

Control

We have animated our program by using arithmetic coupled with the variable `frameCount`. Using this technique in various ways, we can create some very interesting movement patterns. If we add the `random()` function and, possibly the `millis()` function, we can do much more. But what if we want to respond to input by the user from either the keyboard or the mouse or input of some other form from some other source (the net possibly)? We need more tools in our tool kit. One of these tools is the control syntax.

Control refers to a group of syntactic structures in the language that allows the program to “control” or decide

- which functions to call and which ones to skip
- how many time to repeat a function call or group of function calls

The first bullet uses control structures we describe as “*selection*” or “*branching*” control structures. This is where we start. We will return to the second bullet’s structures (the *looping* or *iterative* control structures) in a few days. The control syntax for selection is the `if`. It has several forms:

<pre>if (test or guard) { do this if the test is true }</pre> <p><i>do nothing if the test is false</i></p> <p><i>There is only one branch here - the if branch. Based on the test, Processing will either select the if branch or select nothing</i></p>	<pre>if (test or guard) { do this if the test is true } else { do this if the test is false }</pre> <p><i>There are two branches here: the if and the else. Based on the test, Processing will select only one of these two branches. We say that, “The execution of the program flows through either the if or the else branch but not both of them.”</i></p>
---	--

The test or guard must evaluate to a value of true or false. It cannot evaluate to an int or float value. It must be evaluate to true or false.

Expressions that evaluate to true or false are called boolean expressions in Processing. The **b** is lower case.

- We can declare boolean variables:

```
boolean b = false;
```

- We can define functions that have a boolean return type:

```
boolean weHitTheTarget( int x, int y )
{
    . . .
}
```

- We can use boolean expressions with the **relational** operators that you used in your earlier math classes;

Operator	Meaning
==	Equality
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
!=	Not equal

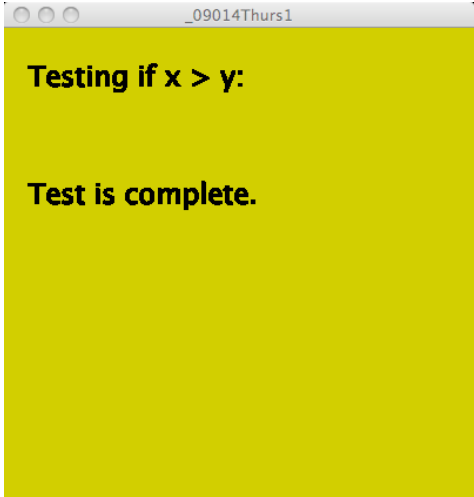
Note that the operators composed of two characters MUST NOT have a space between the two characters. The equality operator is two equal signs. One equal sign is the assignment operator.

- We can also use the relational operators to combine Boolean expressions.

Operator	Meaning
&&	AND
	OR
!	NOT

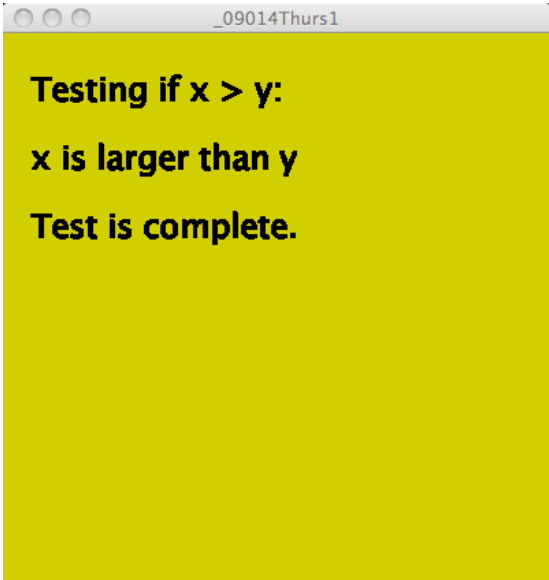
For the next discussion of the if control structure below, we will use boolean expressions. In a later set of notes, we will use boolean variables and boolean functions.

Here are programs that demonstrate the use the if:

<pre>int x, y; void setup() { size(400, 400); textSize(24); fill(0); background(200, 200, 0); x = 100; y = 101; } void draw() { demoIf(); noLoop(); } void demoIf() { text("Testing if x > y: ", 20, 50); if (x > y) { text("x is larger than y" , 20, 100); } text("Test is complete ", 20, 150); }</pre>	
---	---

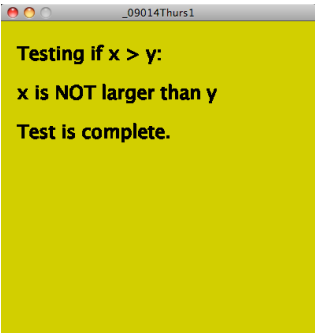
In the program above, the boolean expression that makes up test or guard of the if evaluates to false because x is less than y. Since the test is false, the code inside the braces is skipped. It is not selected. It is not executed.

In this next program, the value of x is larger than the value of y:

<pre>int x, y; void setup() { size(400, 400); textSize(24); fill(0); background(200, 200, 0); x = 100; y = 101; } void draw() { demoIf(); noLoop(); } void demoIf() { text("Testing if x > y: ", 20, 50); if (x > y) { text("x is larger than y" , 20, 100); } text("Test is complete ", 20, 150); }</pre>	
---	---

*In the program above, the test evaluates to true so the code within the braces *is* selected. The code with the braces is executed and we see a different output.*

Next we switch to the other form: if/else:

<pre>int x, y; void setup() { size(400, 400); textSize(24); fill(0); background(200, 200, 0); x = 100; y = 101; } void draw() { demoIf(); noLoop(); } void demoIf() { text("Testing if x > y: ", 20, 50); if (x > y) { text("x is larger than y" , 20, 100); } else { text ("x is NOT larger than y" , 20, 100); } text("Test is complete ", 20, 150); }</pre>	
---	--

The value of x is less than y so the test is false. The program now has an else as part of the if. The else is executed when the test evaluates to false as shown in the output to the right.

Question: What happens if x and y are equal???
Code it and see before reading further. . .

Here is the code and how it is executed when the values of x and y are different

<pre> int x, y; void setup() { size(400, 400); textSize(24); fill(0); background(200, 200, 0); x = 100; y = 101; } void draw() { demoIf(); noLoop(); } void demoIf() { text("Testing if x > y: ", 20, 50); if (x > y) { text("x is larger than y" , 20, 100); } else if (x < y) { text ("x is NOT larger than y" , 20, 100); } else { text ("x EQUAL to y" , 20, 100); } text("Test is complete ", 20, 150); } </pre>	<p>Assume this:</p> <p>x = 200; y = 100</p> <p> </p> <p> </p> <p> </p> <p> </p> <p> </p> <p>true</p> <p> </p> <p>do this</p> <p> </p> <p>skip this</p> <p> </p> <p>to here</p> <p> </p> <p>do this</p>	<p>Assume this:</p> <p>x = 100; y = 200</p> <p> </p> <p> </p> <p> </p> <p> </p> <p> </p> <p>false</p> <p> </p> <p>skip this</p> <p> </p> <p>true</p> <p> </p> <p>do this</p> <p> </p> <p>skip this</p> <p> </p> <p>to here</p> <p> </p> <p>do this</p>	<p>Assume this:</p> <p>x = 100; y = 100</p> <p> </p> <p> </p> <p> </p> <p> </p> <p> </p> <p>false</p> <p> </p> <p>skip this</p> <p> </p> <p>false</p> <p> </p> <p>skip this</p> <p> </p> <p>do this</p> <p> </p> <p>do this</p>
--	--	--	---

The third if is not needed. In this example there are only three possibilities: $x > y$, $x < y$, $x == y$. If the first two possibilities are false, then the third one must be true. This means that we do not need to make the third test:

```
if ( x == y )
```

Two more things:

We can put ifs within other ifs and other else's.

```
if ( )
{
    if( )
    {

    }
    else
    {

    }
}
else
{
    if( )
    {

    }
}
}
```

This is called nesting.

The if/else that is colored blue and red is the outer if (or outer if/else...)

The orange if/else is nested within the outer if.

The green if is nested within the outer else.

You can combine these in any reasonable way that is needed to solve the problem.

We can also “cascade” a series of if/elses. You saw this back on page 6 and it will be very common in our code.

```
if ( )
{

}
else if( )
{

}
else if
{

}
else
{

}
```

When the test in the first if is true, the code inside the braces of the first if is executed and rest of the code in the entire if is finished. No more code (the tests or the code in the braces) is executed.

When the test in the first if is false, execution shifts to the test in the second if.

When the test of the second if is true, the code within the braces of the second if is executed and the rest of the code is skipped

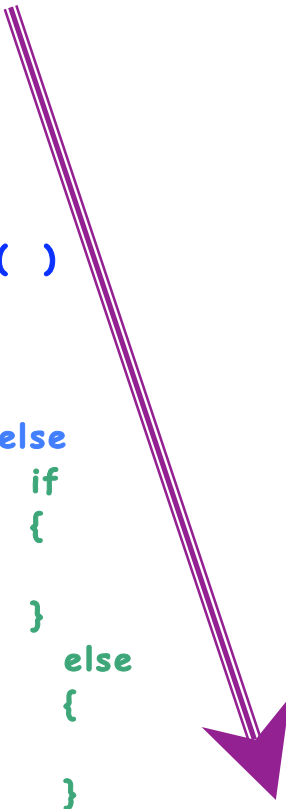
When the tests in both the first and second ifs are false, execution shifts to the third if.

When the test of the third if is true, the code within the braces of the third if is executed. When the test is false, the code within the else of the third if is executed.

You will see a cascaded if/else structure very soon in your code.

You may be wondering why these are called cascaded. The reason is the way “we used to format them” in the code. In the distant past the code might have looked like this:

```
if ( )  
{  
  
}  
  
else  
  if( )  
  {  
  
  }  
  
  else  
    if  
    {  
  
    }  
  
    else  
    {  
  
    }
```



Sorta' like a waterfall. . .