100 XP

Exercise - Create nested decision logic with if, else if, and else

14 minutes

In the previous unit, you used multiple `if` statements to implement the rules of a game. However, at the end of the unit, you noticed that more expressive `if` statements are needed to fix a subtle bug in your code.

In this exercise, you'll use `if`, `else`, and `else if` statements to improve the branching options in your code and fix a logic bug.

Use if and else statements instead of two separate if statements

Instead of performing two checks to display the "You win!" or "Sorry, you lose" message, you'll use the `else` keyword.

1. Ensure that your Program.cs 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.");
}
```

This is the code that you completed in the previous unit.

2. Take a minute to examine the two `if` statements at the end of the file:

```
c#

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

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

Notice that both `if` statements compare `total` with the same numeric value. This is the perfect opportunity to use an `else` statement.

3. Update the two `if` statements as follows:

```
c#

if (total >= 15)
{
    Console.WriteLine("You win!");
}
else
{
    Console.WriteLine("Sorry, you lose.");
```

```
}
```

Here, if `total >= 15` is false, then the code block following the `else` keyword will execute. Since the two outcomes are related opposites, this is a perfect scenario for the `else` keyword.

4. Your updated Program.cs file should contain the following code:

```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!");
}
else
{
    Console.WriteLine("Sorry, you lose.");
}
```

Modify the code to remove the stacking bonus for doubles and triples using nesting

In the previous unit, you saw that a subtle logic bug was introduced into your application. You can fix that issue by nesting your `if` statements.

Nesting allows you to place code blocks inside of code blocks. In this case, you'll nest an `if` and `else` combination (the check for doubles) inside of another `if` statement (the check for triples) to prevent both bonuses from being awarded.

1. Modify your code to match the following code listing:

```
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))
{
    if ((roll1 == roll2) && (roll2 == roll3))
    {
        Console.WriteLine("You rolled triples!  +6 bonus to total!");
        total += 6;
    }
    else
    {
        Console.WriteLine("You rolled doubles!  +2 bonus to total!");
        total += 2;
    }
}

if (total >= 15)
{
    Console.WriteLine("You win!");
}
else
{
    Console.WriteLine("Sorry, you lose.");
}
```

2. Take a minute to review the nested `if` statements.

The goal is to create an inner `if-else` construct where the two outcomes are related opposites, and then use the opposing outcomes (if/true and else/false) to award the bonus points for triples and doubles. To achieve this goal, you check for doubles in the outer `if` statement, and then for triples in the inner `if` statement. This pattern ensures that when the inner check for triples returns `false`, your `else` code block can award the points for doubles.

Coming up, you will "hard code" the results of your three rolls in order to test your code logic.

3. Create a blank code line above the line where `total` is declared and initialized.

4. To test for a roll of doubles, enter the following code:

```
c#

roll1 = 6;
roll2 = 6;
roll3 = 5;
```

Hard coding the three `roll` variables enables you to test the code without having to run the application dozens of times.

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

6. 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.

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

When your code runs, you should see:

```
Output

Dice roll: 6 + 6 + 5 = 17
You rolled doubles!  +2 bonus to total!
You win!
```

8. To test for a roll of triples, update your hard-coded roll variables as follows:

```
c#

roll1 = 6;
roll2 = 6;
roll3 = 6;
```

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

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

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

    When your code runs, you should see:

```
Output

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

Use if, else, and else if statements to give a prize instead of a win-lose message

To make the game more fun, you can change the game from "win-or-lose" to awarding fictitious prizes for each score. You can offer four prizes. However, the player should win only one prize:

- If the player scores greater or equal to 16, they'll win a new car.
- If the player scores greater or equal to 10, they'll win a new laptop.
- If the player scores exactly 7, they'll win a trip.
- Otherwise, the player wins a kitten.

1. Modify the code from the previous steps to the following code listing:

```
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))
{
    if ((roll1 == roll2) && (roll2 == roll3))
    {
        Console.WriteLine("You rolled triples!  +6 bonus to total!");
        total += 6;
    }
    else
    {
        Console.WriteLine("You rolled doubles!  +2 bonus to total!");
        total += 2;
    }
}

if (total >= 16)
{
    Console.WriteLine("You win a new car!");
}
else if (total >= 10)
{
    Console.WriteLine("You win a new laptop!");
}
else if (total == 7)
{
    Console.WriteLine("You win a trip for two!");
}
else
{
    Console.WriteLine("You win a kitten!");
}
```

2. Take a minute to review the updated `if-elseif-else` construct.

   The `if`, `else if`, and `else` statements allow you to create multiple exclusive conditions as Boolean expressions. In other words, when you only want one outcome to happen, but you have several possible conditions and results, use as many `else if` statements as you want. If none of the `if` and `else if` statements apply, the final `else` code block will be executed. The `else` is optional, but it must come last if you choose to include it.

3. Use the technique of temporarily hard coding the `roll` variables to test each message.

Recap

- The combination of `if` and `else` statements allows you to test for one condition, and then perform one of two outcomes. The code block for the `if` will be run when the Boolean expression is `true`, and the code block for the `else` will be run when the Boolean expression is `false`.
- You can nest `if` statements to narrow down a possible condition. However, you should consider using the `if`, `else if`, and `else` statements instead.
- Use `else if` statements to create multiple exclusive conditions.
- An `else` is optional, but it must always come last when included.

Module complete:

Unlock achievement