Variables, Motion, Programming Style

1. Introduction

We continue to study Scratch. Last time we studied statements, conditionals, numeric and Boolean values, and three kinds of loops. These ideas are essential to programming computers. There are just a few other essential ideas. The rest of programming computers involves combining those ideas into programs.

2. Variables: Places to Store Values

Computers, like the automated music machines and animated paintings, perform actions, but they also store data. Memory chips, hard disks, USB memory sticks, iPods, CDROMs, DVDs are all examples of where computers store data. How does a program use storage to get its work done? We begin with by revisiting an example from last time.

Do you recall the Hello5 program that introduced random numbers? It looks like:

This script picks a random number in the range 1-10 and, based on that value, plays a sound or not. What if we wanted the script to "Say" the value it picks so we know what that number is? In order to "Say" the number and also to test if the number is less than 6 requires that the script remember what number it picked. Somehow, the script has to write down the value so it can refer to it later. The computer thing for storing a number for later use is called a variable. A variable is a place to write down a number with a label so you can refer to it later.

Why not call a place you store a value a "storage space"? Why is it called a variable? Consider a locker or a lunch box. Lockers and lunch boxes are both storage spaces, but you can vary the contents of these storage spaces. You can put a sandwich in a lunch box, or some pieces of fruit, or some cookies. And you can add more things to a lunch box, and you can remove things from a lunchbox. Same for a locker. Some storage spaces allow you to vary the contents, some do not. When you burn some data or music on to a write-only CD, you cannot vary the contents. Memory storage on a write-only CD is not variable.

Creating a Variable

Select the Variables palette. Then click the button to "Make a Variable". You will be asked to give the variable a name. For this example, I used the name num. Then I click the box that this variable is for this sprite only. In a later example, we shall see why one would use the other choice. When we click the OK button, we now the second image at the left.

Suddenly there are two more puzzle pieces and a new little value piece (the shape that fits in spots where numbers are allowed.) This new value thing is a storage container called num. The checkbox next to the variable tells Scratch we want to see the variable displayed on the stage. If we uncheck the box, the variable will be hidden from our audience. Usually we leave the variables visible when we are designing the program, and then we hide the variables when the program works.

We created a variable so we can store a number so we can use that number in more than one place. In particular, we want to show the number to the viewer, and we also want to compare the number to 6. How do we use the variable to store a value and then refer back to that value?
Using a Variable

Here we see hello5 changed to use our num variable. There are two important changes. At the start of our script, we select a random number and we use the new puzzle piece to store that random value in the variable. Computer people often use the phrase set num to ... to describe the action of storing a value in a variable. I use the phrase num gets .. in the same sense one says to a waitress Fred gets the grilled cheese.

The other important change is that we drag the num variable thing into the Say statement and also into the comparison expression. In fact, a variable can be plugged into any roundish box designed for numbers.

This example shows one use of variables: a place to store a value so we can use the value in more than one place. Here, we want to display the value, and we also want to compare the value to 6. If we did not record the value somewhere, we would not be able to use it in two different parts of the program. In Scratch, variables can only store numbers. In other programming languages, variables can store words, sentences, even blocks of text. The other languages we shall study in this course allow this. Variables are used in many, many ways. We shall look at a few examples to see some of the applications of variables.

3. Variables for Counting

Computers can count. Can cows count? We shall program a cow to count clicks. We shall click on our cow some number of times. Then, when we press the ‘m’ key, the cow will moo that many times.

Before we write the code, think how you would do this if you were the cow. I’ll tap the blackboard some number of times. Then when I say ’m’, you moo that many times with a brief pause between each moo. Ready, here goes.

Then, having done that, ask yourself how you did it. Well, you probably kept a count on your fingers, on paper, or in your head. Each time you heard a click, you added 1 to the count. Then when you heard the ’m’, you used that count to determine how many moo’s to say. Here is Scratch code to do some of that.

There are three scripts and one variable for this cow. The variable is called clicks and will be used to keep track of how many times the cow has been clicked. One script responds to click events. Each time the cow is clicked, the value of clicks is changed by 1.

Another script responds to a the ‘r’ key being pressed. This script resets the counter to zero.

Finally, the third script is the one that responds to the ’m’ key. This one needs code. We can write at least three versions of this code. Let us see how many different ways we can do it. This exercise in creating and comparing different ways of solving the same problem is our first introduction to the imaginative variation in using the same basic set of tools.

In Scratch, and in most programming languages, variables can be used for communication. For example, a movie theatre can post its schedule on a website, and patrons can view the website to read the information.

Similarly, a sprite can store a number in a variable, and another sprite can examine that variable to read the number. Here, though, the variable has to be viewable by all sprites. Keeping a movie schedule on a private website prevents customers from seeing what films are shown when. Therefore, our next topic is how sprites can use variables to communicate.
4. Variables for Communication

In our Scratch fields, cows can add, and they form study groups to practice their arithmetic skills. Here is a picture of one group in action. The two cows on the left pick random integers and say them. The cow on the right says the sum.

This project uses three sprites, one for each cow. The project uses three variables. Each of the cows on the left, when clicked, thinks of a number and stores that number in a variable, then says the number. When we click the cow on the right, that cow has to add the two numbers her classmates have picked.

How can cow3 know the values of the variables of her friends? The answer is we make the variables used by cow1 and cow2 to be available For all sprites. In computer terms, these are global variables. On the other hand, the variable cow3 uses to store the sum of those two variables is private: for that cow only. Most computer languages include the ideas of global variables and private variables. In Unix and Javascript, we shall see global variables and private variables.

Global/Local and Public/Private

In Scratch, variables come in two flavors: "all sprites" or "this sprite only". This distinction is not isolated to Scratch -- most computer languages provide, at a minimum, two levels of visibility. Computer programs, like you, have a private life and a public life. Some information is private -- none of anyone else’s business. Some information is public -- anyone can see the information, maybe even update the information.

What words do we use to distinguish personal information from public information? Scratch uses "this sprite only" and "all sprites". Some computer languages use "private" and "public". Some other languages use "local" and "global". In all three of these examples, terms are used that distinguish between information limited to a small area with information available across a broader area.

This idea of what programs or people can see what data applies to almost everything in modern computing. On a website, some pages can only be seen by users who know a password. On a Unix system, every file and folder can be set to be private to its owner, viewable by a group of users, or viewable by any user on the computer. And the visibility and accessibility of files and folders can involve reading, writing, and running programs. Scratch is much simpler: either all sprites can use a variable or only one sprite can.

5. Programming Motion and Interaction

Sprites move around the stage. We shall now look in more detail at the geometry of the stage and some of the ways we can program sprites to move in certain ways. Each sprite has a position and a direction. The position is given by the \(x\) and \(y\) coordinates as shown on the left side of this diagram, and the direction is given by angles as shown on the right side of this diagram.

We shall now examine a few ways to control the motion of a sprite.
**Bouncing Sprites with a Trail**

Create two sprites. I used soccer balls, but you can use anything. Write this script for one of the sprites:

```scratch
when flag clicked
pen down
forever
    if on edge, bounce
    if touching [sprite2]?
        point towards [sprite2]
        turn cw (180) degrees
        move (5) steps
        wait (0.01) secs
```

Now move the static sprite to make the moving one draw a picture. What if you wanted to make both sprites move and bounce off each other? What about counting the number of times they bounce? What if you wanted the speed of the sprites to change as the number of bounces increased? What if you wanted the sprites to rotate as they bounced off the walls?

**Coordinated Motion**

This project shows how sprites can interact in more intelligent ways than those in the previous example.

This example demonstrates the three main ways we can specify motion:

- **relative** The commands to `move ( ) steps` or `turn ( ) degrees` tell the sprite to move relative to its current position and direction. These commands specify relative motion.
- **absolute** The commands `go to x: ( ) y: ( )` and `point in direction ( )` tell the sprite to move, regardless of its current location and direction, to specific places or angles.
- **dependent** The commands to `point towards [ ]` and `go to [ ]` tell the sprite to change direction and position that depend on another sprite. (note: the term ‘dependent’ was invented by Alex Lavers on Sep 13, 2007.)

**6. A Class Exercise: Bouncing without Using Bounce**

In the previous class, I said the cows in my animated Dutch landscape walked to the end of the screen and got stuck. One simple solution is:

```scratch
when flag clicked
forever
    if on edge, bounce
    move (10) steps
    wait (0.1) secs
```

but this has an odd visual effect. How can we avoid using bounce to get the result we want? How many different ways can we solve this problem? What if we wanted to have the cow speed up near the middle of the stage and slow down as it approached the sides? How could we do that?