www.logofoundation.org

Reflections on Hour of Code

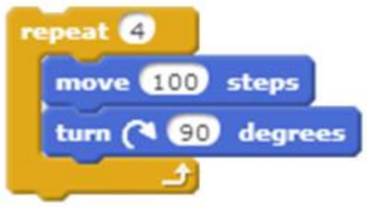
By Michel Tempel

More than 100 million people have tried an Hour of Code. What does this introduction to computer programming, or "coding" look like?

I went through two of the Hour of Code tutorials: *Code with Anna and Elsa* from Code Studio, with a *Frozen* theme, and the Scratch tutorials.

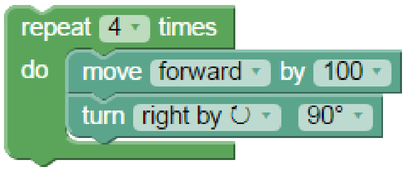
Both use blocks, a type of visual programming in which programs are represented by a collection of icons that are like jigsaw puzzle pieces. They snap together to form instructions. The blocks fit together only in ways that are syntactically correct. This avoids the chronic problem of syntax errors and the flood of error messages that confront beginners using text-based programming languages[1].

The colors of the blocks give the programmer information about what they do. For example, in Scratch the blocks that move sprites are blue; those that control program flow, such as REPEAT, are mustard.



```
repeat 4
  move 100 steps
  turn ↻ 90 degrees
```

The *Frozen* tutorial uses Blockly, which has a similar structure.



```
repeat 4 times
do  move forward by 100
    turn right by ↻ 90°
```

[1] See Blocks Programming for an overview of this type of programming environment.

The activities in the *Frozen* tutorial require the user to solve a series of puzzles based on Turtle Geometry. The Scratch tutorial offers three options: create an animation of your name, make a holiday greeting card, or build a pong game.

Both tutorials offer a pathway for the user to follow in order to have a beginning coding experience. But the pedagogical approaches are different. The *Frozen* tutorial starts with a very limited number of available blocks, with additional blocks added as the tutorial progresses through a series of 20 puzzles. At each step you get the puzzle right, are congratulated, and move on to the next puzzle, or are told to try again.

The Scratch tutorials are on the "Tips" tab that is built into the Scratch programming environment. The tutorials show how to use a specific few blocks, but all Scratch blocks are available to the user all the time. Nothing is hidden. If you follow the guidelines, you will get a result that is similar to what is described in the video at the beginning of each tutorial. There is no right or wrong outcome. Even within the narrow scope of the tutorials, there is room for variety and individual choice. In the case of the "Animate Your Name" project, some individualization is inherent in the activity.

One can visualize a Scratch tutorial like this:

There is a path to follow, but you can also see what's off the track and it is possible to wander away in different directions.

The *Frozen* tutorial looks more like this:

There is a path to follow, and you can't stray from it.

The *Frozen* tutorial is restrictive. Possibilities are limited and students are not likely to get lost. The Scratch tutorials are more open-ended. There are more possibilities for creativity. Although some students may get distracted by the larger programming environment, which is visible while doing the tutorial, the tutorial provides guidance that focuses the activity.

This is somewhat like the proverbial problem of herding cats. There are two approaches to keeping a group of cats together. One method is to build a fence around the area you want them to stay in. Another approach is to forget about the fence and instead, put a bowl of milk in the center of that area.

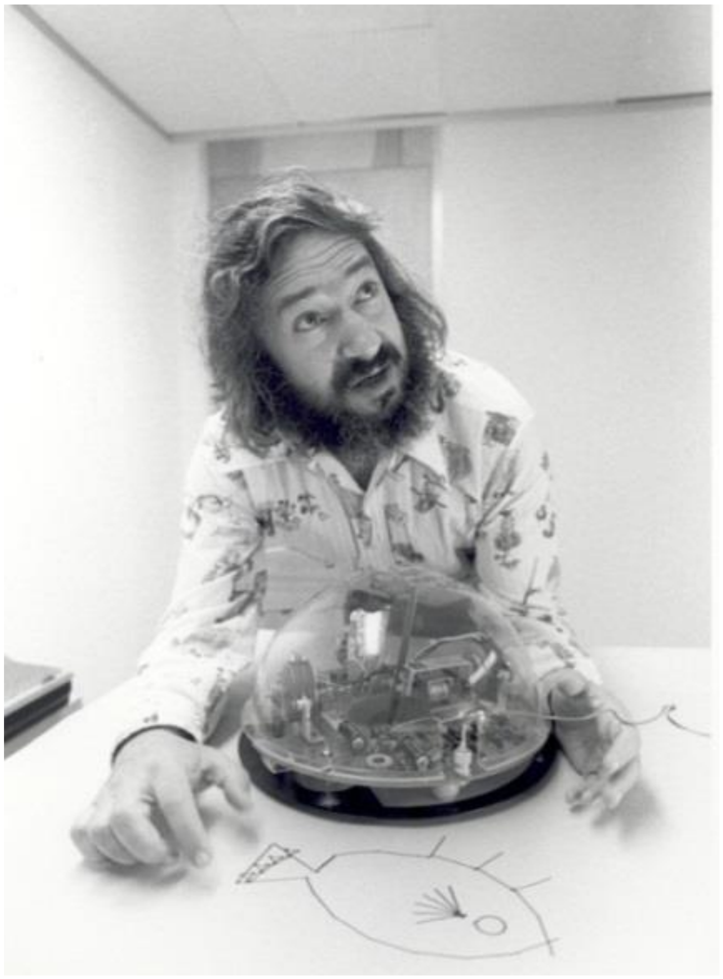
Some of the restrictions in the *Frozen* tutorial serve to guide the user toward the desired result. For example, the TURN block has a pull-down menu with a limited list of choices for the number of degrees to turn, e.g., 45, 60, 90, 120. The choices change depending on the particular puzzle. Any of these turns will produce familiar geometric shapes when used along with the MOVE and REPEAT blocks.

If one were doing this activity in Scratch, any number could be an input to TURN. But guidance is offered by having the default value be 15. This number is arguably the most useful single value you can use with the TURN block because it can be repeated some number of times to get some very useful turns: 30, 45, 60, 90, and 120 degrees.

Similarly, the POINT IN DIRECTION block can be used to aim a sprite in any direction, but the often used headings 0, 90, 180, and -90 (or 270) are built in, also labeled up, right, down, and left.

Turtle Geometry

The *Frozen* tutorial is based on Turtle Geometry. Before 1980, when Logo programming environments that included the Turtle became available, it was virtually unknown in schools. Now it is an established part of the educational landscape. The arrival of Logo and the Turtle coincided with the 1981 publication of *Mindstorms*[2], by Seymour Papert, in which he describes Turtle Geometry and how it was designed as a "mathland" where students could learn mathematics informally, much as children learn to speak their native language without formal instruction. Papert contrasts it with the more formal Euclidian Geometry that is common in school math. Turtle Geometry is "body-syntonic" – the turtle moves around the way you do. So if you want to know how to get the turtle to move in a particular pattern, get up and walk around, playing turtle and describing what you are doing.



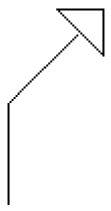
Since the turtle is a computational creature, you teach it how to move with a computer program. Logo has been widely popular as the language to talk to turtles, although other languages, e.g., Python are used as well.

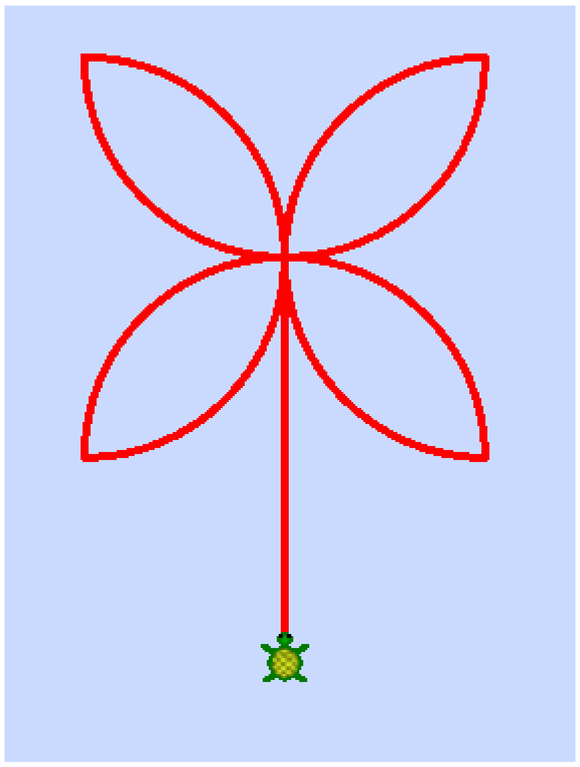
The early turtles were robots that lived on the floor and were connected to computers with cables. The commands FORWARD and BACK would cause a turtle to move in a straight line. RIGHT and LEFT would cause it to rotate clockwise or counterclockwise without changing position.
The turtle also had a pen in its middle, which could be lowered to the floor with the PENDOWN command, and raised with PENUP. With the pen down, it would draw as it moved.

With those very few commands, the turtle could move anywhere and create any line drawing. In this photo, the turtle has drawn a fish.

[2] Papert, Seymour *Mindstorms*, Basic Books, 1981

Other species of turtles inhabited the computer screen, some represented as triangles to show which way they were pointing. Others were graphic images of a turtle, as seen from above.

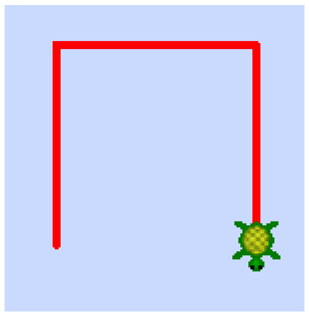




That last point is important. Looking at a turtle on the screen is like having an aerial view of the creature and its surroundings. Imagine looking down on the turtle from above. Then imagine you are the turtle as you give instructions on how to move around.

Keep this in mind when presented with the task of drawing a square. Let's say you've gotten this far.

```
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
```

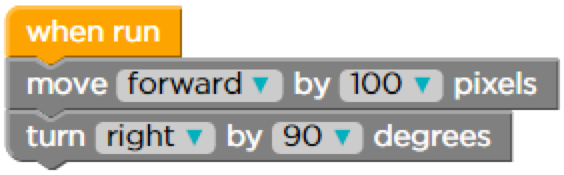


What's the next instruction?

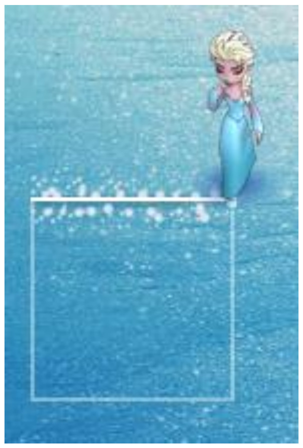
(*Frozen* and Scratch use MOVE, which can take a positive number as input for a forward move, and a negative number to move backwards.)

This context for Turtle Geometry is worth remembering when looking at The *Frozen* tutorial.

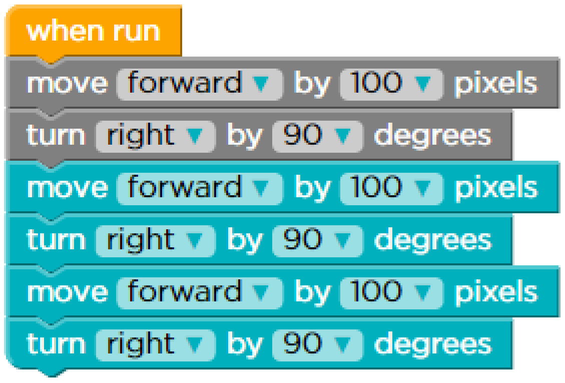
In Puzzle 3 the task is to have Elsa skate in a square. Here she is after we run this stack of commands:



```
when run
move forward ▼ by 100 ▼ pixels
turn right ▼ by 90 ▼ degrees
```

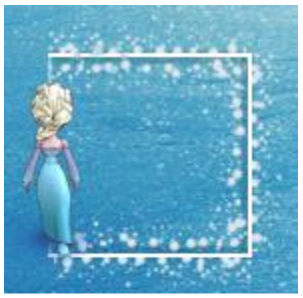


Now after these commands:



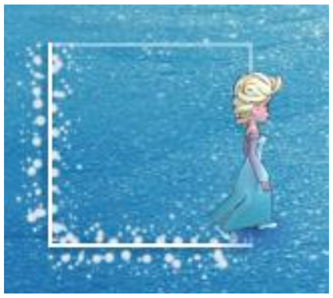
```
when run
move forward ▼ by 100 ▼ pixels
turn right ▼ by 90 ▼ degrees
move forward ▼ by 100 ▼ pixels
turn right ▼ by 90 ▼ degrees
move forward ▼ by 100 ▼ pixels
turn right ▼ by 90 ▼ degrees
```

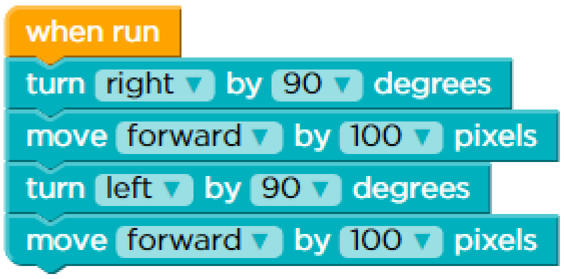
we see this:



The emerging square is being viewed directly from above, but Elsa is viewed as if on stage, not from above. Our point of view of Elsa is a bit elevated - we seem to have front row balcony seats - but not from above. There are two different perspectives mixed together. The turtle metaphor is confused.

This tutorial also represents a pedagogical departure from the discussion of Turtle geometry in *Mindstorms*. You get a puzzle right or wrong. And sometimes the tutorial is unhappy even when the task is met. For example, Puzzle 2 says "Now let's see if we can create two lines that are at a 90 degree angle to each other." Here's what I drew and the stack of blocks I used to draw it:



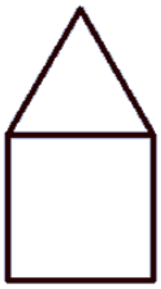


```
when run
turn right ▼ by 90 ▼ degrees
move forward ▼ by 100 ▼ pixels
turn left ▼ by 90 ▼ degrees
move forward ▼ by 100 ▼ pixels
```

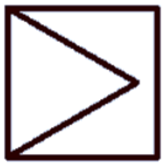
This got the message: "You are using all of the necessary types of blocks but not in the right way." Now I know that the intention of the puzzle maker was for me to first draw the line from left to right, turn right 90˚ and then draw the line from top to bottom. But what I did fulfilled the requirements of the puzzle as stated. And I'm not the only kid in the class who will deliberately try to trip up the app, which is arguably a more challenging and creative approach than following the tutorial as it is intended.

In *Mindstorms* Papert describes how children developed projects in a Logo-Turtle Geometry environment. Children might start with a pencil and paper drawing and then see how to teach the turtle to draw something similar. Or, they might come up with the instructions for some shapes and see how they could be assembled into something more complex.

As an example, Papert describes how a child tried to put together a square and an equilateral triangle to make a house that would look something like this:



Quite far into the process of developing this project she had the turtle draw this:



One could say that this is wrong, but in fact it is mostly right. The square and triangle components are fine and they are connected. What is left to do is make an adjustment that repositions the triangle on top of the square. The program just needs to be debugged, and there are several different ways in which that can be done to get the desired result.

The concept of debugging originated and grew in the computer culture, and is an excellent perspective for teaching and learning more generally. It is often lacking in schools, pushed aside by a test-driven agenda that emphasizes getting it right. The solve-a-puzzle approach of the Frozen tutorial follows the right/wrong paradigm. It encourages guessing to solve a problem as much as debugging.

What's Next?
Each Scratch tutorial ends with suggestions about how to extend the project and also to look at the other tutorials under the Tips tab. They also suggest that you share your project in a Studio set up for that tutorial. Scratch Studios are collections of projects around various topics or themes. For example, the many Animate Your Name projects that Scratchers have created are collected in this Scratch Studio. When you share your project there you also see the scores of other name-animation projects that have been posted. A key aspect of the Scratch environment is this sharing; learning from what other people do, remixing projects, and communicating with other users.

At the end of the *Frozen* tutorial "You've officially become a master artist!" and you are invited to "Create a winter wonderland" using the blocks you've become familiar with while solving the previous 19 puzzles. There are no suggestions as to what you might do, no guidance or further support. This is like learning to swim by practicing strokes and kicking on dry land. After successfully completing a series of such exercises, your instructor congratulates you, officially declares you to be a swimmer, drops you in the deep end of the pool, and walks away. Good luck!

The divergent approaches of these two tutorials are certainly not unique to teaching and learning coding. They reflect long-standing differences in pedagogical approach in schools: skills-based vs. project-based. One may learn a series of skills and then apply them in a series of exercises, or maybe a project. Or, the project is the focus from the beginning, with skills, strategies, and practices learned as needed along the way.

An Hour of Code is just a beginning. This is recognized up front on the Hour of Code website with a prominent Beyond an Hour of Code link on the home page, which goes to a page of links to many other tutorials.

One option is to continue from the *Frozen* introduction to other Code Studio courses. These follow the same general format of the first tutorial – a series of puzzles.

With Scratch, in addition to the tutorials under Tips, there are tutorials posted by users. For example, if you search for "platform game tutorials" on the Scratch website you will find 180 projects. And there are Forums where you can get help. A user posted "i really need help. i want to create a platform game but i don't know how. please help me find a guide that you would recommend for basic platformers" and quickly got replies with pointers to tutorials created by other Scratch users.

And for teachers there is the ScratchEd website with a large body of resource material, forums, courses, and pointers to professional development and other Scratch events.

More About That Turtle
As we saw, the Code Studio tutorials make use of Turtle Geometry. There are excellent resources available for exploring this domain more deeply. Although the Hour of Code Scratch tutorials focus on animation and games, Turtle Geometry is also part of the Scratch environment using the blocks in the Motion, Pen, and Control tabs.

A more focused application is TurtleArt, a blocks-programming environment that is specifically designed for creating drawings with the turtle. It also includes an option for text programming in Logo, so it provides a pathway for a transition from blocks to text.

As we saw earlier, it is valuable to have experience with a robot floor turtle as a framework for experience with screen turtles. There are several good options available now including BeeBot, ProBot, and Finch robots.

And there are several hundred versions of Logo, most of which include Turtle Geometry. Look at the Logo Tree Project for the full list.