Making Decisions in Your Program

If This Then That

Programs that always do the same thing aren't nearly as powerful or interesting as those that can make decisions. These decisions can be based on any number of things, from user input to the current state of the program. Based on some kind of *condition*, the program can choose a different set of instructions to run. The most common C# keyword used to make decisions like this is the `if` statement. Because the `if` statement depends on a condition, the statement is sometimes referred to as a *conditional* statement.

You can use `if` statements for all kinds of things in your programs. One common use for these statements is validating user input, such as checking to see if any arguments have been passed to a program or method. When you write an `if` statement, it consists of the `if` keyword, a logical expression inside of parentheses, and then one or more statements. It's generally considered a best practice to always wrap the statement(s) after an `if` statement in curly braces (`{ }`), unless it's a very short statement that can fit on the same line as the `if` statement. For example, you can exit a method by *returning* from it if a required argument is not specified:

```
static void Greet(string name)
{
    if (String.IsNullOrEmpty(name)) return;

    Console.WriteLine($"Hello, {name}!");
}
```

```
using System;

class Program
{
    static void Main()
    {
        Greet(""); // try calling this with "", null, and your name as arguments
    }

    static void Greet(string name)
    {
        if (String.IsNullOrEmpty(name)) return;

        Console.WriteLine($"Hello, {name}!");
    }
}
```

However, it can cause confusion if you don't use curly braces around the statements that go with your `if` statement, especially if your indentation is off:

```
var sum = 10;
if (someCondition)
    sum = sum + 5;
    sum = sum * 2;

Console.WriteLine(sum);
```

```
using System;

class Program
{
    static void Main()
    {
        bool someCondition = false; // try it with both true and false values

        var sum = 10;
        if (someCondition)
            sum = sum + 5;
            sum = sum * 2;

        Console.WriteLine(sum);
    }
}
```

In this example, when `someCondition` is true, the value of sum will be 30. Otherwise, it will be 20. However, based on the indentation, the reader might think that the `sum = sum * 2` line only occurs when the `if` condition is true, which isn't the case. If that's the correct behavior, the code should be rewritten as:

```
var sum = 10;
if (someCondition)
{
    sum = sum + 5;
    sum = sum * 2;
}

Console.WriteLine(sum);
```

```
using System;

class Program
{
    static void Main()
    {
        bool someCondition = false;

        var sum = 10;
        if (someCondition)
        {
            sum = sum + 5;
            sum = sum * 2;
        }

        Console.WriteLine(sum);
    }
}
```

Now the above code will print a sum of 30 when the `if` condition evaluates to true, or the original sum value of 10 otherwise.

> **Tip** {.tip .newLanguage}
> It's a good idea to follow a standard code formatting convention when programming. Such conventions often vary, even between similar languages. In C#, for instance, it's recommended that opening curly braces always start on their own line, indented to match their associated keyword. Closing curly braces should also appear on their own line, indented to match their opening brace.

> **Tip** {.tip .vb}
> Unlike in VB, C# allows statements on the same line as `if` statements. Be aware, too, that unlike VB's `If-Then` statements that must end with an `End If` statement, C# uses curly braces to end many kinds of blocks, not just `if` statements.

Or Else!

Often, you'll want to have your program do one thing if a condition is true, and another, different thing if the condition is false. Either way, the code should execute **only one** of these two different paths. This is accomplished using an `if-else` statement. To create this statement, simply start with any `if` statement, and follow it with the `else` keyword and a statement block. Your `else` block can consist of a single statement without curly braces, but again it's generally much clearer, especially with `else` statements, to always wrap the statement(s) in curly braces.

For example, let's say you're writing a simple guessing game. You ask the user a question, and they answer it. Either they're correct, or they're not, and in either case you display a message telling them so. You could implement this as follows:

```
Console.WriteLine("What is the capital of Ohio?");
var answer = Console.ReadLine();
if (answer.ToLower() == "columbus")
{
    Console.WriteLine("Correct!");
}
else
{
    Console.WriteLine("Sorry, that's not the right answer.");
}
```

```
using System;
```

```
class Program
{
    static void Main()
    {
        Console.WriteLine("What is the capital of Ohio?");
        var answer = "your answer here"; // change this to your answer
        if (answer.ToLower() == "columbus")
        {
            Console.WriteLine("Correct!");
        }
        else
        {
            Console.WriteLine("Sorry, that's not the right answer.");
        }
    }
}
```

Of course, sometimes the decisions your program needs to make are a little more complicated than just "this or that". You can *nest* conditional statements within other conditional statements, creating potentially complex trees of conditions. It's generally a good idea to try to limit how deeply you do such nesting, as complex hierarchies of conditional logic are common sources of bugs in software. However, often they're necessary. For example, sorting algorithms often use comparison logic to determine which of two values is greater, or if they're equal, like this:

```
// two values, x and y, are being compared
if (x < y)
{
    return -1;
}
else
{
    if (x == y)
    {
        return 0;
    }
    else // x must be greater than y
    {
        return 1;
    }
}
```

As you can see, these kinds of nested conditionals can get quite long to write out. One way these can be shortened is by using the `else if` construct. If your `else` block is going to contain another `if` statement, use `else if` instead, which eliminates the need for one set of curly braces and a level of indentation. Here's the above example, rewritten to use `else if`:

```
// two values, x and y, are being compared
if (x < y)
{
    return -1;
}
```

```
else if (x == y) // == checks for equality between two values
{
    return 0;
}
else // x must be greater than y
{
    return 1;
}
```

If you find that you have more than 3 independent options in your conditional, you should consider using a different conditional structure: the *switch* statement.

Switch Statements

The `switch` keyword is used to have your program execute a different *section* of code for potentially many different cases. The `switch` statement relies on the value of a single statement. You can use a `switch` statement to achieve the same behavior as you can with an `if-else` statement in many cases, but generally `switch` is preferred when you have more than a few options to deal with. Within a `switch` block, you define separate `case` labels, with code to execute. You can have more than one label for a given section of code. Each section must not allow execution to continue to the next section. This is usually achieved by using the `break` keyword to break out of the `switch` statement, but you can use other statements (like `return`) as well.

The following example shows a simple `switch` statement based on an integer variable:

```
uint guessedNumber = 1;
switch (guessedNumber)
{
    case 0:
        Console.WriteLine("Sorry, 0 is not a valid guess.");
        break;
    case 1:
    case 2:
    case 3:
        Console.WriteLine("You guessed low.");
        break;
    case 4:
        Console.WriteLine("You guessed the right number!");
        break;
    default:
        Console.WriteLine("You guessed high.");
        break;
}
```

```
using System;

class Program
{
    static void Main()
    {
        uint guessedNumber = 1; // try different numbers to see resulting behavior
        switch (guessedNumber)
        {
            case 0:
                Console.WriteLine("Sorry, 0 is not a valid guess.");
                break;
            case 1:
            case 2:
            case 3:
                Console.WriteLine("You guessed low.");
                break;
            case 4:
                Console.WriteLine("You guessed the right number!");
                break;
            default:
                Console.WriteLine("You guessed high.");
                break;
        }
    }
}
```

In the example above, you can see how multiple different `case` labels (in this case, 1, 2, and 3) can be applied to a single switch section ("You guessed low").

> **Note** {.note}
> `uint` is an *unsigned* integer type, which cannot hold negative values.