100 XP

Exercise - Use whitespace to make your code easier to read

10 minutes

Print and web designers understand that putting too much information in a small space overwhelms the viewer. So, they strategically use whitespace, or negative space, to break up information to maximize the viewer's ability to consume the primary message of their work.

Developers can use a similar strategy when writing code in an editor. By using white space to convey meaning, developers can increase the clarity of their code's intent.

What is whitespace?

The term "whitespace" refers to individual spaces produced by the `space bar`, tabs produced by the `tab` key, and new lines produced by the `enter` key.

The C# compiler ignores whitespace. To understand how whitespace is ignored, and how to maximize clarity using white space, work through the following exercise.

Add code to illustrate how whitespace is ignored by the C# compiler

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

   If necessary, open Visual Studio Code, and then complete the following steps to prepare a Program.cs file in the Editor:

   a. On the **File** menu, select **Open Folder**.

   b. Use the Open Folder dialog to navigate to, and then open, the **CsharpProjects** folder.

   c. In the Visual Studio Code EXPLORER panel, select **Program.cs**.

   d. On the Visual Studio Code **Selection** menu, select **Select All**, and then press the Delete key.

2. Enter the following code:

```c#
// Example 1:
Console
.
WriteLine
(
"Hello Example 1!"
)
;

// Example 2:
string firstWord="Hello";string lastWord="Example
2";Console.WriteLine(firstWord+" "+lastWord+"!");
```

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

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

5. At the Terminal command prompt, type **dotnet run** and then press Enter.

   You should see the following output:

```Output
Hello Example 1!
Hello Example 2!
```

6. Take a minute to consider what this result tells you about how you should use whitespace in your code.

   These two code examples illustrate two vital ideas:

   - Whitespace doesn't matter to the compiler. However ...
   - Whitespace, when used properly, can increase your ability to read and comprehend the code.

   You likely write your code once, but need to read the code multiple times. Therefore, you should focus on the readability of the code you write. Over time, you'll get a feel for when and how to use whitespace, such as the space character, tabs, and new lines.

Early guidance:

- Each complete command (a *statement*) belongs on a separate line.
- If a single line of code becomes long, you can break it up. However, you should avoid arbitrarily splitting up a single statement to multiple lines until you have a good reason to do so.
- Use a space to the left and right of the assignment operator.

7. Replace your existing code with 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)) {
    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;
    }
}
```

Notice that this code doesn't include much whitespace. This code will be used to illustrate an approach for adding whitespace to your applications. Effective whitespace should make it easier to understand what your code is doing.

> **Note**
>
> The code uses the `Random` class to help develop a simulated dice game, where the total value from three rolls is used to evaluate a "winning" score. The code awards extra points for rolling doubles or triples. You don't need to fully understand this code in order to see the benefit of including whitespace.

8. Take a minute to consider how you would use whitespace to improve the readability of this code.

There are two features of this code to take note of:

- There's no vertical whitespace in this code example. In other words, there's no empty lines separating the lines of code. It all runs together into one dense code listing.
- The code blocks as defined by the opening and closing curly brace symbols `{ }` are compressed together, making their boundaries difficult to visually discern.

Generally speaking, to improve readability, you introduce a blank line between two, three, or four lines of code that do similar or related things.

Phrasing your code using vertical whitespace is subjective. It's possible two developers won't agree on what is most readable, or when to add whitespace. Use your best judgment.

9. To add vertical whitespace that improves readability, update your code as follows:

```
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;
    }
}
```

Your first line of whitespace is used to separate the declaration of the `dice` variable from the code lines used to assign values to your roll variables. This separation makes it easier to see how `dice` is being used in your code.

Your next line of whitespace separates the declaration of your roll variables from the declaration of `total`. Grouping the declaration of your three roll variables is helpful in two ways. First, it creates a group of code lines that includes related variables. Second, the variable names are so similar and the declaration follows the same pattern. So, grouping them together draws your eye to the similarities and helps to expose the differences.

Finally, your third line of whitespace separates another group of related statements from your nested `if` statements. The group of statements that includes the declaration of `total` and the `Console.WriteLine()` method is related by purpose rather than appearance. Your code is focused on the total value achieved by the three dice and whether the roll included doubles or triples. These lines are related because you need to calculate `total` and report the results of the roll to the user.

Some developers might argue that you should add an empty line in between the declaration of `total` and the `Console.WriteLine()`. Again, the choice of whitespace is up to your best judgment. You should decide which is more readable for you and use that style consistently.

All that you have left is the `if` statement. You can examine that now.

10. Focusing on the lines of code below the `if` keyword, modify your code as follows:

```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;
    }
}
```

11. Notice that you've moved the opening and closing curly braces to their own line to improve spacing.

    The `{` and `}` symbols create code blocks. Many C# constructs require code blocks. These symbols should be placed on a separate line so that their boundaries are clearly visible and

readable.

Furthermore, it's important to use the `tab` key to line up the code block symbols under the keyword they belong to. For example, notice the line of code that starts with the keyword `if`. Below that line is the `{` symbol. This alignment makes it easy to understand that the `{` "belongs to" the `if` statement. Furthermore, the last `}` symbol lines up with the `if` statement as well. The combination of alignment and indentation makes it easy to understand where the code block begins and ends.

The code lines inside of this code block are indented, indicating that they "belong" to this code block.

You follow a similar pattern with the inner `if` statement and `else` statement, and the code inside of those code blocks.

Not everyone agrees with this style guidance for including whitespace. However, you should consider using this guidance as a starting point when writing code. In the future, you can be purposeful when making a decision to deviate from this guidance.

Recap

The main takeaways from this exercise:

- Use whitespace judiciously to improve the readability of your code.
- Use line feeds to create empty lines to separate phrases of code. A phrase includes lines of code that are similar, or work together.
- Use line feeds to separate code block symbols so that they are on their own line of code.
- Use the `tab` key to line up a code block with the keyword they're associated with.
- Indent code inside of a code block to show ownership.

Module complete:

Unlock achievement