Key Features review

6

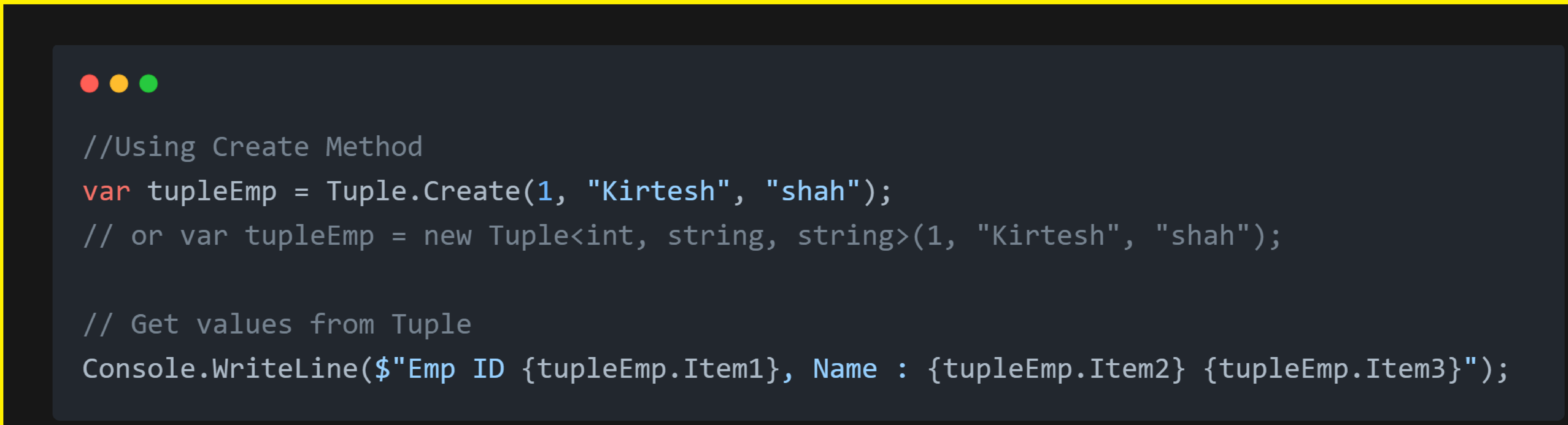
C# 7

8

9

Nabi Karampoor
@thisisnabi

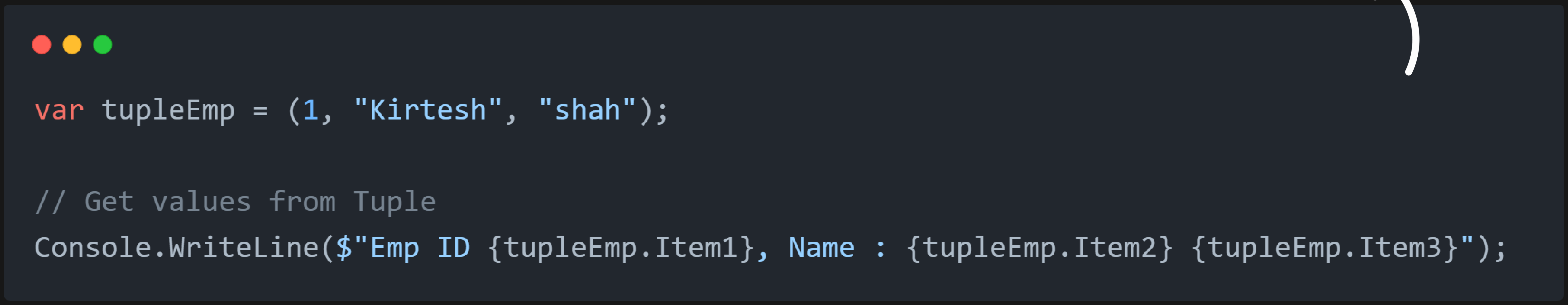
1 Tuples Improvement



```
//Using Create Method
var tupleEmp = Tuple.Create(1, "Kirtesh", "shah");
// or var tupleEmp = new Tuple<int, string, string>(1, "Kirtesh", "shah");

// Get values from Tuple
Console.WriteLine($"Emp ID {tupleEmp.Item1}, Name : {tupleEmp.Item2} {tupleEmp.Item3}");
```

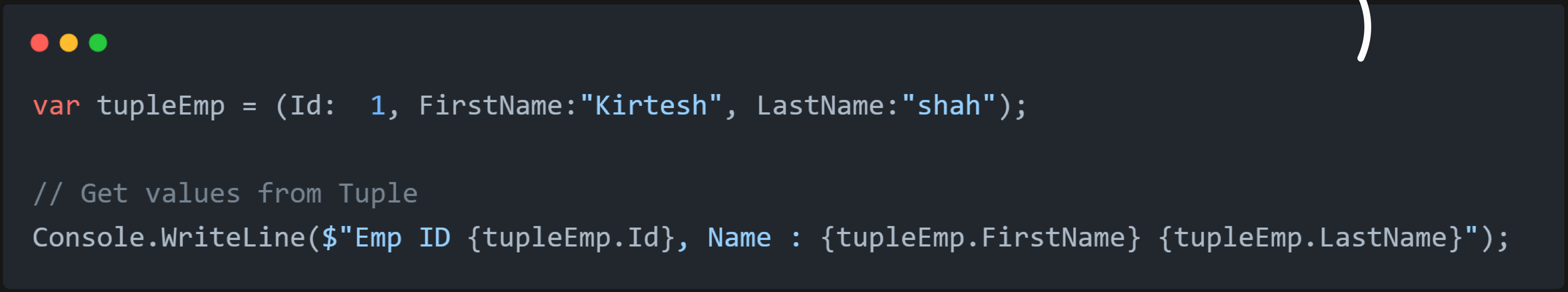
It’s not strongly typed and needs to access the value using the above method.

Value Tuple, C# 7



```
var tupleEmp = (1, "Kirtesh", "shah");

// Get values from Tuple
Console.WriteLine($"Emp ID {tupleEmp.Item1}, Name : {tupleEmp.Item2} {tupleEmp.Item3}");
```

Named Tuple, C# 7.1



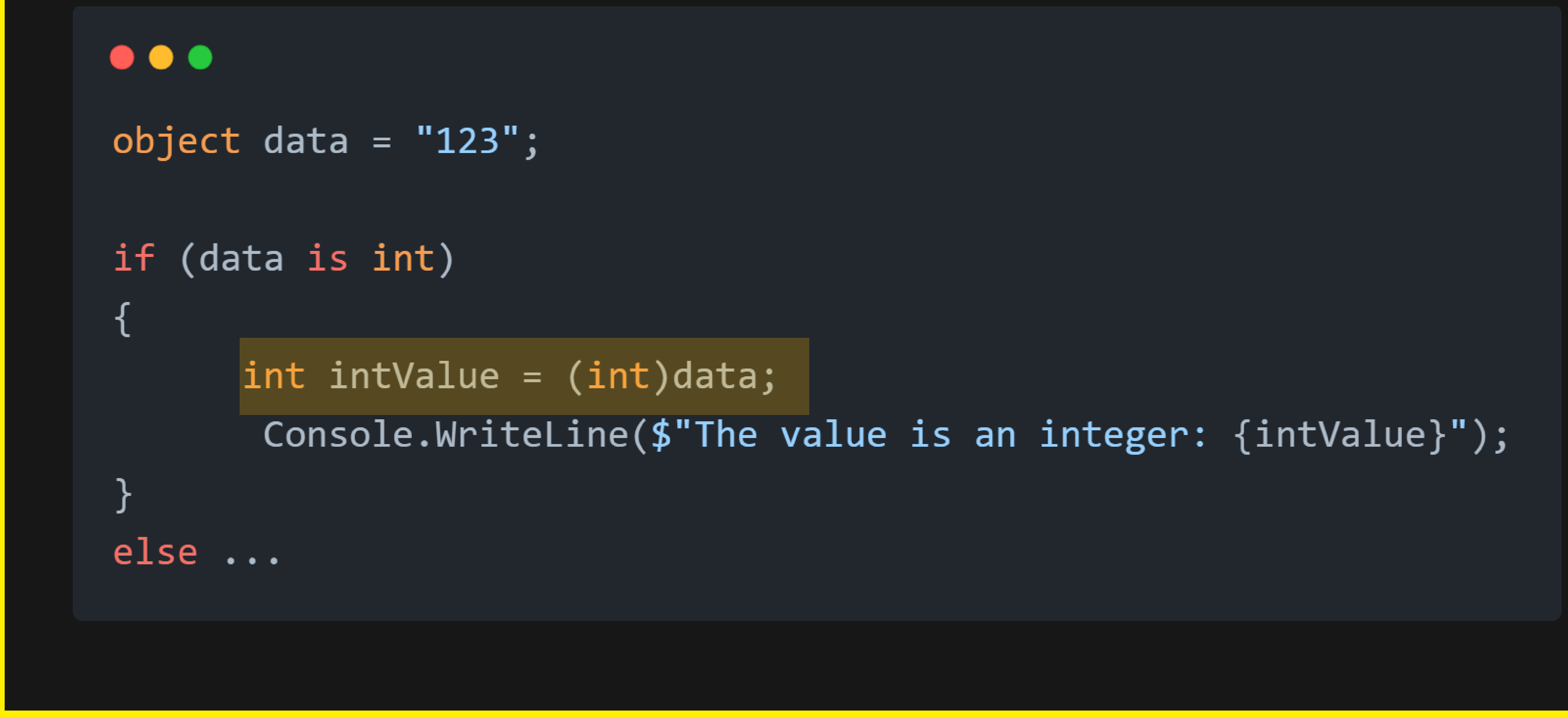
```
var tupleEmp = (Id:  1, FirstName:"Kirtesh", LastName:"shah");

// Get values from Tuple
Console.WriteLine($"Emp ID {tupleEmp.Id}, Name : {tupleEmp.FirstName} {tupleEmp.LastName}");
```

A simple way to return multiple values from a method without declaring a new type.

Nabi Karampoor
@thisisnabi

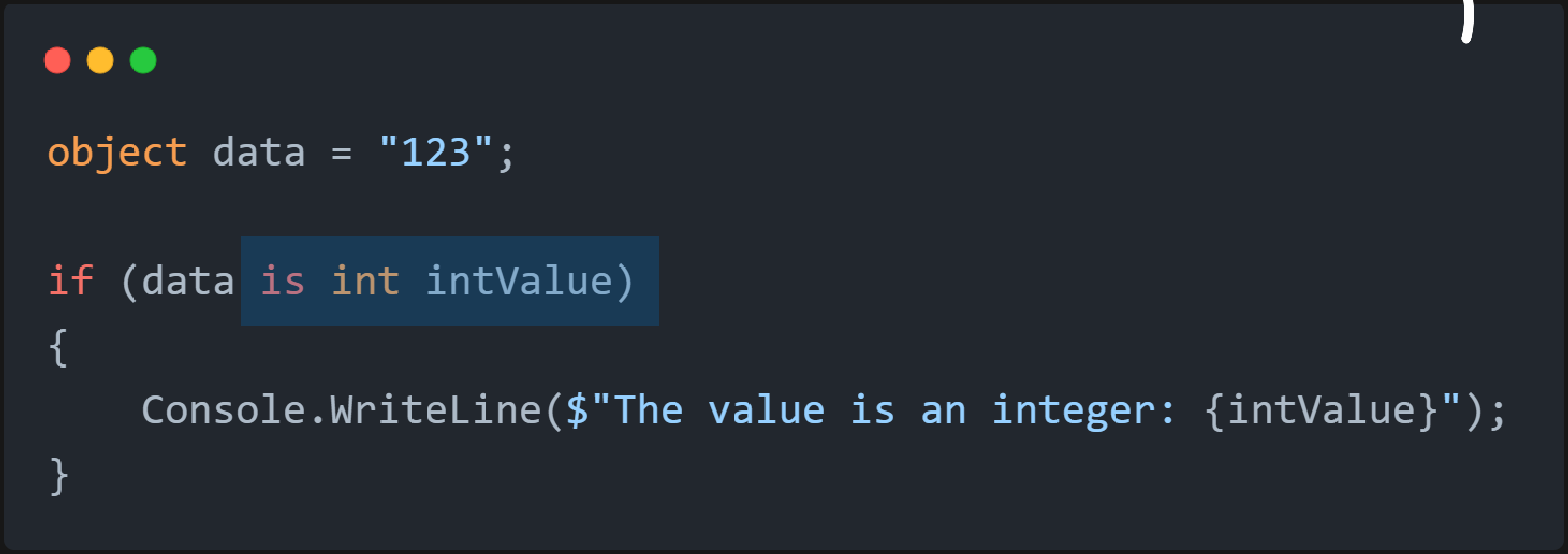
2 Pattern Matching

Do extra work after checking!



```
object data = "123";

if (data is int)
{
       int intValue = (int)data;
        Console.WriteLine($"The value is an integer: {intValue}");
}
else ...
```

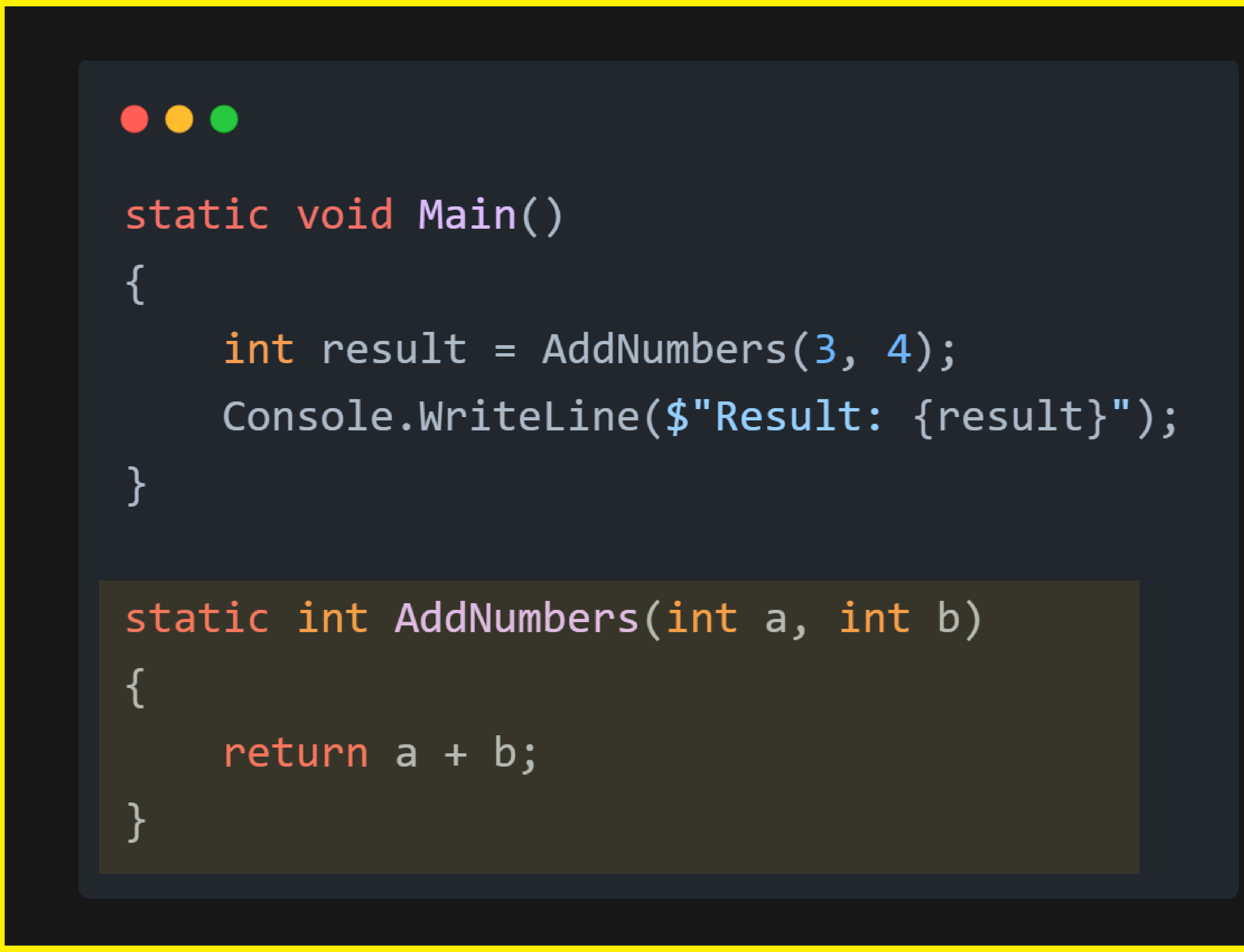
is keyword is used for pattern matching.
If data is int, it is assigned to the intValue.



```
object data = "123";

if (data is int intValue)
{
    Console.WriteLine($"The value is an integer: {intValue}");
}
```

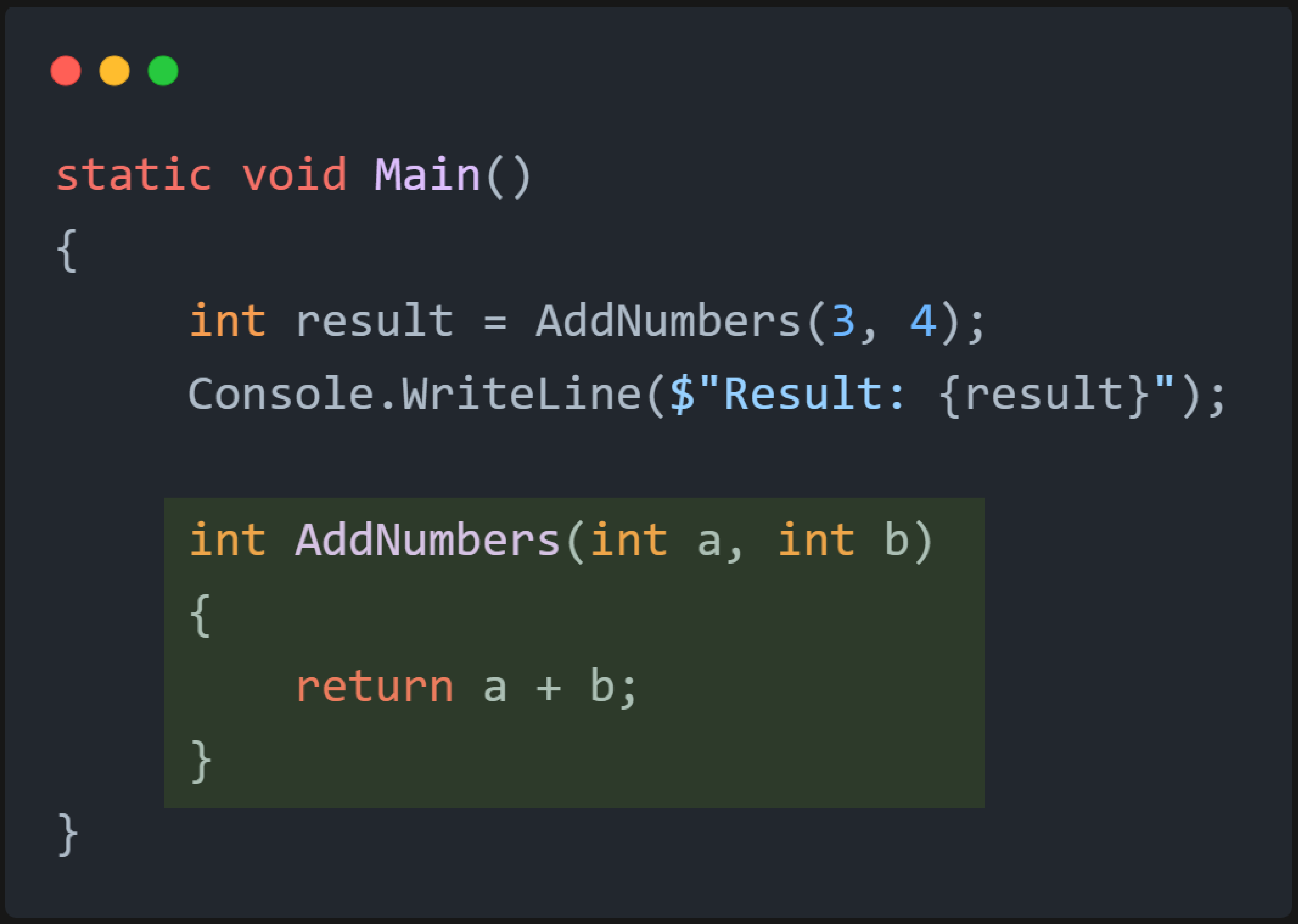
Nabi Karampoor
@thisisnabi

3 Local Functions



```
static void Main()
{
    int result = AddNumbers(3, 4);
    Console.WriteLine($"Result: {result}");
}

static int AddNumbers(int a, int b)
{
    return a + b;
}
```

It can be accessed by other methods in this class.

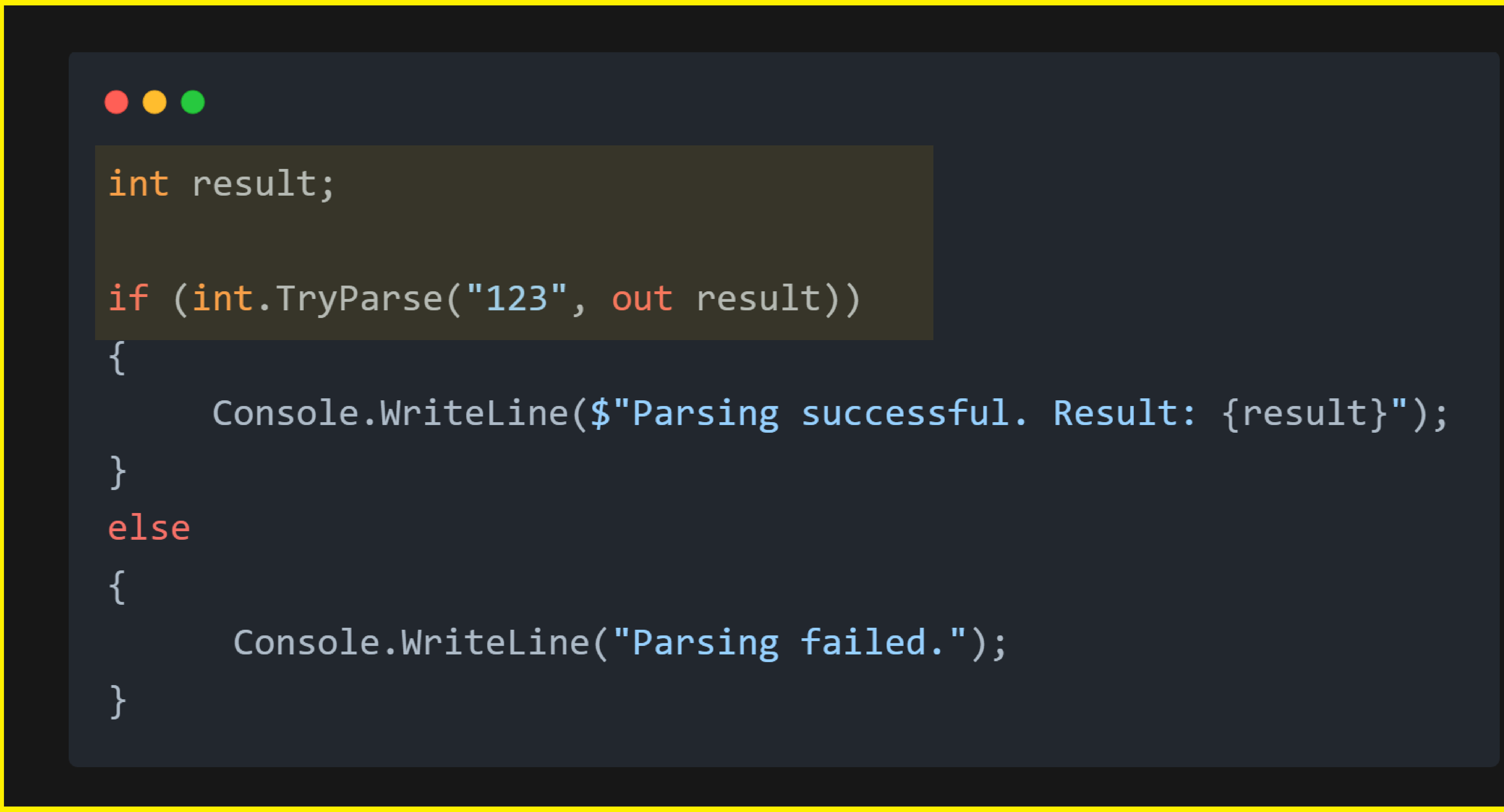


```
static void Main()
{
     int result = AddNumbers(3, 4);
     Console.WriteLine($"Result: {result}");

     int AddNumbers(int a, int b)
     {
         return a + b;
     }
}
```

Short, nested helper functions within a method for encapsulation and readability.

Nabi Karampoor
@thisisnabi

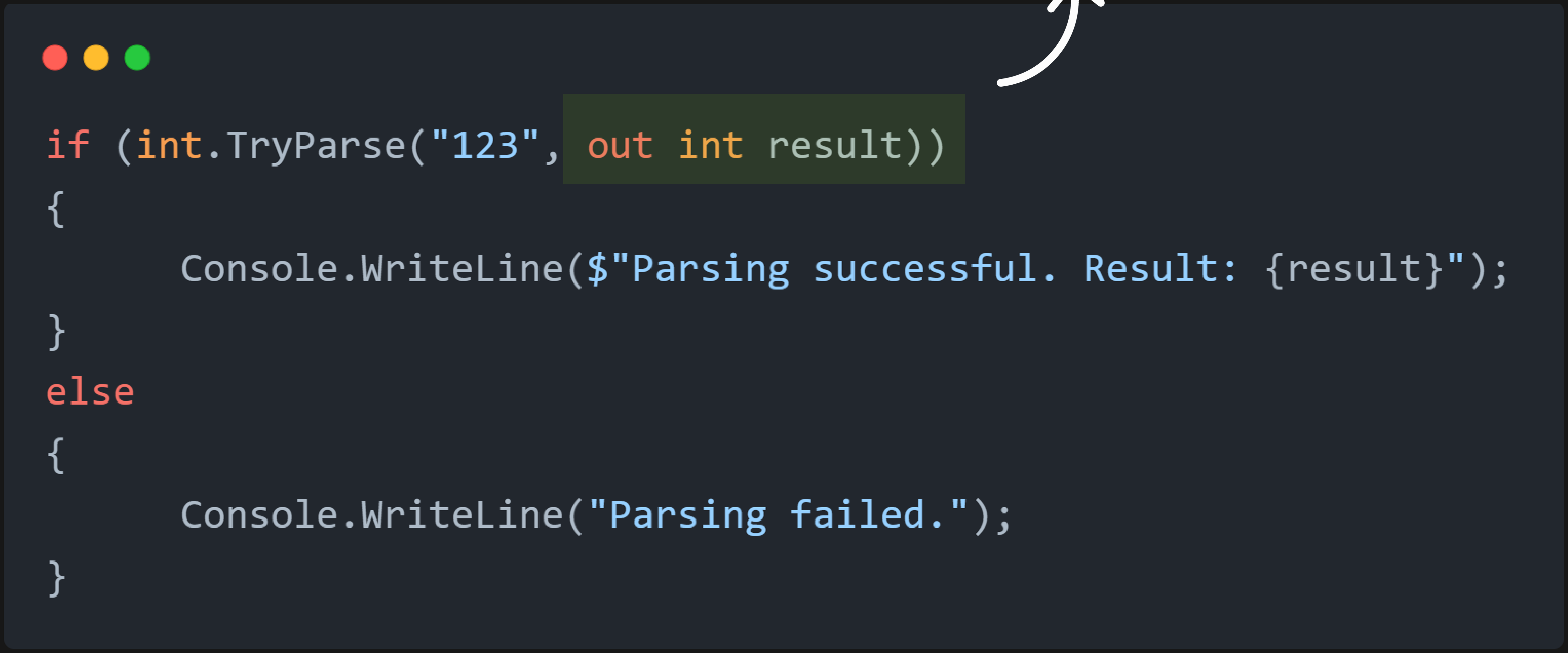
4 Out Variables Improvement



```
int result;

if (int.TryParse("123", out result))
{
     Console.WriteLine($"Parsing successful. Result: {result}");
}
else
{
      Console.WriteLine("Parsing failed.");
}
```

Separate variable declaration

Inline declaration and assignment using out

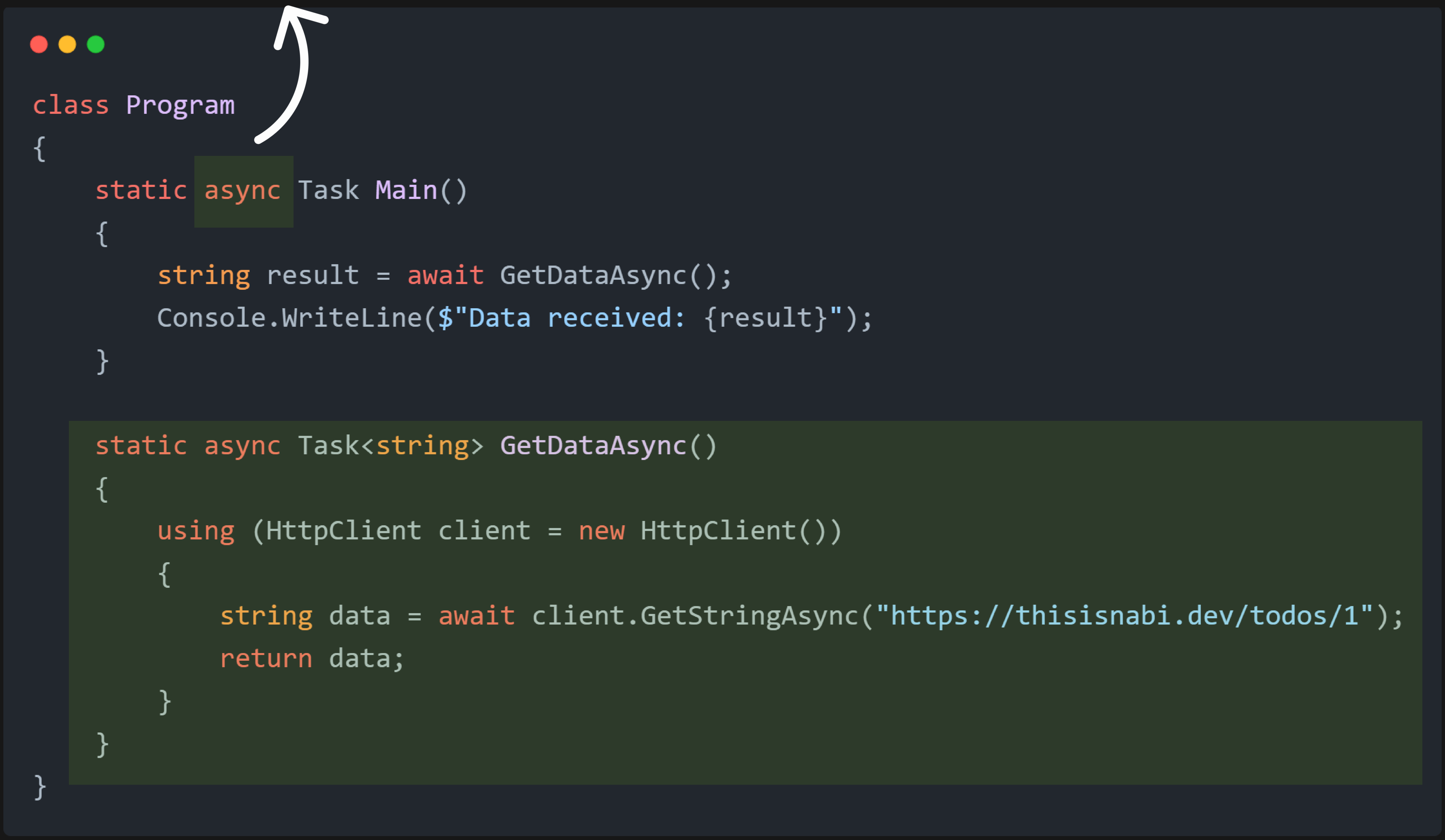


```
if (int.TryParse("123", out int result))
{
      Console.WriteLine($"Parsing successful. Result: {result}");
}
else
{
      Console.WriteLine("Parsing failed.");
}
```

Nabi Karampoor
@thisisnabi

5 Async Main

You can now write asynchronous code directly in the Main method.

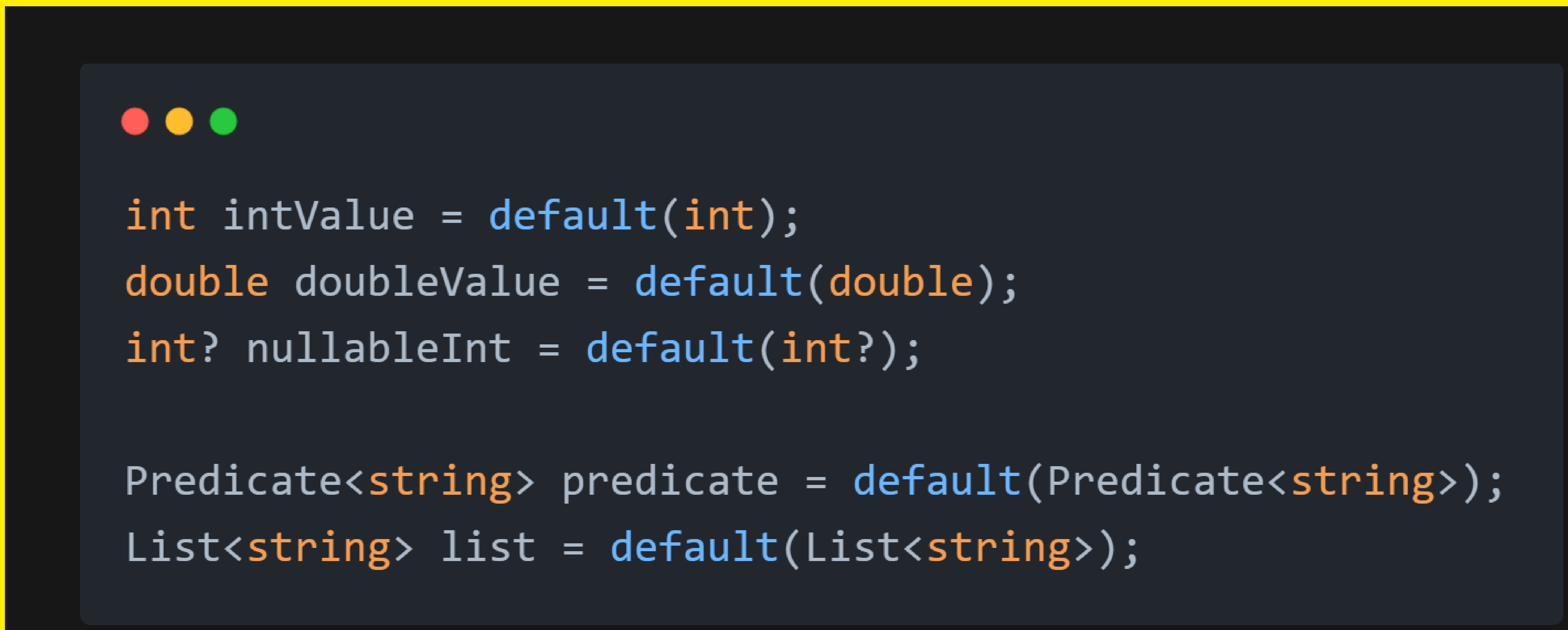


```
class Program
{
    static async Task Main()
    {
        string result = await GetDataAsync();
        Console.WriteLine($"Data received: {result}");
    }

    static async Task<string> GetDataAsync()
    {
        using (HttpClient client = new HttpClient())
        {
            string data = await client.GetStringAsync("https://thisisnabi.dev/todos/1");
            return data;
        }
    }
}
```

This allows your program to remain responsive and efficiently utilize resources while waiting for asynchronous operations to complete.

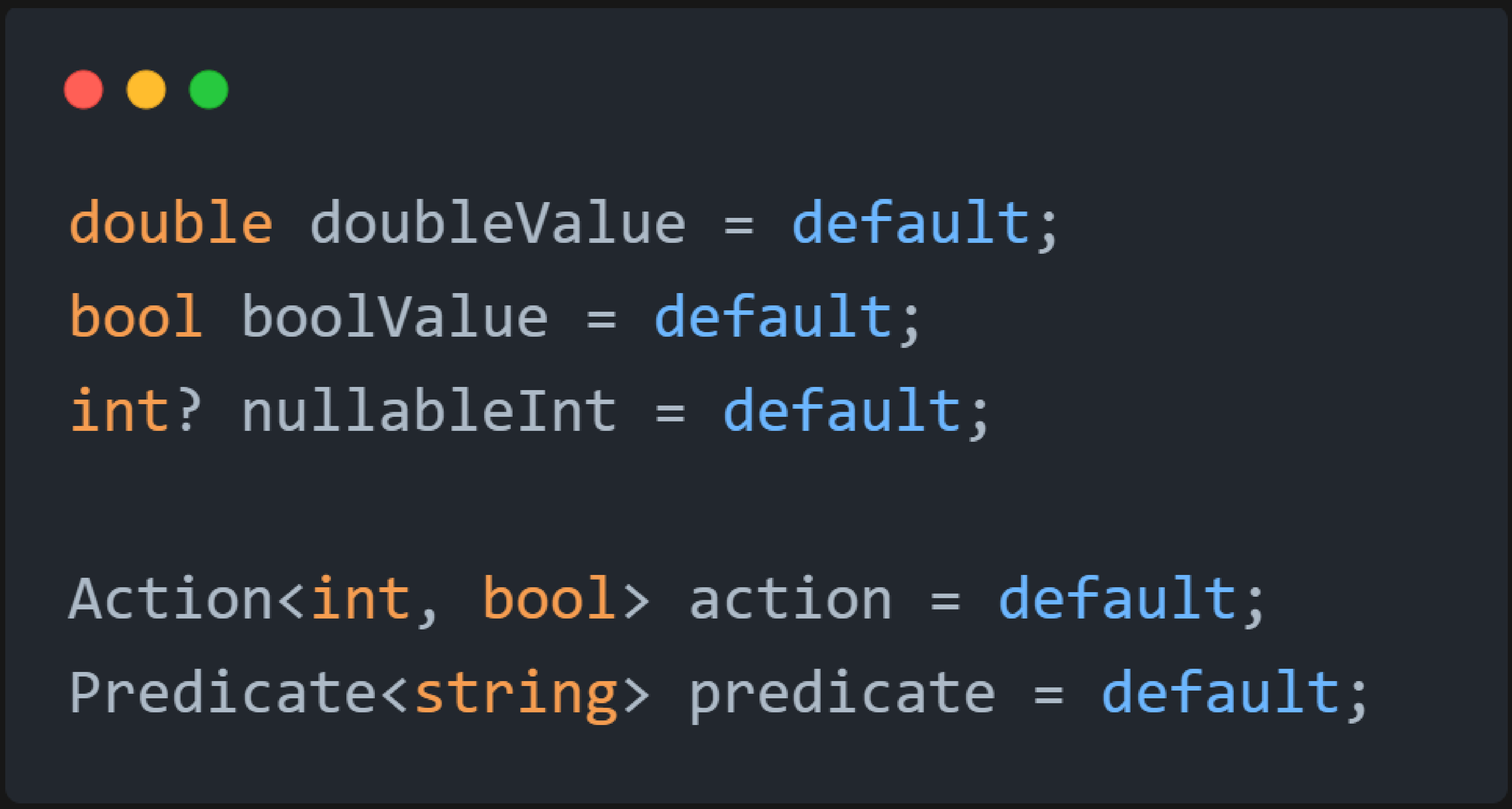
Nabi Karampoor
@thisisnabi

6 default Literal Expression



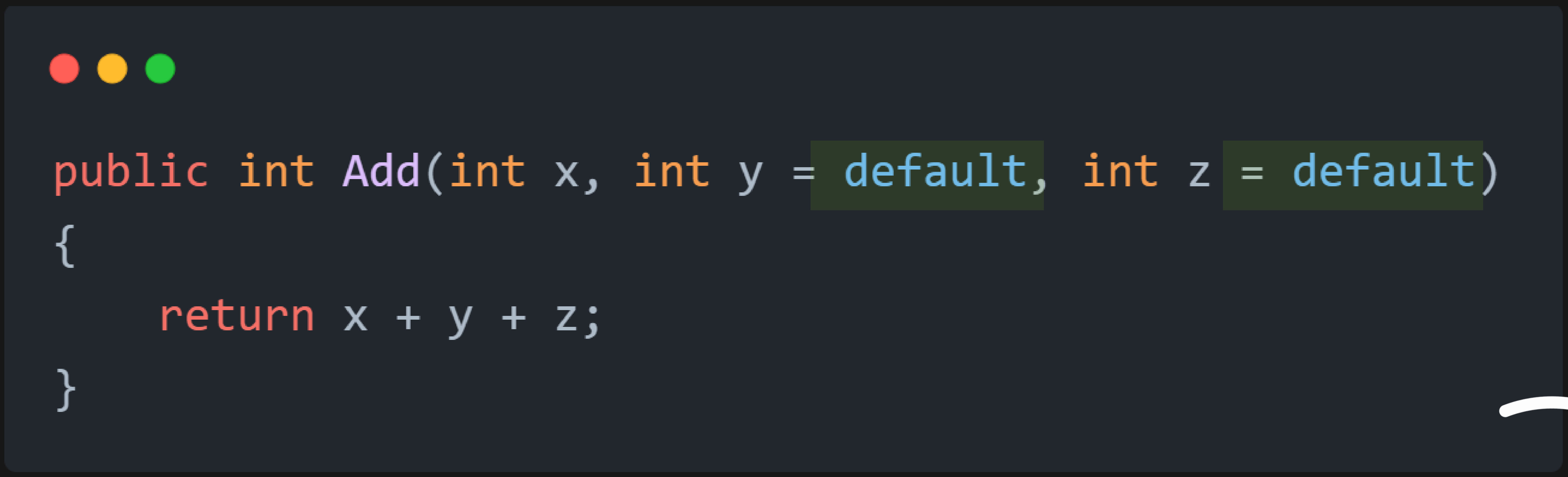
```
int intValue = default(int);
double doubleValue = default(double);
int? nullableInt = default(int?);

Predicate<string> predicate = default(Predicate<string>);
List<string> list = default(List<string>);
```

default(T) where T can be a value type or reference type.



```
double doubleValue = default;
bool boolValue = default;
int? nullableInt = default;

Action<int, bool> action = default;
Predicate<string> predicate = default;
```

Enhanced by removing the need to pass T as a parameter.

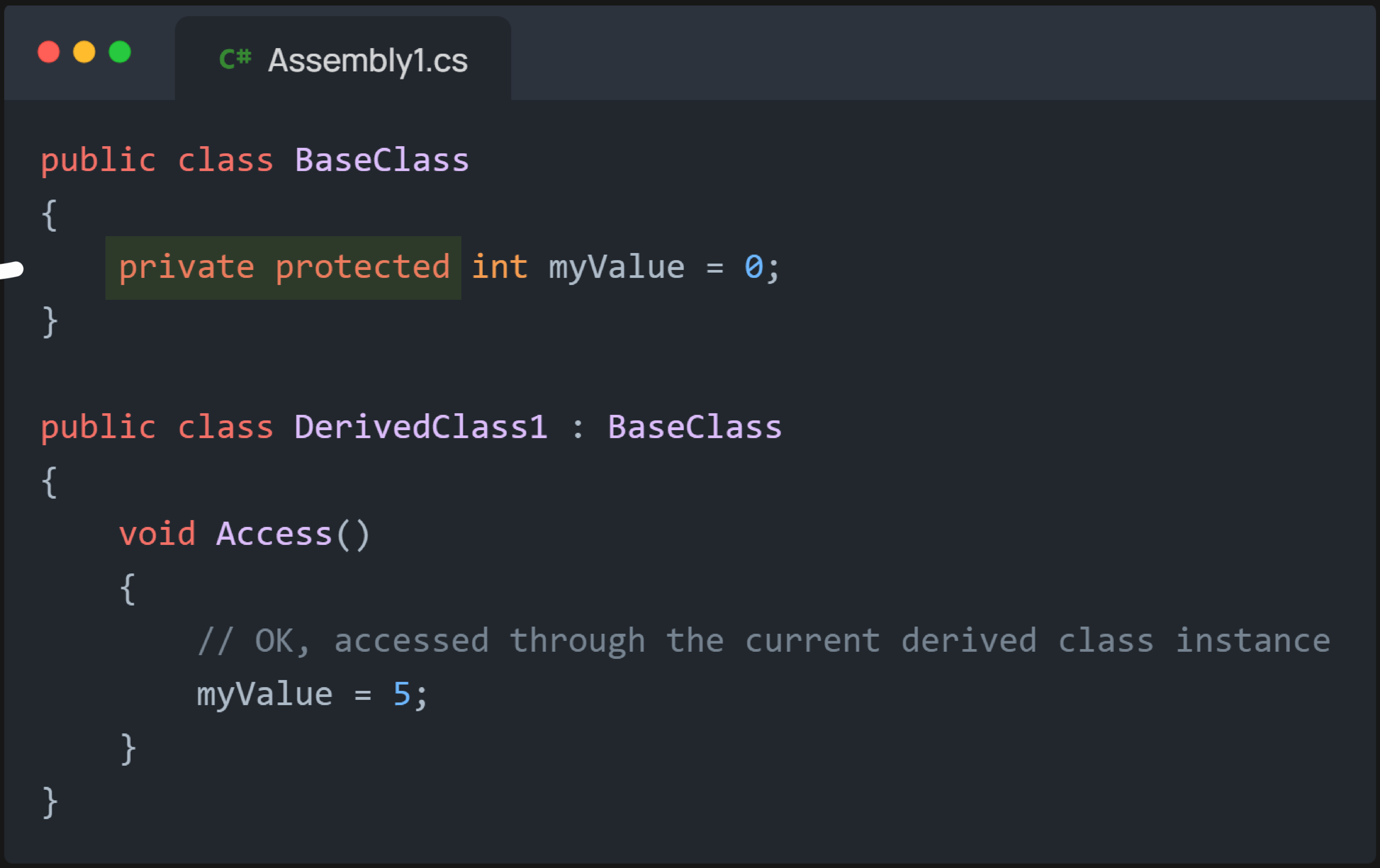


```
public int Add(int x, int y = default, int z = default)
{
    return x + y + z;
}
```

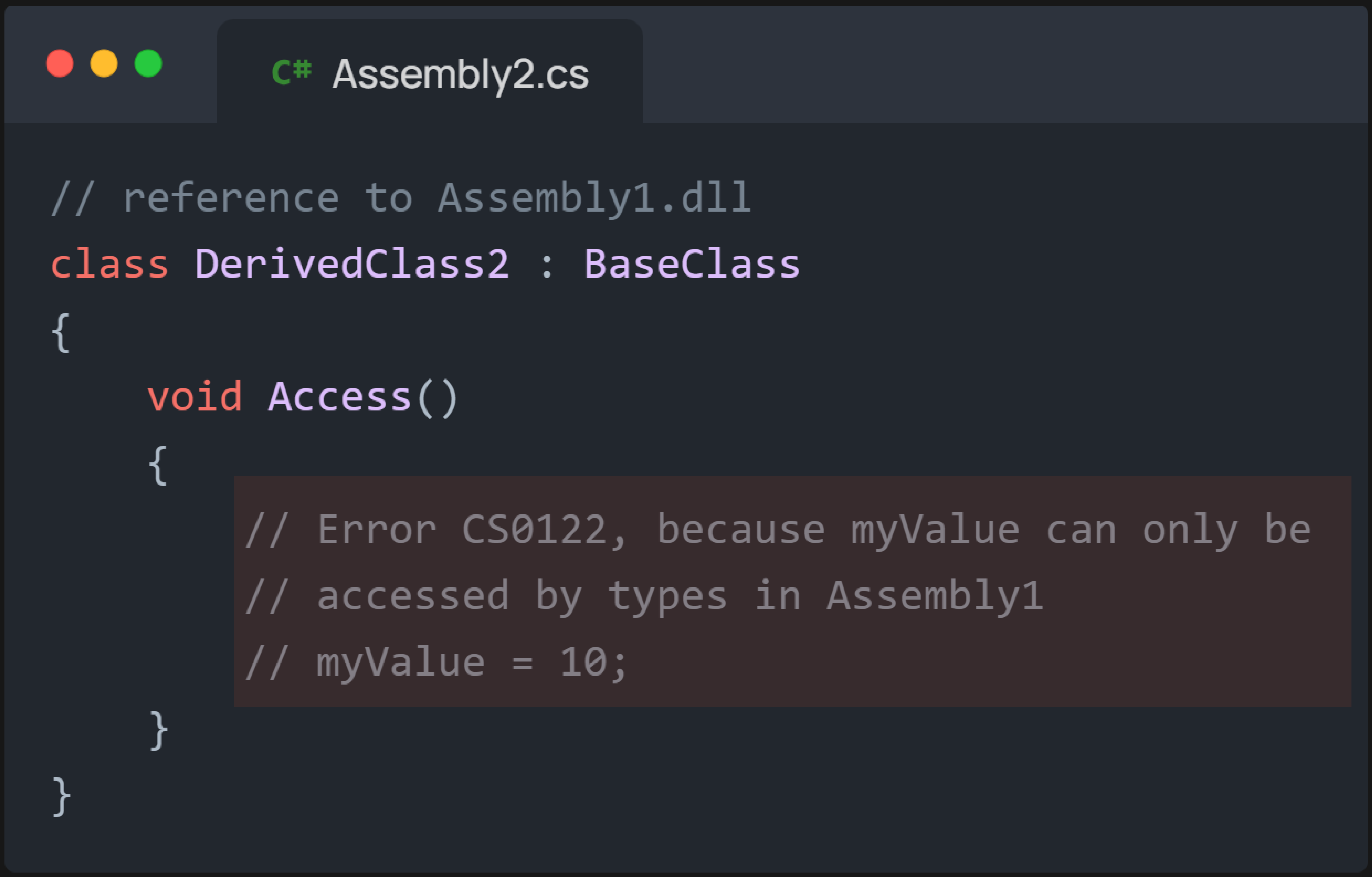
also work with method arguments

Nabi Karampoor
@thisisnabi

7 New Access modifier



```csharp
public class BaseClass
{
    private protected int myValue = 0;
}

public class DerivedClass1 : BaseClass
{
    void Access()
    {
        // OK, accessed through the current derived class instance
        myValue = 5;
    }
}
```

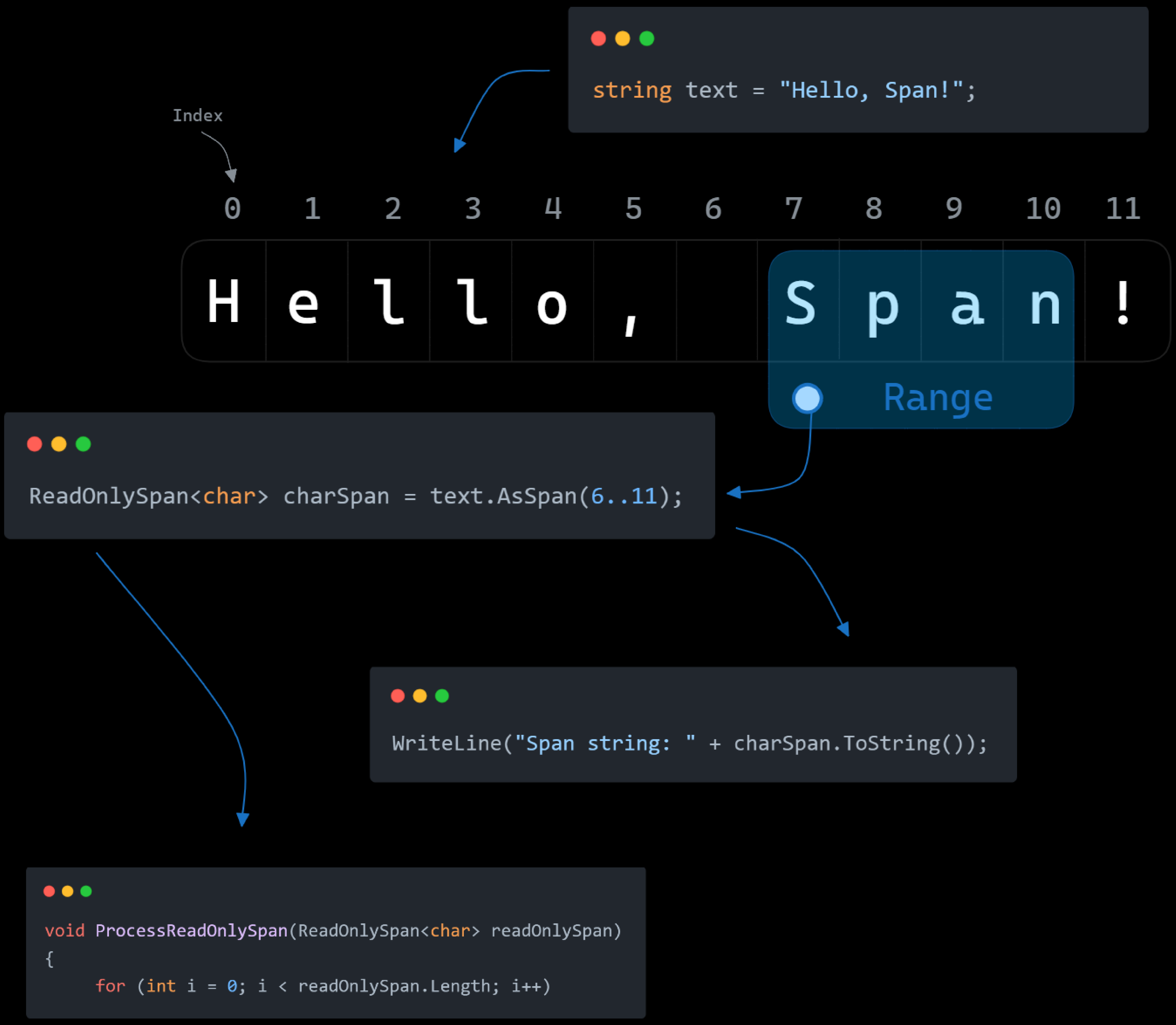


```csharp
// reference to Assembly1.dll
class DerivedClass2 : BaseClass
{
    void Access()
    {
        // Error CS0122, because myValue can only be
        // accessed by types in Assembly1
        // myValue = 10;
    }
}
```

The member is accessible within the containing assembly and by derived types, but only if they are in the same assembly.

Nabi Karampoor
@thisisnabi

8 Span<T>



```
string text = "Hello, Span!";
```

0	1	2	3	4	5	6	7	8	9	10	11
H	e	l	l	o	,		S	p	a	n	!

```
ReadOnlySpan<char> charSpan = text.AsSpan(6..11);
```

```
WriteLine("Span string: " + charSpan.ToString());
```

```
void ProcessReadOnlySpan(ReadOnlySpan<char> readOnlySpan)
{
    for (int i = 0; i < readOnlySpan.Length; i++)
```

Provides a more efficient and convenient way to work with memory buffers directly, reducing the need for unnecessary memory allocations and improving performance in certain scenarios.

Nabi Karampoor
@thisisnabi