithm refers to a set of instructions that accomplish a task. A computer algorithm refers to a set of instructions that can be implemented by a computer.

CS Principle AAP-2.A Developing Algorithms

An algorithm that uses sequencing can be expressed without using a computer language. For example, before approaching the code, a student should understand that to make a square, they will need to write a procedure that creates a single side and a right angle and then repeat that procedure four times. After understanding that, the concepts can be transferred to assembling code blocks.

CS Principle AAP-2.B Programming Algorithms

Represent a step-by-step algorithmic process using sequential code statements.

CS Principle AAP-2.J Iteration

A computer often needs to repeat the same series of steps several times. The term iteration refers to a repeated sequence of steps. In the example in the preceding section, a series of steps is repeated to create an artistic pattern.

Application

In Topic 1.1, an algorithm for drawing a square is developed. Development of this algorithm depends on recognition that the **Move** code block and the **Turn** code block can be combined and repeated to form a square.

Topic 1.2 Abstraction – Making a Custom Code Block

A procedure is a group of program instructions that have been given a name. In the example above, the instructions for drawing a square are assigned the name “*Square*”.

CS Principle AAP-3.A Procedures

A procedure is a named group of programming instructions. In Snap! a custom code block is created by assigning a name to a group of instructions. Custom code blocks can be used in Snap! in the same way that built-in code blocks (known as primitives) are used.

CS Principle AAP-3.B Procedural Abstraction

Assigning a name to a group of instructions to create a procedure is one form of abstraction. Once a code sequence is tested and encapsulated in the form of a procedure, the programmer no longer needs to remember the specific details of the procedure, enabling the programmer to focus on other program details.

Application

In the example in this section, a custom code block named **Square** replaced a longer sequence of instructions. Replacing an entire sequence of instructions with a named procedure yields several benefits. First, it makes the top level of the program more concise and easier to read. Second, if meaningful names are used, it makes the program easier to follow. Finally, it reduces the cognitive load on the programmer by replacing a longer sequence of code with a procedure name, enabling the programmer to devote their attention to other details of the program.

Topic 1.3. Generalization – Creating Inputs for Custom Code Blocks

CS Principle AAP-3.A Parameters for Procedures

Procedures frequently have inputs. For example, an input to the **Square** procedure can be used to specify the size of the square. In computer science, the term *parameter* can be used to describe an input such as size that can be varied in this manner.

CS Principle AAP-3.C Developing Procedures

Developing abstractions helps manage the complexity of a procedure. For example, use of an input such as size allows the procedure to be generalized. This enables the procedure to be re-used to draw squares with a wide range of different sizes.

Application

In Topic 1.3, the numbers “50”, “100”, and “200” are used as inputs to a procedure. The *size* parameter allows the procedure to be generalized, enabling **Square** to be used to draw squares of different sizes.

Topic 1.4 Building Complex Code Block by Block

CS Principle AAP-3.D Selecting Appropriate Procedures to Use in Programs

An effective coding strategy breaks a larger problem into a series of smaller sub-problems. Procedures are created to solve each of the smaller sub-problems. The larger problem is then solved by combining the procedures that were developed to solve each of the sub-problems.

Procedural abstraction allows a solution to a large problem to be based on the solutions of smaller sub-problems. This is accomplished by creating procedures to solve each of the sub-problems.

Application

In Topic 1.4, the larger problem was separated into two smaller problems: (1) drawing the square and (2) turning the square to create the **Spin Square** procedure. The **Spin Square** code block is based on the procedure **Square**. Development of the **Square** procedure makes the code easier to understand when it is incorporated into a more complex program in the future.

Topic 1.5 Using Loops to Vary a Parameter

CS Principle AAP-1.A Variables

A variable is an abstraction inside a program that can hold a value. Each variable has associated data storage that represents one value at a time, but that value can be a list or other collection that in turn contains multiple values.

Using meaningful variable names helps with the readability of program code and understanding of what values are represented by the variables.

Application

Variables can be used to change the length of a shape’s sides or the angle of its corners.

CS Principle AAP-2.C Mathematical Expressions

Mathematical expressions, sometimes called operators, enable the computer to perform mathematical calculations. Expressions are evaluated to produce a single value.

Expressions are part of most programming languages and include addition, subtraction, multiplication, division, and modulus operators.

Application

By modifying a variable with a mathematical expression, the shape drawn can be changed with each iteration.

CS Principle AAP-2.K Iterative Statements

Iteration is a repeating portion of an algorithm that repeats a certain number of times or until a given condition is met.

Application

Iteration can be used to repeat statements that will draw a single shape. It can also be used to repeat procedures that will draw shapes differently with each iteration.

CS Principle CRD-2.B Programs

A program is a collection of statements that performs a specific task when run by a computer. Statements with a program are called code segments. Code segments often incorporate procedures to simplify the code and make it easier for humans to read and write. A program can be described broadly by what it does, or in more detail by both what the program does and how its statements accomplish this function.

Application

In this section, a program is used to draw a sequence of inset squares. The program consists of an iterative loop, a variable, a procedure, and a multiplication operator. The procedure draws a square using the variable from the loop and a multiplication operator to set the square's size. Each time the loop repeats, the variable increases to the next integer and the procedure draws a square. This process continues until ten squares have been drawn.