```
        !false; // Returns true
```

Conditionals

Much of the power of a computer lies in its ability to make rudimentary decisions. At the root of this power lie the Boolean `true` and `false`. You can use these Boolean values to build conditionals and steer a program on a unique course. As you are building your flow of logic through code, remember that a machine can only make a single, simple decision at a time. Put enough of those decisions together, though, and you can build complex interactions.

The `if` Statement

The basis of conditionals is the `if` statement, which is structured like this:

Click here to view code image

```
if ( <some Boolean condition> )
{
  // do something
}
```

The `if` structure can be read as “if this is true, do this.” So, if you want to output “Hello World” to the Console if the value of `x` is greater than 5, you could write the following:

Click here to view code image

```
if (x > 5)
{
  print("Hello World");
}
```

Remember that the contents of the `if` statement condition must evaluate to either `true` or `false`. Putting numbers, words, or anything else in there will not work:

Click here to view code image

```
if ("Hello" == "Hello") // Correct

if (x + y) // Incorrect
```

Finally, any code that you want to run if the condition evaluates to `true` must go inside the opening and closing brackets that follow the `if` statement.

TIP

Odd Behavior

Conditional statements use a specific syntax and can give you strange behaviors if you don't follow that syntax exactly. You may have an `if` statement in your code and notice that something isn't quite right. Maybe the condition code runs all the time, even when it shouldn't. You may also notice that it never runs, even if it should. You should be aware of two common causes of this. First, the `if` condition may not have a semicolon after it. If you write an `if` statement with a semicolon, the code following it will always run. Second, be sure that you are using the equality operator (`==`) and not the assignment operator (`=`) inside the `if` statement. Doing otherwise leads to bizarre behavior:

Click here to view code image

```
if (x > 5); // Incorrect
if (x = 5)  // Incorrect
```

Being mindful of these two common mistakes will save you heaps of time in the future.

The `if / else` Statement

The `if` statement is nice for conditional code, but what if you want to diverge your program down two different paths? The `if / else` statement enables you to do that. The `if / else` is the same basic premise as the `if` statement, except it can be read more like "if this is true do this; otherwise (else), do this other thing." The `if / else` statement is written like this:

Click here to view code image

```
if ( <some Boolean condition> )
{
    // Do something
}
else
{
    // Do something else
}
```

For example, if you want to print "X is greater than Y" to the Console if the variable `x` is larger than the variable `y`, or you want to print "Y is greater than X" if `x` isn't bigger than `y`, you could write the following:

Click here to view code image

```
if (x > y)
{
    print("X is greater than Y");
}
else
{
    print("Y is greater than X");
}
```

The `if` / `else` if Statement

Sometimes you want your code to diverge down one of many paths. You might want the user to be able to pick from a selection of options (such as a menu, for example). An `if / else if` is structured in much the same way as the previous two structures, except that it has multiple conditions:

Click here to view code image

```
if( <some Boolean condition> )
{
    // Do something
}
else if ( <some other Boolean condition> )
{
    // Do something else
}
else {
    // The else is optional in the IF  ELSE IF statement
    / Do something else
}
```

For example, if you want to output a person’s letter grade to the Console based on his percentage, you could write the following:

Click here to view code image

```
if (grade >= 90) {
    print ("You got an A");
} else if (grade >= 80) {
    print ("You got a B");
} else if(grade >= 70) {
   print ("You got a C");
} else if (grade >= 60) {
    print ("You got a D");
} else {
    print ("You got an F");
}
```

TIP

The Great Bracket Crusade

You may have noticed that sometimes I place opening brackets (also called curly braces) on their own lines and sometimes I put the opening bracket on the same line as a class name, a method name, or an `if` statement. The truth is that either is completely fine, and the choice is up to personal preference. That being said, great online debates and code wars have been waged (and still rage on) over which method is superior. Truly, this is a topic of great importance. There are no bystanders. Everyone must choose a side.…

TIP

Single-line if Statements

Strictly speaking, if your `if` statement code is only a single line, you do not need to have the curly braces (also called brackets). Therefore, your code that looks like this:

Click here to view code image

```
if (x > y)
{
    print("X is greater than Y");
}
```

could also be written as follows:

Click here to view code image

```
if (x > y)
    print("X is greater than Y");
```

However, I recommend that you include the curly braces for now. This can save a lot of confusion later, as your code gets more complex. Code inside curly braces is referred to as a *code block* and is executed together.

Iteration

You have so far seen how to work with variables and make decisions. This is certainly useful if you want to do something like add two numbers together. But what if you want to add all the numbers between 1 and 100 together? What about between 1 and 1000? You definitely would not want to type all of that redundant code. Instead, you can use something called *iteration* (commonly referred to as *looping*). There are two primary types of loops to work with: the `while` loop and the `for` loop.

The `while` Loop

The `while` loop is the most basic form of iteration. It follows a structure similar to that of an `if` statement:

Click here to view code image

```
While ( <some Boolean condition> )
{
    // do something
}
```

The only difference is that an `if` statement runs its contained code only once, whereas a loop runs the contained code over and over until the condition becomes `false`. Therefore, if you want to add together all the numbers between 1 and 100 and then output them to the Console, you could write something like this:

Click here to view code image

```
int sum = 0;
int count = 1;

while (count <= 100)
{
    sum += count;
    count++;
}

print(sum);
```

As you can see, the value of `count` starts at 1 and increases by 1 every iteration—or execution of the loop—until it equals 101. When `count` equals 101, it is no longer less than or equal to 100, so the loop exits. Omitting the `count++` line results in the loop running infinitely—so be sure it's there. During each iteration of the loop, the value of `count` is added to the variable `sum`. When the loop exits, the sum is written to the Console.

In summation, a `while` loop runs the code it contains over and over as long as its condition is `true`. When its condition becomes `false`, it stops looping.

The `for` Loop

The `for` loop follows the same idea as the `while` loop, except it is structured a bit differently. As you saw in the code for the `while` loop, you had to create a `count` variable, you had to test the variable (as the condition), and you had to increase the variable all on three separate lines. The `for` loop condenses that syntax down to a single line. It looks like this:

Click here to view code image

```
for (<create a counter>; <Boolean conditional>; <increment the counter >
{
    // Do something
}
```

The `for` loop has three special *compartments* for controlling the loop. Notice the semicolons, not commas, in between the sections in the `for` loop header. The first compartment creates a variable to be used as a counter. (A common name for the counter is `i`, short for *iterator*.) The second compartment is the conditional statement of the loop. The third compartment handles increasing or decreasing the counter. The previous `while` loop example can be rewritten using a `for` loop. It would look like this:

Click here to view code image

```
int sum = 0;

for (int count = 1; count <= 100; count++)
{
    sum += count;
}

print(sum);
```

As you can see, the different parts of the loop get condensed and take up less space. You can see that the `for` loop is really good at things like counting.

Summary

In this hour, you took your first steps into video game programming. You started by looking at the basics of scripting in Unity. You learned how to make and attach scripts. You also looked at the basic anatomy of a script. From there, you studied the basic logical components of a program. You worked with variables, operators, conditionals, and loops.

Q&A

Q. How much programming is required to make a game?

A. Most games use some form of programming to define complex behaviors. The more complex the behaviors need to be, the more complex the programming needs to be. If you want to make games, you should definitely become comfortable with the concepts of programming. This is true even if you don’t intend to be the primary developer for a game. With that in mind, rest assured that this book provides everything you need to know to make your first few simple games.

Q. Is this all there is to scripting?

A. Yes and no. Presented in this text are the fundamental blocks of programming. They never really change but just get applied in new and unique ways. That said, a lot of what is presented here is simplified because of the complex nature of programming in general. If you want to learn more about programming, you should read books or articles specifically on the subject.

Workshop

Take some time to work through the questions here to ensure that you have a firm grasp of the material.

Quiz

1. What two languages does Unity allow you to use for programming?
2. True or False: The code in the `Start` method runs at the start of every frame.
3. Which variable type is the default floating-point number type in Unity?
4. Which operator returns the remainder of division?
5. What is a conditional statement?
6. Which loop type is best suited for counting?

Answers

1. C# and JavaScript
2. False. The `Start` method runs at the beginning of the scene. The `Update` method runs every frame.

3. `float`
4. Modulus
5. A code structure that allows the computer to choose a code path based on a simple decision
6. The `for` loop

Exercise

It can often be helpful to view coding structures as building blocks. Alone, each piece is simple. Put together, however, they can build complex entities. In the following steps, you will encounter multiple programming challenges. Use the knowledge you have gained in this hour to build a solution to each problem. Put each solution in its own script and attach the scripts to the Main Camera of a scene to ensure that they work. You can find the solution to this exercise in the book assets for Hour 7.

1. Write a script that adds together all the even numbers from 2 to 499. Output the result to the Console.
2. Write a script that outputs all the numbers from 1 to 100 to the Console except for multiples of 3 or 5. In place of multiples of 3 or 5, output "Programming is awesome!" (*Hint:* You can tell whether a number is a multiple of another number if the result of a modulus operation is 0; for example, `12 % 3 == 0` because 12 is a multiple of 3.)
3. In the Fibonacci sequence, you determine a number by adding the two previous numbers together. The sequence starts with 0, 1, 1, 2, 3, 5.…Write a script that determines the first 20 places of the Fibonacci sequence and outputs them to the Console.