One minor gotcha regarding named arguments is that if you begin to invoke a method using

positional parameters, they must be listed before any named parameters. In other words, **named arguments must always be packed onto the end of a method call.**

static void DisplayFancyMessage(ConsoleColor textColor, ConsoleColor backgroundColor, string message)

{

// Store old colors to restore after message is printed.

ConsoleColor oldTextColor = Console.ForegroundColor;

ConsoleColor oldbackgroundColor = Console.BackgroundColor;

// Set new colors and print message.

Console.ForegroundColor = textColor;

Console.BackgroundColor = backgroundColor;

Console.WriteLine(message);

// Restore previous colors.

Console.ForegroundColor = oldTextColor;

Console.BackgroundColor = oldbackgroundColor;

}

static void Main(string[] args)

{

Console.WriteLine("\*\*\*\*\* Fun with Methods \*\*\*\*\*");

...

DisplayFancyMessage(message: "Wow! Very Fancy indeed!",

textColor: ConsoleColor.DarkRed,

backgroundColor: ConsoleColor.White);

DisplayFancyMessage(backgroundColor: ConsoleColor.Green,

message: "Testing...",

textColor: ConsoleColor.DarkBlue);

Console.ReadLine();

}

// This is OK, as positional args are listed before named args.

DisplayFancyMessage(ConsoleColor.Blue,

message: "Testing...",

backgroundColor: ConsoleColor.White);

// This is an ERROR, as positional args are listed after named args.

DisplayFancyMessage(message: "Testing...",

backgroundColor: ConsoleColor.White,

ConsoleColor.Blue);

This restriction aside, you might still be wondering when you would ever want to use this language feature. After all, if you need to specify three arguments to a method, why bother flipping around their position?

Well, as it turns out, if you have a method that defines optional arguments, this feature can actually be really helpful. Assume DisplayFancyMessage() has been rewritten to now support optional arguments, as you have assigned fitting defaults:

static void DisplayFancyMessage(ConsoleColor textColor = ConsoleColor.Blue,

ConsoleColor backgroundColor = ConsoleColor.White,

string message = "Test Message")

{

...

}

Given that each argument has a default value, named arguments allow the caller to specify only the parameter(s) for which they do not wish to receive the defaults. Therefore, if the caller wants the value

"Hello!" to appear in blue text surrounded by a white background, they can simply specify:

DisplayFancyMessage(message: "Hello!");