5 Conditionals

"That language is an instrument of human reason, and not merely a medium for the expression of thought, is a truth generally admitted."
—George Boole

"The way I feel about music is that there is no right and wrong. Only true and false."
—Fiona Apple

In this chapter:
– Boolean expressions.
– Conditional statements: How a program produces different results based on varying circumstances.
– *If, Else If, Else.*

5.1 Boolean Expressions

What's your favorite kind of test? Essay format? Multiple choice? In the world of computer programming, we only take one kind of test: a *boolean* test—true or false. A *boolean expression* (named for mathematician George Boole) is an expression that evaluates to either true or false. Let's look at some common language examples:

- I am hungry. → true
- I am afraid of computer programming. → false
- This book is a hilarious read. → false

In the formal logic of computer science, we test relationships between numbers.

- 15 is greater than 20 → false
- 5 equals 5 → true
- 32 is less than or equal to 33 → true

In this chapter, we will learn how to use a variable in a boolean expression, allowing our sketch to take different paths depending on the current value stored in the variable.

- $x > 20$ → depends on current value of x
- $y == 5$ → depends on current value of y
- $z <= 33$ → depends on current value of z

The following operators can be used in a boolean expression.

Relational Operators

>	greater than	<=	less than or equal to
<	less than	==	equality
>=	greater than or equal to	!=	inequality

5.2 Conditionals: If, Else, Else If

Boolean expressions (often referred to as "conditionals") operate within the sketch as questions. Is 15 greater than 20? If the answer is yes (i.e., true), we can choose to execute certain instructions (such as draw a rectangle); if the answer is no (i.e., false), those instructions are ignored. This introduces the idea of branching; depending on various conditions, the program can follow different paths.

In the physical world, this might amount to instructions like so:

If I am hungry then eat some food, otherwise if I am thirsty, drink some water, otherwise, take a nap.

In *Processing*, we might have something more like:

If the mouse is on the left side of the screen, draw a rectangle on the left side of the screen.

Or, more formally, with the output shown in Figure 5.1,

```
if (mouseX < width/2) {
  fill(255);
  rect(0,0,width/2,height);
}
```

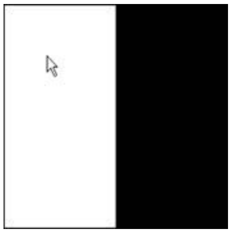
fig. 5.1

The boolean expression and resulting instructions in the above source code is contained within a block of code with the following syntax and structure:

```
if (boolean expression) {
  // code to execute if boolean expression is true
}
```

The structure can be expanded with the keyword *else* to include code that is executed if the boolean expression is false. This is the equivalent of "otherwise, do such and such."

```
if (boolean expression) {
  // code to execute if boolean expression is true
} else {
  // code to execute if boolean expression is false
}
```

For example, we could say the following, with the output shown in Figure 5.2.

If the mouse is on the left side of the screen, draw a white background, otherwise draw a black background.

```
if (mouseX < width/2) {
  background(255);
} else {
  background(0);
}
```

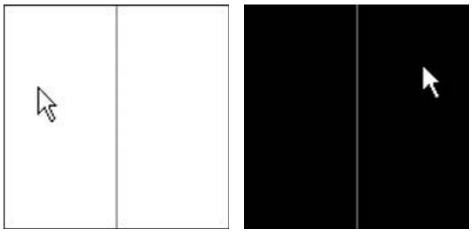
fig. 5.2

Finally, for testing multiple conditions, we can employ an "else if." When an *else if* is used, the conditional statements are evaluated in the order presented. As soon as one boolean expression is found to be true, the corresponding code is executed and the remaining boolean expressions are ignored. See Figure 5.3.

```
if (boolean expression #1) {
  // code to execute if boolean expression #1 is true
} else if (boolean expression #2) {
  // code to execute if boolean expression #2 is true
} else if (boolean expression #n) {
  // code to execute if boolean expression #n is true
} else {
  // code to execute if none of the above
  // boolean expressions are true
}
```

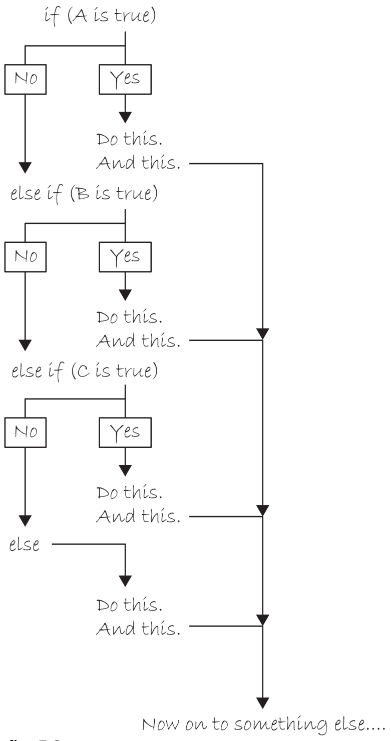

fig. 5.3

Taking our simple mouse example a step further, we could say the following, with results shown in Figure 5.4.

If the mouse is on the left third of the window, draw a white background, if it is in the middle third, draw a gray background, otherwise, draw a black background.

```
if (mouseX < width/3) {
  background(255);
} else if (mouseX < 2*width/3) {
  background(127);
} else {
  background(0);
}
```

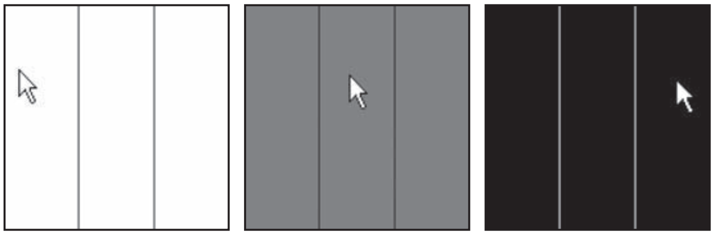
fig. 5.4

Exercise 5-1: Consider a grading system where numbers are turned into letters. Fill in the blanks in the following code to complete the boolean expression.

```
float grade = random(0,100);
if (_______) {
    println("Assign letter grade A.");
} else if (________) {
    println (________);
```

In one conditional statement, you can only ever have one *if* and one *else*. However, you can have as many *else if*'s as you like!

```
} else if (________) {
   println(________);
} else if (________) {
   println(________);
} else {
   println(________);
}
```

Exercise 5-2: Examine the following code samples and determine what will appear in the message window. Write down your answer and then execute the code in Processing *to compare.*

Problem #1: Determine if a number is between 0 and 25, 26 and 50, or greater than 50.

```
int x = 75;

if (x > 50) {
  println(x + " is greater than
   50!");
} else if (x > 25) {
  println(x + " is greater than
   25!");
} else {
  println(x + " is 25 or
   less!");
}
```

OUTPUT:________________________

```
int x = 75;

if(x > 25) {
  println(x + " is greater
   than 25!");
} else if (x > 50) {
  println(x + " is greater
   than 50!");
} else {
  println(x + " is 25 or
   less!");
}
```

OUTPUT:________________________

Although the syntax is correct, what is problematic about the code in column two above?

Problem #2: If a number is 5, change it to 6. If a number is 6, change it to five.

```
int x = 5;                        int x = 5;

println("x is now: " + x);        println("x is now: " + x);
if (x == 5) {                     if (x == 5) {
  x = 6;                            x = 6;
}                                 } else if (x == 6) {
if (x == 6) {                       x = 5;
  x = 5;                          }
}                                 println("x is now: " + x);
println("x is now: " + x);

OUTPUT:____________________       OUTPUT:____________________
```

Although the syntax is correct, what is problematic about the code in column one above?

It is worth pointing out that in Exercise 5-2 when we test for equality we must use *two* equal signs. This is because, when programming, asking if something is equal is different from assigning a value to a variable.

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

"Is *x* equal to *y*?" Use double equals!

```
x = y;
```

"Set *x* equal to *y*." Use single equals!

5.3 Conditionals in a Sketch

Let's look at a very simple example of a program that performs different tasks based on the result of certain conditions. Our pseudocode is below.

Step 1. Create variables to hold on to red, green, and blue color components. Call them *r*, *g*, and *b*.
Step 2. Continuously draw the background based on those colors.
Step 3. If the mouse is on the right-hand side of the screen, increment the value of *r*, if it is on the left-hand side decrement the value of *r*.
Step 4. Constrain the value *r* to be within 0 and 255.

This pseudocode is implemented in *Processing* in Example 5-1.

Example 5-1: Conditionals

```
float r = 150;
float g = 0;
float b = 0;


void setup() {
  size(200,200);
}


void draw() {
  background(r,g,b);
  stroke(255);
  line(width/2,0,width/2,height);


  if(mouseX > width/2) {
    r = r + 1;
  } else {
    r = r - 1;
  }


  if (r > 255) {
    r = 255;
  } else if (r < 0) {
    r = 0;
  }
}
```

1. Variables.

2. Draw stuff.

3. "If the mouse is on the right side of the screen" is equivalent to "if ***mouseX*** is greater than width divided by 2."

4. If *r* is greater than 255, set it to 255. If *r* is less than 0, set it to 0.

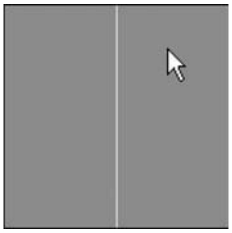
fig. 5.5

Constraining the value of a variable, as in Step 4, is a common problem. Here, we do not want color values to increase to unreasonable extremes. In other examples, we might want to constrain the size or location of a shape so that it does not get too big or too small, or wander off the screen.

While using if statements is a perfectly valid solution to the constrain problem, *Processing* does offer a function entitled ***constrain()*** that will get us the same result in one line of code.

```
if (r > 255) {
  r = 255;
} else if (r < 0) {
  r = 0;
}

r = constrain(r,0,255);
```

Constrain with an "if" statement.

Constrain with the ***constrain()*** function.

constrain() takes three arguments: the value we intend to constrain, the minimum limit, and the maximum limit. The function *returns* the "constrained" value and is assigned back to the variable *r*. (Remember what it means for a function to *return* a value? See our discussion of ***random()***.)

Getting into the habit of *constraining* values is a great way to avoid errors; no matter how sure you are that your variables will stay within a given range, there are no guarantees other than ***constrain()*** itself. And someday, as you work on larger software projects with multiple programmers, functions such as ***constrain()*** can ensure that sections of code work well together. Handling errors before they happen in code is emblematic of good style.

Let's make our first example a bit more advanced and change all three color components according to the mouse location and click state. Note the use of ***constrain()*** for all three variables. The system variable ***mousePressed*** is true or false depending on whether the user is holding down the mouse button.

Example 5-2: More conditionals

```
float r = 0;
float b = 0;
float g = 0;

void setup() {
    size(200,200);
}

void draw() {
  background(r,g,b);
  stroke(0);

  line(width/2,0,width/2,height);
  line(0,height/2,width,height/2);

  if(mouseX > width/2) {
    r = r + 1;
  } else {
    r = r - 1;
  }

  if (mouseY > height/2) {
    b = b + 1;
  } else {
    b = b - 1;
  }

  if (mousePressed) {
    g = g + 1;
  } else {
    g = g - 1;
  }

  r = constrain(r,0,255);
  g = constrain(g,0,255);
  b = constrain(b,0,255);
}
```

Three variables for the background color.

Color the background and draw lines to divide the window into quadrants.

If the mouse is on the right-hand side of the window, increase red. Otherwise, it is on the left-hand side and decrease red.

If the mouse is on the bottom of the window, increase blue. Otherwise, it is on the top and decrease blue.

If the mouse is pressed (using the system variable ***mousePressed***) increase green.

Constrain all color values to between 0 and 255.

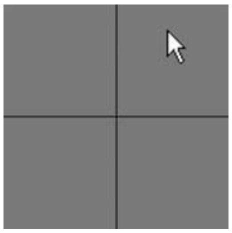
fig. 5.6

Exercise 5-3: Move a rectangle across a window by incrementing a variable. Start the shape at x coordinate 0 and use an if statement to have it stop at coordinate 100. Rewrite the sketch to use ***constrain()*** *instead of the if statement. Fill in the missing code.*

```
// Rectangle starts at location x
float x = 0;

void setup() {
  size(200,200);
}

void draw() {
  background(255);
  // Display object
  fill(0);
  rect(x,100,20,20);

  // Increment x
  x = x + 1;
  ______________________________________________

  ______________________________________________

  ______________________________________________

}
```

5.4 Logical Operators

We have conquered the simple ***if*** statement:

If my temperature is greater than 98.6, then take me to see the doctor.

Sometimes, however, simply performing a task based on one condition is not enough. For example:

If my temperature is greater than 98.6 ***OR*** *I have a rash on my arm, take me to see the doctor.*

If I am stung by a bee ***AND*** *I am allergic to bees, take me to see the doctor.*

We will commonly want to do the same thing in programming from time to time.

If the mouse is on the right side of the screen ***AND*** *the mouse is on the bottom of the screen, draw a rectangle in the bottom right corner.*

Our first instinct might be to write the above code using a nested if statement, like so:

```
if (mouseX > width/2) {
  if (mouseY > height/2) {
    fill(255);
    rect(width/2,height/2,width/2,height/2);
  }
}
```

In other words, we would have to bypass *two* if statements before we can reach the code we want to execute. This works, yes, but can be accomplished in a simpler way using what we will call a "logical and," written as two ampersands ("&&"). A single ampersand ("&") means something else[1] in *Processing* so make sure you include two!

|| (logical OR)
&& (logical AND)
! (logical NOT)

A "logical or" is two vertical bars (AKA two "pipes") "||". If you can't find the pipe, it is typically on the keyboard as shift-backslash.

```
if (mouseX > width/2 && mouseY > height/2) {
  fill(255);
  rect(width/2,height/2,width/2,height/2);
}
```

If the mouse is on the right side *and* on the bottom.

In addition to && and ||, we also have access to the logical operator "not," written as an exclamation point: !

*If my temperature is **NOT** greater than 98.6, I won't call in sick to work.*

*If I am stung by a bee **AND** I am **NOT** allergic to bees, do not worry!*

A *Processing* example is:

*If the mouse is **NOT** pressed, draw a circle, otherwise draw a square.*

```
if (!mousePressed) {
  ellipse(width/2,height/2,100,100);
} else {
  rect(width/2,height/2,100,100);
}
```

! means not. ***"mousePressed"*** is a *boolean* variable that acts as its own boolean expression. Its value is either true or false (depending on whether or not the mouse is currently pressed). Boolean variables will be explored in greater detail in Section 5.6.

Notice this example could also be written omitting the *not*, saying:

If the mouse is pressed, draw a square, otherwise draw a circle.

[1]"&" or "|" are reserved for *bitwise* operations in *Processing*. A bitwise operation compares each bit (0 or 1) of the binary representations of two numbers. It is used in rare circumstances where you require low-level access to bits.

Exercise 5-4: Are the following boolean expressions true or false? Assume variables x = 5 and y = 6.

```
!(x > 6)          ________________________________
(x==6 && x==5)    ________________________________
(x==6 || x==5)    ________________________________
(x>-1 && y<10)    _________________________________
```

Although the syntax is correct, what is flawed about the following boolean expression?

```
(x > 10  &&  x < 5) ___________________________
```

Exercise 5-5: Write a program that implements a simple rollover. In other words, if the mouse is over a rectangle, the rectangle changes color. Here is some code to get you started.

```
int x = 50;
int y = 50;
int w = 100;
int h = 75;

void setup() {
  size(200,200);
}

void draw() {
  background(0);

  stroke(255);
  if (_______ && _______ && _______ && _______) {
    _______
  } _______ {
    _______
  }
  rect(x,y,w,h);
}
```