100 XP

Exercise - Combine strings using character escape sequences

10 minutes

Suppose you've been asked to create a mockup of a command-line tool that will generate invoices in both English and Japanese. You don't have to build the actual functionality that generates the invoices yet. You only need to provide the command line interface to internal customers in the billing department for their approval. Your manager asked you to make sure you add formatting to make the current progress of the tool clear. Your manager also asked you to provide instructions for the Japanese users on how to generate invoices in Japanese.

Exercise - Format literal strings in C#

In this exercise, you'll learn different techniques to display special characters and add different types of formatting to the output.

Character escape sequences

An **escape character sequence** is an instruction to the runtime to insert a special character that will affect the output of your string. In C#, the escape character sequence begins with a backslash `\` followed by the character you're escaping. For example, the `\n` sequence will add a new line, and a `\t` sequence will add a tab.

The following code uses escape character sequences to add newlines and tabs:

C#

```
Console.WriteLine("Hello\nWorld!");
Console.WriteLine("Hello\tWorld!");
```

If you run the code, you'll see the following output:

Output

```
Hello
World!
Hello   World!
```

What if you need to insert a double-quotation mark in a literal string? If you don't use the character escape sequence, you'll confuse the compiler because it will think you want to terminate the string prematurely. The compiler won't understand the purpose of the characters after the second double-quotation mark.

C#

```
Console.WriteLine("Hello "World"!");
```

The .NET Editor will put a red squiggly line under `World`. But if you attempt to run the code anyway, you would see the following output:

```
(1,27): error CS1003: Syntax error, ',' expected
(1,27): error CS0103: The name 'World' does not exist in the current context
(1,32): error CS1003: Syntax error, ',' expected
```

To handle that situation, use the `\"` escape sequence:

C#

```
Console.WriteLine("Hello \"World\"!");
```

If you run the code above, you would see the following output:

Output

```
Hello "World"!
```

What if you need to use the backslash for other purposes, like to display a file path?

C#

```
Console.WriteLine("c:\source\repos");
```

Unfortunately, C# reserves the backslash for escape sequences, so if you run the code, the compiler will display the following error:

Output

```
(1,22): error CS1009: Unrecognized escape sequence
```

The problem is the sequence `\s`. The `\r` doesn't produce an error because it's a valid escape sequence for a carriage return. However, you don't want to use a carriage return in this context.

To solve this problem, you use the `\\` to display a single backslash.

C#

```
Console.WriteLine("c:\\source\\repos");
```

Escaping the back slash character produces the output you intended:

Output

```
c:\source\repos
```

Format output using character escape sequences

1. Select all of the code in the .NET Editor, and press `Delete` or `Backspace` to delete it.
2. To create the mockup of the command line tool, enter the following code in the editor:

   C#

   ```
   Console.WriteLine("Generating invoices for customer \"Contoso Corp\" ...\n");
   Console.WriteLine("Invoice: 1021\t\tComplete!");
   Console.WriteLine("Invoice: 1022\t\tComplete!");
   Console.WriteLine("\nOutput Directory:\t");
   ```

3. Now, run the code. You'll see the following result in the output console:

   Output

   ```
   Generating invoices for customer "Contoso Corp" ...

   Invoice: 1021           Complete!
   ```

```
Invoice: 1022           Complete!

Output Directory:
```

Verbatim string literal

A verbatim string literal will keep all whitespace and characters without the need to escape the backslash. To create a verbatim string, use the `@` directive before the literal string.

C#

```
Console.WriteLine(@"    c:\source\repos
        (this is where your code goes)");
```

Notice that the string spans two lines and the whitespace generated by this C# instruction is kept in the following output.

Output

```
    c:\source\repos
        (this is where your code goes)
```

Format output using verbatim string literals

1. Add the following line of code beneath the code that you created previously:

   C#

   ```
   Console.Write(@"c:\invoices");
   ```

2. Now, run the code. You'll see the following result that includes the "Output Directory":

   Output

   ```
   Generating invoices for customer "Contoso Corp" ...

   Invoice: 1021           Complete!
   Invoice: 1022           Complete!
   ```

```
Output Directory:
c:\invoices
```

Unicode escape characters

You can also add encoded characters in literal strings using the `\u` escape sequence, then a four-character code representing some character in Unicode (UTF-16).

C#

```
// Kon'nichiwa World
Console.WriteLine("\u3053\u3093\u306B\u3061\u306F World!");
```

> **Note**
>
> There are several caveats here. First, some consoles like the Windows Command Prompt will not display all Unicode characters. It will replace those characters with question mark characters instead. Also, the examples used here are UTF-16. Some characters require UTF-32 and therefore require a different escape sequence. This is a complicated subject, and this module is only aiming at showing you what is possible. Depending on your need, you may need to spend quite a bit of time learning and working with Unicode characters in your applications.

Format output using unicode escape characters

To complete the mock-up of the command-line tool, you'll add a phrase in Japanese that translates: "To generate Japanese invoices". Then you'll display a verbatim literal string that represents a command the user can enter. You'll also add some escape sequences for formatting.

1. Add the following code to your application:

C#

```
// To generate Japanese invoices:
// Nihon no seikyū-sho o seisei suru ni wa:
Console.Write("\n\n\u65e5\u672c\u306e\u8acb\u6c42\u66f8\u3092\u751f\u6210\u3059\
u308b\u306b\u306f\uff1a\n\t");
```

```
// User command to run an application
Console.WriteLine(@"c:\invoices\app.exe -j");
```

2. To ensure your code is correct, compare it with the following:

C#

```
Console.WriteLine("Generating invoices for customer \"Contoso Corp\" ...\n");
Console.WriteLine("Invoice: 1021\t\tComplete!");
Console.WriteLine("Invoice: 1022\t\tComplete!");
Console.WriteLine("\nOutput Directory:\t");
Console.Write(@"c:\invoices");

// To generate Japanese invoices:
// Nihon no seikyū-sho o seisei suru ni wa:
Console.Write("\n\n\u65e5\u672c\u306e\u8acb\u6c42\u66f8\u3092\u751f\u6210\u3059\
u308b\u306b\u306f\uff1a\n\t");
// User command to run an application
Console.WriteLine(@"c:\invoices\app.exe -j");
```

3. Now, run the code. You'll see the following result in the output console:

Output

```
Generating invoices for customer "Contoso Corp" ...

Invoice: 1021           Complete!
Invoice: 1022           Complete!

Output Directory:
c:\invoices

日本の請求書を生成するには：
    c:\invoices\app.exe -j
```

Recap

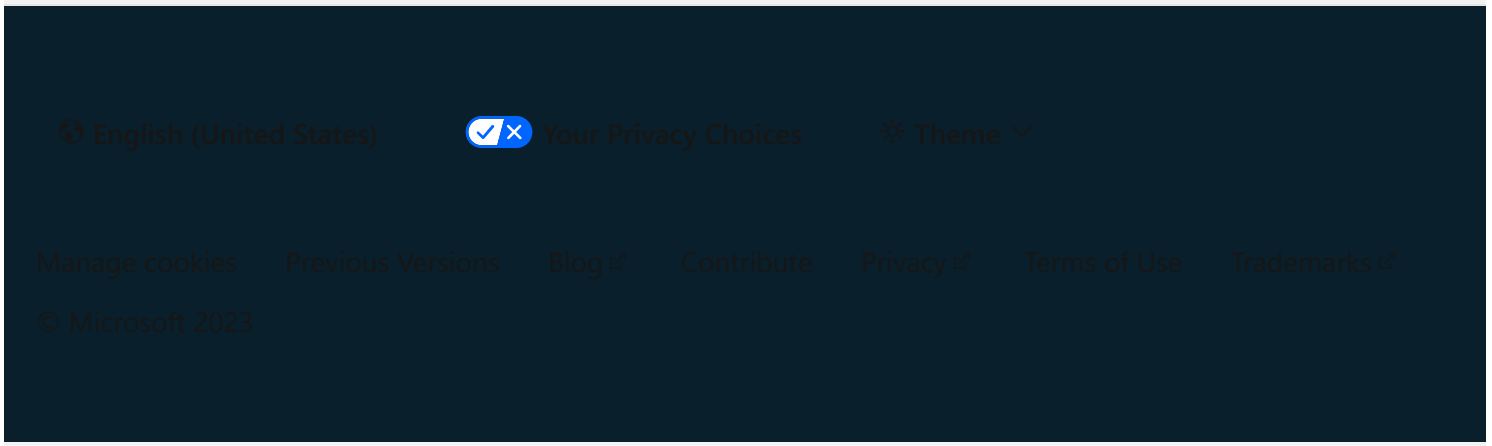
Here's what you've learned about formatting literal strings so far:

- Use character escape sequences when you need to insert a special character into a literal string, like a tab `\t`, new line `\n`, or a double quotation mark `\"`.
- Use an escape character for the backslash `\\` when you need to use a backslash in all other scenarios.

- Use the `@` directive to create a verbatim string literal that keeps all whitespace formatting and backslash characters in a string.
- Use the `\u` plus a four-character code to represent Unicode characters (UTF-16) in a string.
- Unicode characters may not print correctly depending on the application.

Module complete:

Unlock achievement



.NET Editor

Press CTRL + M, TAB to exit the editor

Clear Run

```
1
```

Output

☺