200 XP

Exercise - Create decision logic with if statements

19 minutes

Most applications include a large number of execution paths. For example, an application could implement different execution paths based on which menu option a user selects. Developers refer to the code that implements different execution paths as *code branches*.

The most widely used code branching statement is the `if` statement. The `if` statement relies on a Boolean expression that is enclosed in a set of parentheses. If the expression is true, the code after the `if` statement is executed. If not, the .NET runtime ignores the code and doesn't execute it.

In this exercise, you'll practice writing `if` statements by creating a game. First you'll define the rules of the game, then you'll implement them in code.

You'll use the `Random.Next()` method to simulate rolling three six-sided dice. You'll evaluate the rolled values to calculate the score. If the score is greater than an arbitrary total, then you'll display a winning message to the user. If the score is below the cutoff, you'll display a losing message to the user.

- If any two dice you roll result in the same value, you get two bonus points for rolling doubles.
- If all three dice you roll result in the same value, you get six bonus points for rolling triples.
- If the sum of the three dice rolls, plus any point bonuses, is 15 or greater, you win the game. Otherwise, you lose.

You'll refine the rules as you learn more about the `if` statement.

> **ⓘ Important**
>
> This exercise makes extensive use of the `System.Random` class. You can refer to the Microsoft Learn module titled "Call methods from the .NET Class Library using C#" if you need a refresher how `Random.Next()` works.

Prepare your coding environment

This module includes activities that guide you through the process of building and running sample code. You're encouraged to complete these activities using Visual Studio Code as your development environment. Using Visual Studio Code for these activities will help you to become more comfortable writing and running code in a developer environment that's used by professionals worldwide.

1. Open Visual Studio Code.

   You can use the Windows Start menu (or equivalent resource for another OS) to open Visual Studio Code.

2. On the Visual Studio Code **File** menu, select **Open Folder**.

3. In the **Open Folder** dialog, navigate to the Windows Desktop folder.

   If you have a different folder location where you keep code projects, you can use that folder location instead. For this training, the important thing is to have a location that’s easy to locate and remember.

4. In the **Open Folder** dialog, select **Select Folder**.

   If you see a security dialog asking if you trust the authors, select **Yes**.

5. On the Visual Studio Code **Terminal** menu, select **New Terminal**.

   Notice that a command prompt in the Terminal panel displays the folder path for the current folder. For example:

   ```dos
   C:\Users\someuser\Desktop>
   ```

   > **Note**
   >
   > If you are working on your own PC rather than in a sandbox or hosted environment and you have completed other Microsoft Learn modules in this C# series, you may have already created a project folder for code samples. If that's the case, you can skip over the next step, which is used to create a console app in the TestProject folder.

6. At the Terminal command prompt, to create a new console application in a specified folder, type **dotnet new console -o ./CsharpProjects/TestProject** and then press Enter.

   This .NET CLI command uses a .NET program template to create a new C# console application project in the specified folder location. The command creates the CsharpProjects and TestProject folders for you, and uses TestProject as the name of your `.csproj` file.

7. In the EXPLORER panel, expand the **CsharpProjects** folder.

   You should see the TestProject folder and two files, a C# program file named Program.cs and a C# project file named TestProject.csproj.

8. In the EXPLORER panel, to view your code file in the Editor panel, select **Program.cs**.

9. Delete the existing code lines.

   You'll be using this C# console project to create, build, and run code samples during this module.

10. Close the Terminal panel.

Write code that generates three random numbers and displays them in output

1. Ensure that you have an empty Program.cs file open in Visual Studio Code.

2. To create the initial code for this exercise, enter the following:

```c#
Random dice = new Random();

int roll1 = dice.Next(1, 7);
int roll2 = dice.Next(1, 7);
int roll3 = dice.Next(1, 7);

int total = roll1 + roll2 + roll3;

Console.WriteLine($"Dice roll: {roll1} + {roll2} + {roll3} = {total}");
```

3. Take a minute to review the code that you entered.

To begin, you create a new instance of the `System.Random` class and store a reference to the object in a variable named `dice`. Then, you call the `Random.Next()` method on the `dice` object three times, providing both the lower and upper bounds to restrict the possible values between `1` and `6` (the upper bound is exclusive). You save the three random numbers in the variables `roll1`, `roll2`, and `roll3`, respectively.

Next, you sum the three dice rolls and save the value into an integer variable named `total`.

Finally, you use the `WriteLine()` method to display the three values using string interpolation.

When you run the code, you should see the following message (the numbers will be different).

Output

```
Dice roll: 4 + 5 + 2 = 11
```

This first task was a setup task. Now, you can add the decision logic into your code to make the game more interesting.

Add an if statement to display different messages based on the value of the total variable

1. In the Visual Studio Code Editor, locate the cursor at the bottom of your code file, and then create a blank code line.

2. To create your first game feature, enter the following `if` statements.

c#

```
if (total > 14)
{
    Console.WriteLine("You win!");
}

if (total < 15)
{
    Console.WriteLine("Sorry, you lose.");
}
```

These two `if` statements are used to handle the winning and losing scenarios. Take a minute to examine the first `if` statement.

Notice that the `if` statement is made up of three parts:

- The `if` keyword
- A *Boolean expression* between parenthesis `()`
- A *code block* defined by curly braces `{ }`

At run time, the Boolean expression `total > 14` is evaluated. If this is a true statement (if the value of `total` is greater than `14`) then the flow of execution will continue into the code defined in the code block. In other words, it will execute the code in the curly braces.

However, if the Boolean expression is false (the value of `total` not greater than `14`) then the flow of execution will skip past the code block. In other words, it will not execute the code in the curly braces.

Finally, the second `if` statement controls the message if the user loses. In the next unit, you'll use a variation on the `if` statement to shorten these two statements into a single statement that more clearly expresses the intent.

What is a Boolean expression?

A Boolean expression is any code that returns a Boolean value, either `true` or `false`. The simplest Boolean expressions are simply the values `true` and `false`. Alternatively, a Boolean expression could be the result of a method that returns the value `true` or `false`. For example, here's a simple code example using the `string.Contains()` method to evaluate whether one string contains another string.

c#

```
string message = "The quick brown fox jumps over the lazy dog.";
bool result = message.Contains("dog");
Console.WriteLine(result);

if (message.Contains("fox"))
{
    Console.WriteLine("What does the fox say?");
}
```

Because the `message.Contains("fox")` returns a `true` or `false` value, it qualifies as a Boolean expression and can be used in an `if` statement.

Other simple Boolean expressions can be created by using operators to compare two values. Operators include:

- `==`, the "equals" operator, to test for equality
- `>`, the "greater than" operator, to test that the value on the left is greater than the value on the right
- `<`, the "less than" operator, to test that the value on the left is less than the value on the right
- `>=`, the "greater than or equal to" operator
- `<=`, the "less than or equal to" operator
- and so on

> **Note**
>
> The C# training series on Microsoft Learn devotes an entire module to Boolean expressions. There are many operators you can use to construct a Boolean expression, and you'll only cover a few of the basics here in this module. For more on Boolean expressions, see the Microsoft Learn module titled "Evaluate Boolean expressions to make decisions in C#".

In this example, you evaluated the Boolean expression `total > 14`. However, you could have chosen the Boolean expression `total >= 15` because in this case, they're the same. Given that the rules to the game specify "If the sum of the three dice, plus any bonuses, is 15 or greater, you win the game", you should probably implement the `>= 15` expression. You'll make that change in the next step of the exercise.

What is a code block?

A code block is a collection of one or more lines of code that are defined by an opening and closing curly brace symbol `{ }`. It represents a complete unit of code that has a single purpose in your software system. In this case, at runtime, all lines of code in the code block are executed if the Boolean expression is true. Conversely, if the Boolean expression is false, all lines of code in the code block are ignored.

You should also know that code blocks can contain other code blocks. In fact, it's common for one code block to be "nested" inside another code block in your applications. You'll begin nesting

your own code blocks later in this module when you create one `if` statement inside the code block of another.

> **Note**
>
> The C# training series on Microsoft Learn devotes an entire module to understanding code blocks. Code blocks are central to understanding code organization and structure, and they define the boundaries of variable scope. See the module `[Control variable scope and logic using code blocks in C#](TBD)`.

Add another if statement to implement the doubles bonus

Next, you can implement the rule: "If any two dice you roll result in the same value, you get two bonus points for rolling doubles". Modify the code from the previous step to match the following code listing:

1. In the Visual Studio Code Editor, locate the cursor on the blank code line above the first `if` statement.

2. To create your "doubles" game feature, enter the following `if` statement.

   ```
   c#

   if ((roll1 == roll2) || (roll2 == roll3) || (roll1 == roll3))
   {
       Console.WriteLine("You rolled doubles! +2 bonus to total!");
       total += 2;
   }
   ```

   Here you combine three Boolean expressions to create one composite Boolean expression in a single line of code. This is sometimes called a *compound condition*. You have one outer set of parentheses that combines three inner sets of parentheses separated by two pipe characters.

   The double pipe characters `||` are the **logical OR** operator, which basically says "either the expression to my left OR the expression to my right must be true in order for the entire Boolean expression to be true". If both Boolean expressions are false, then the entire

Boolean expression is false. You use two logical OR operators so that you can extend the evaluation to a third Boolean expression.

First, you evaluate `(roll1 == roll2)`. If that's true, then the entire expression is true. If it's false, you evaluate `(roll2 == roll3)`. If that's true, then the entire expression is true. If it's false, you evaluate `(roll1 == roll3)`. If that's true, then the entire expression is true. If that is false, then the entire expression is false.

If the composite Boolean expression is true, then you execute the following code block. This time, there are two lines of code. The first line of code prints a message to the user. The second line of code increments the value of `total` by `2`.

3. To improve the readability of your code, update the second `if` statement as follows:

```
c#

if (total >= 15)
```

Notice that you're now using the `>=` operator in the expression that's used to evaluate a winning roll. The `>=` operator means "greater or equal to". As a result, you can compare `total` to a value of `15` rather than `14`. With these changes, the expression that you use to evaluate a winning roll now resembles the expression that you evaluate for a losing roll. This should help to make your code easier to understand (more readable). Since you are dealing with integer values, your new expression `(total >= 15)` will function identically to what you wrote previously `(total > 14)`.

4. Take a minute to review your code.

Your code should match the following:

```
c#

Random dice = new Random();

int roll1 = dice.Next(1, 7);
int roll2 = dice.Next(1, 7);
int roll3 = dice.Next(1, 7);

int total = roll1 + roll2 + roll3;

Console.WriteLine($"Dice roll: {roll1} + {roll2} + {roll3} = {total}");

if ((roll1 == roll2) || (roll2 == roll3) || (roll1 == roll3))
```

```
{
    Console.WriteLine("You rolled doubles! +2 bonus to total!");
    total += 2;
}

if (total >= 15)
{
    Console.WriteLine("You win!");
}

if (total < 15)
{
    Console.WriteLine("Sorry, you lose.");
}
```

Notice the improved alignment between the expressions used to evaluate winning and losing rolls.

Add another if statement to implement the triples bonus

Next, you can implement the rule: "If all three dice you roll result in the same value, you get six bonus points for rolling triples." Modify the code from the previous steps to match the following code listing:

1. In the Visual Studio Code Editor, create a blank code line below the code block of your "doubles" `if` statement.

2. To create your "triples" game feature, enter the following `if` statement.

```
c#

if ((roll1 == roll2) && (roll2 == roll3))
{
    Console.WriteLine("You rolled triples! +6 bonus to total!");
    total += 6;
}
```

Here you combine two Boolean expressions to create one composite Boolean expression in a single line of code. You have one outer set of parentheses that combines two inner sets of parentheses separated by two ampersand characters.

The double ampersand characters `&&` are the **logical AND** operator, which basically says "only if both expressions are true, then the entire expression is true". In this case, if `roll1` is equal to `roll2`, and `roll2` is equal to `roll3`, then by deduction, `roll1` must be equal to `roll3`, and the user rolled triples.

3. On the Visual Studio Code **File** menu, click **Save**.

4. Take a minute to review your code.

   Ensure that your code matches the following:

```
c#
```

```
Random dice = new Random();

int roll1 = dice.Next(1, 7);
int roll2 = dice.Next(1, 7);
int roll3 = dice.Next(1, 7);

int total = roll1 + roll2 + roll3;

Console.WriteLine($"Dice roll: {roll1} + {roll2} + {roll3} = {total}");

if ((roll1 == roll2) || (roll2 == roll3) || (roll1 == roll3))
{
    Console.WriteLine("You rolled doubles! +2 bonus to total!");
    total += 2;
}

if ((roll1 == roll2) && (roll2 == roll3))
{
    Console.WriteLine("You rolled triples! +6 bonus to total!");
    total += 6;
}

if (total >= 15)
{
    Console.WriteLine("You win!");
}

if (total < 15)
{
    Console.WriteLine("Sorry, you lose.");
}
```

5. In the EXPLORER panel, to open a Terminal at your TestProject folder location, right-click **TestProject**, and then select **Open in Integrated Terminal**.

   A Terminal panel should open, and should include a command prompt showing that the Terminal is open to your TestProject folder location.

6. At the Terminal command prompt, to run your code, type **dotnet run** and then press Enter.

   > **Note**
   >
   > If you see a message saying "Couldn't find a project to run", ensure that the Terminal command prompt displays the expected TestProject folder location. For example: `C:\Users\someuser\Desktop\csharpprojects\TestProject>`

   You should see output that resembles one of the following results:

   Output

   ```
   Dice roll: 3 + 6 + 1 = 10
   Sorry, you lose.
   ```

   Or, like this:

   Output

   ```
   Dice roll: 1 + 4 + 4 = 9
   You rolled doubles! +2 bonus to total!
   Sorry, you lose.
   ```

   Or, like this:

   Output

   ```
   Dice roll: 5 + 6 + 4 = 15
   You win!
   ```

   Or, if you're lucky, you'll see this:

   Output

```
Dice roll: 6 + 6 + 6 = 18
You rolled doubles! +2 bonus to total!
You rolled triples! +6 bonus to total!
You win!
```

But wait, should you really reward the player with both the triple bonus and the double bonus? After all, a roll of triples implies that they also rolled doubles. Ideally, the bonuses shouldn't *stack*. There should be two separate bonus conditions. This is a bug in logic that will need to be corrected.

Problems in your logic and opportunities to improve the code

Although this is a good start, and you've learned a lot about the `if` statement, Boolean expressions, code blocks, logical OR and AND operators, and so on, there's much that can be improved. You'll do that in the next unit.

Recap

- Use an `if` statement to branch your code logic. The `if` decision statement will execute code in its code block if its Boolean expression equates to true. Otherwise, the runtime will skip over the code block and continue to the next line of code after the code block.
- A Boolean expression is any expression that returns a Boolean value.
- Boolean operators will compare the two values on its left and right for equality, comparison, and more.
- A code block is defined by curly braces `{ }`. It collects lines of code that should be treated as a single unit.
- The logical AND operator `&&` aggregates two expressions so that both subexpressions must be true in order for the entire expression to be true.
- The logical OR operator `||` aggregates two expressions so that if either subexpression is true, the entire expression is true.

Check your knowledge

1. What is a code block? *

- ◯ A .NET Class Library.
- ◯ Lines of code that should be treated as a single unit.
- ◯ A block of code that is blocked from being accessed.

2. What is a Boolean statement or expression? *

- ◯ A modulus expression.
- ◯ An ordinal term.
- ◯ Code that returns either `true` or `false`.

Check your answers

Module complete:

Unlock achievement