.NET

WEEK 2

Sharad K Singh

AGENDA

Day1 - .NET Framework

Day 2 – Setup and Run .NET

Day 3 – Soft Skill

Day 4 – Data Types

Day 5 – Arrays

Sharad K Singh

DAY 1 OBJECTIVES

Introduction to .NET Framework

.NET Framework Apps

.NET Library Project

Microservice Project

Sharad K Singh

CHALLENGE OF THE DAY

Sharad K Singh

.NET FRAMEWORK

Console App Structure

.csproj - In .NET Framework, every single file and reference in your project had to be explicitly listed in a giant, messy XML file.

Must "Unload Project" first to edit this XML.

Uses App.config (XML)

References Node: Located in the Solution Explorer. You manually add DLLs

Sharad K Singh

.NET FRAMEWORK

Manual Inclusion: If you add a file to the folder, it doesn't appear in the app until you "Include in Project.“

Generates an .exe by default

No global usings

.sln vs .slnx file

Sharad K Singh

LIBRARY PROJECT

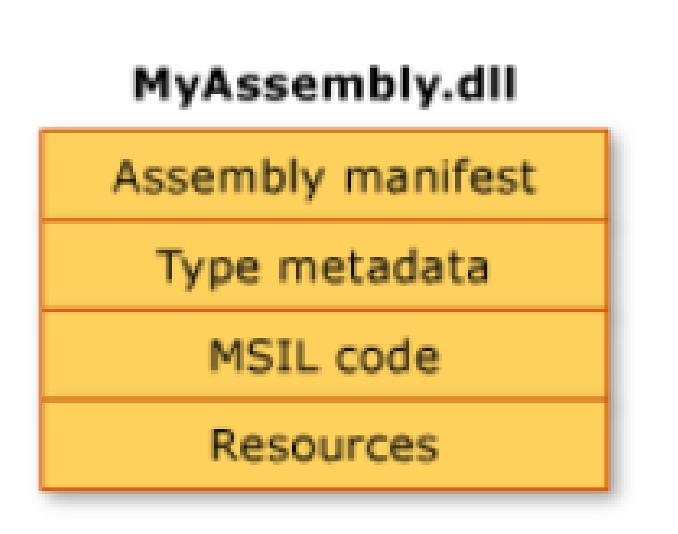
Create Library

Create Console

Refer and call library

Sharad K Singh

ASSEMBLY FILE



Assembly can consist of four elements:

1 The assembly manifest, which contains assembly metadata.

2 Type metadata.

3 Common intermediate language (CIL) code that implements the types.

4 A set of resources.

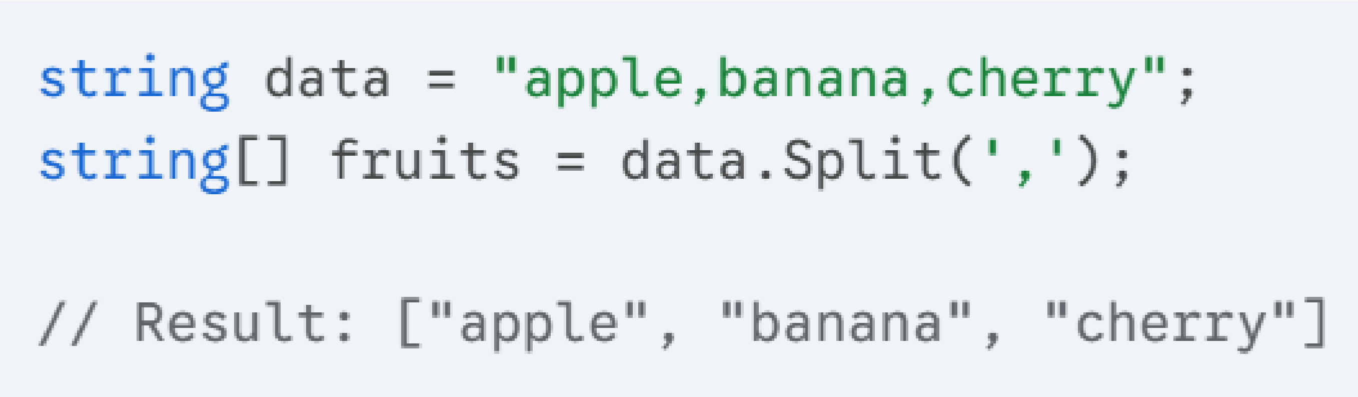
INPUT

Split

Join

Sharad K Singh

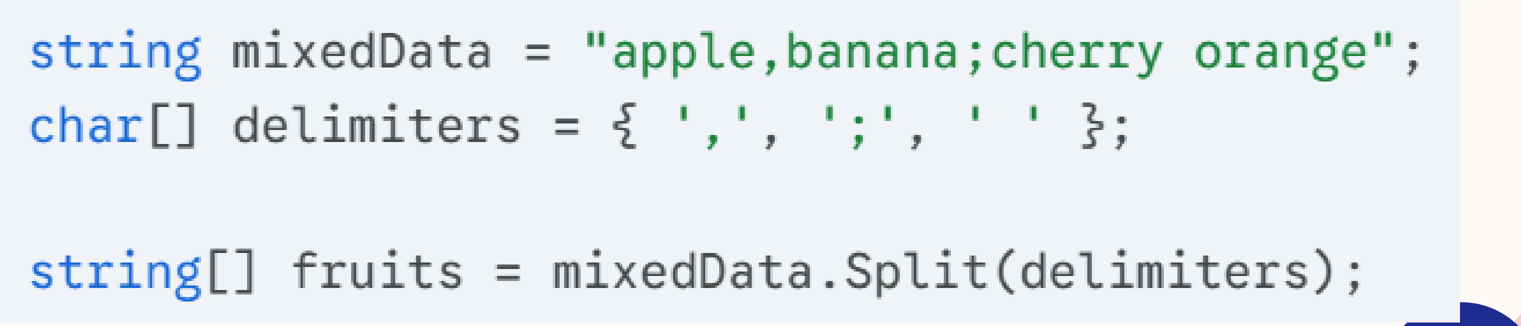
SPLIT()



```
string data = "apple,banana,cherry";
string[] fruits = data.Split(',');

// Result: ["apple", "banana", "cherry"]
```

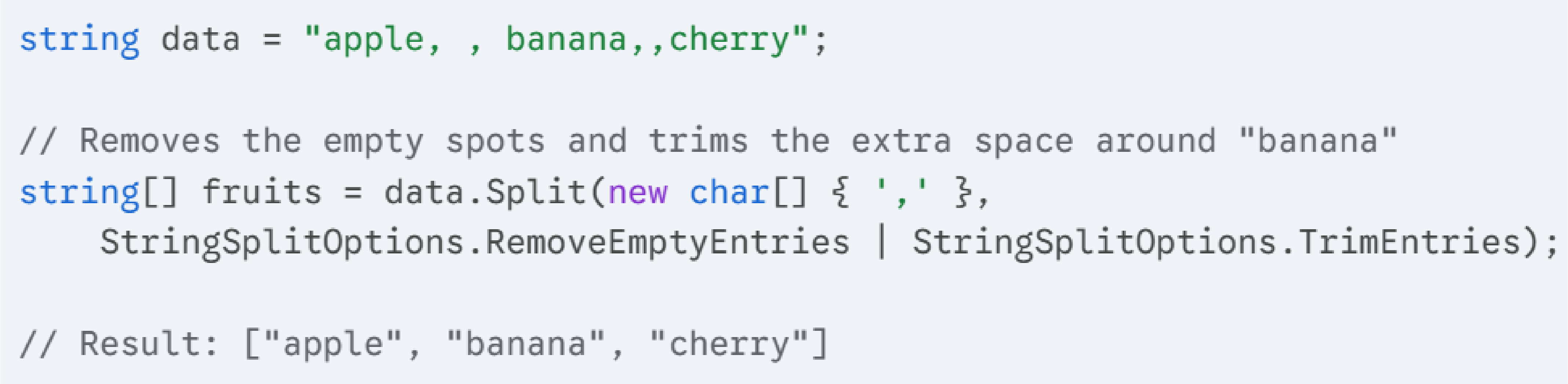
Sharad K Singh

SPLIT() WITH MULTIPLE DELIMITERS



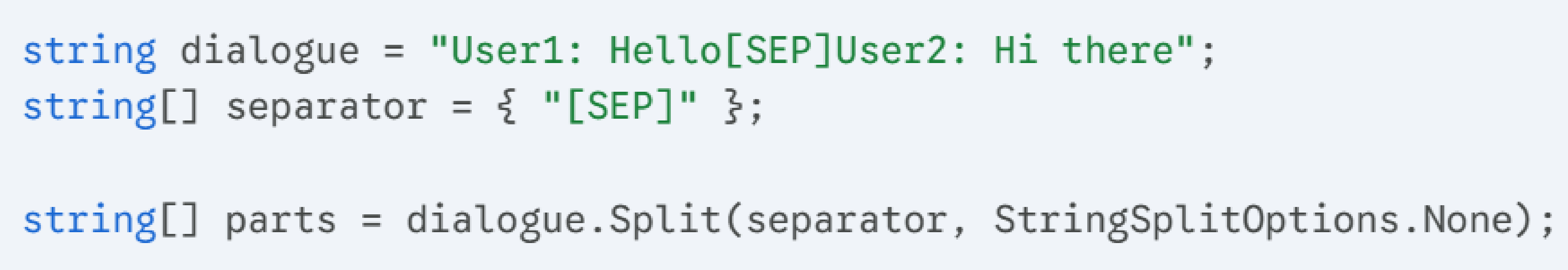
```
string mixedData = "apple,banana;cherry orange";
char[] delimiters = { ',', ';', ' ' };

string[] fruits = mixedData.Split(delimiters);
```

SPLIT() – EMPTY ENTRIES



```
string data = "apple, , banana,,cherry";

// Removes the empty spots and trims the extra space around "banana"
string[] fruits = data.Split(new char[] { ',' },
    StringSplitOptions.RemoveEmptyEntries | StringSplitOptions.TrimEntries);

// Result: ["apple", "banana", "cherry"]
```

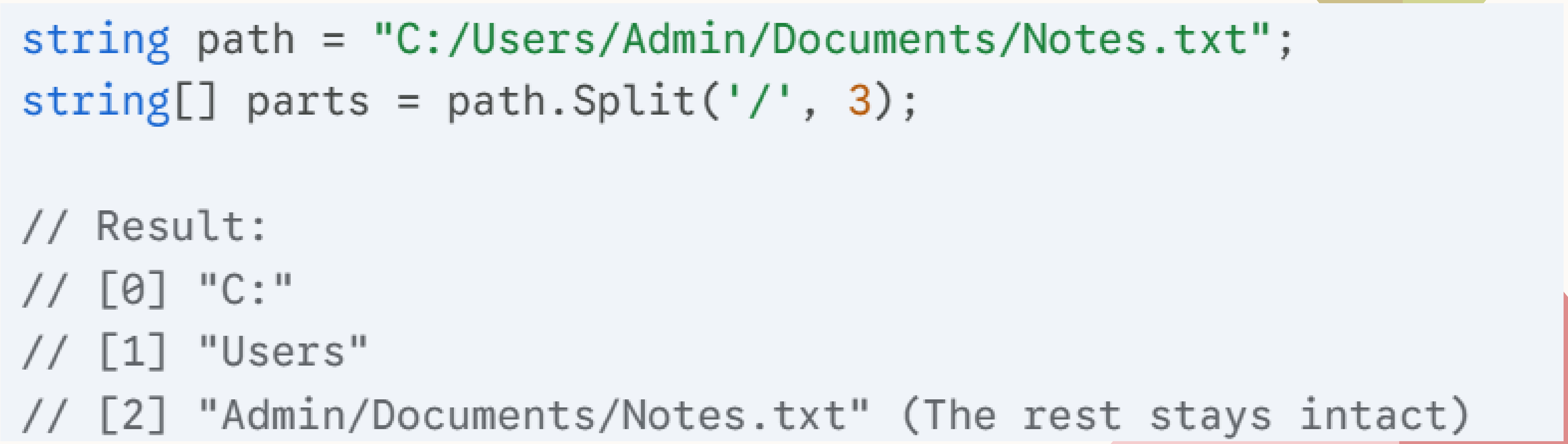
SPLIT() BY STRING AND STRING ARRAY



```
string dialogue = "User1: Hello[SEP]User2: Hi there";
string[] separator = { "[SEP]" };

string[] parts = dialogue.Split(separator, StringSplitOptions.None);
```

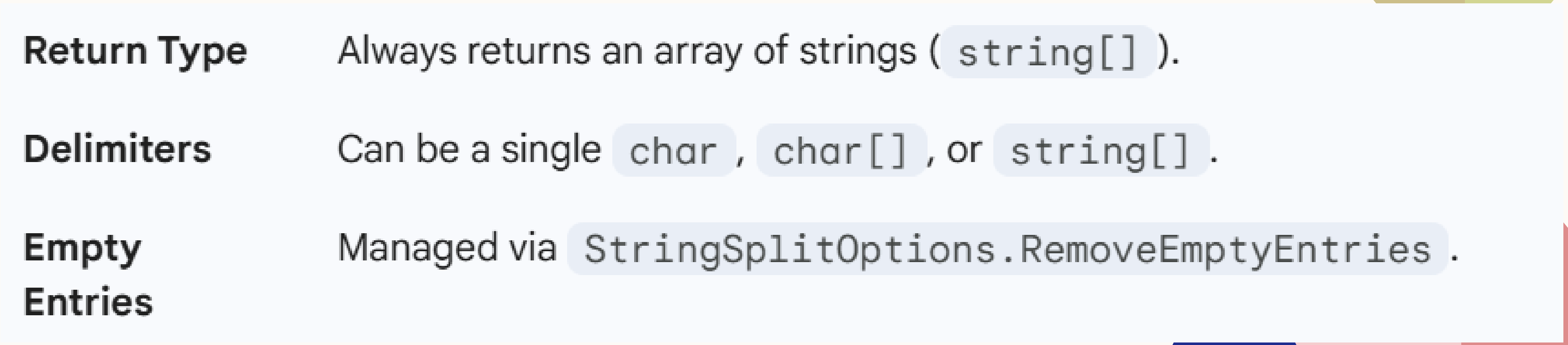
Sharad K Singh

SPLIT() – LIMIT RESULTS



```
string path = "C:/Users/Admin/Documents/Notes.txt";
string[] parts = path.Split('/', 3);

// Result:
// [0] "C:"
// [1] "Users"
// [2] "Admin/Documents/Notes.txt" (The rest stays intact)
```

SPLIT() – NOTES



Return Type	Always returns an array of strings (`string[]`).
Delimiters	Can be a single `char` , `char[]` , or `string[]` .
Empty Entries	Managed via `StringSplitOptions.RemoveEmptyEntries` .

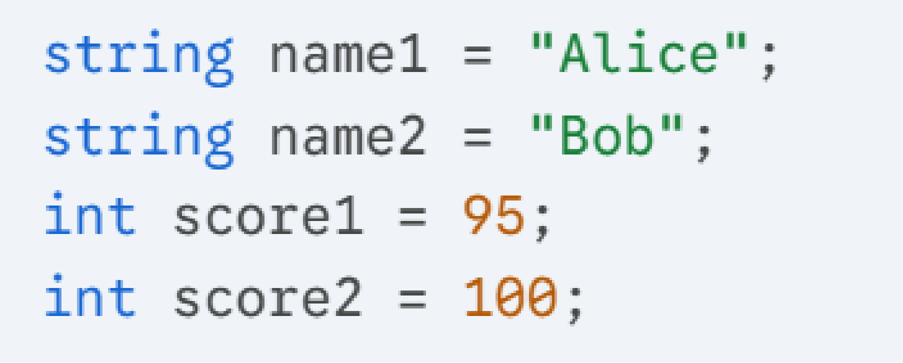
OUTPUT FORMATTING

Width

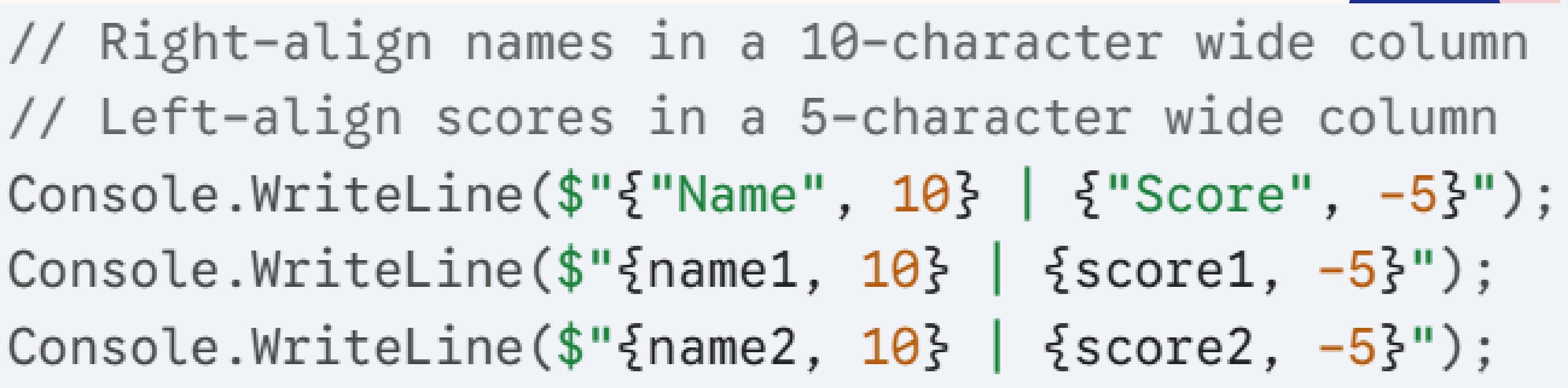
Format

Sharad K Singh

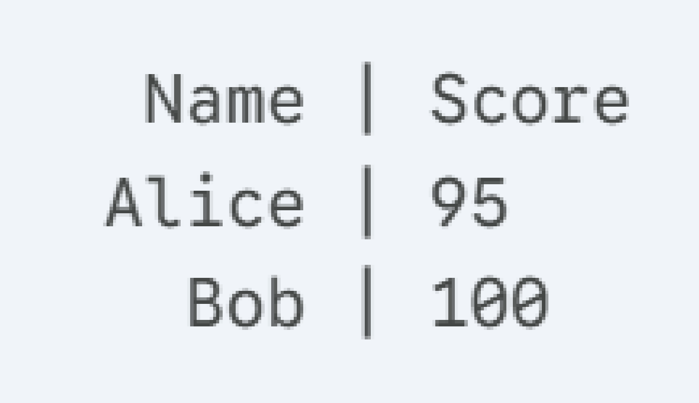
OUTPUT WIDTH FORMATTING



```
string name1 = "Alice";
string name2 = "Bob";
int score1 = 95;
int score2 = 100;
```



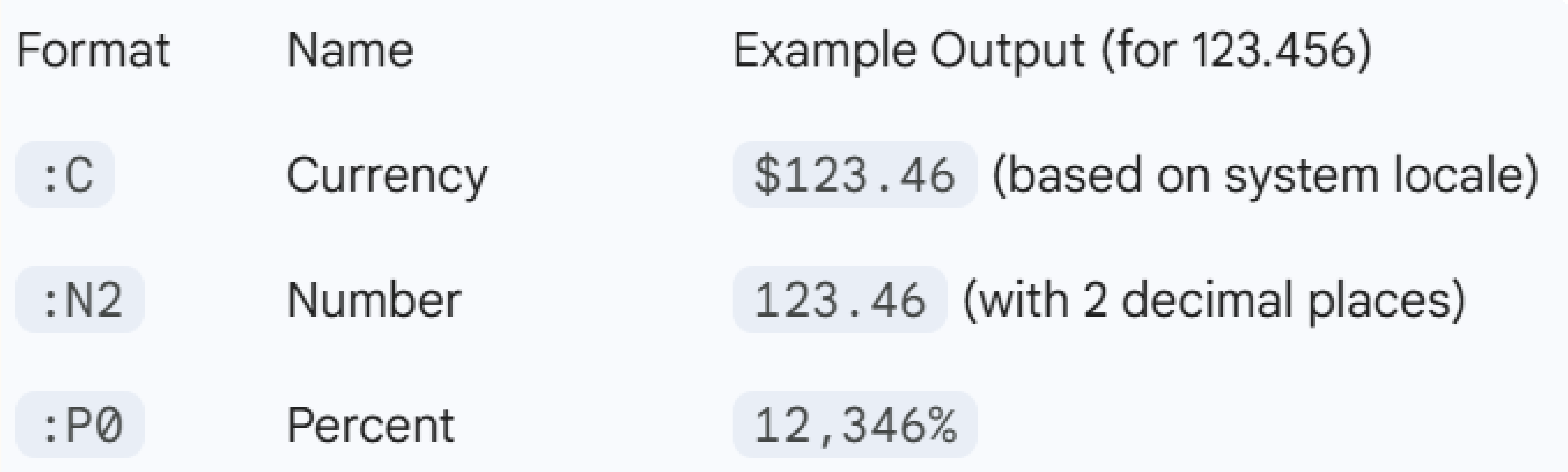
```
// Right-align names in a 10-character wide column
// Left-align scores in a 5-character wide column
Console.WriteLine($"{"Name", 10} | {"Score", -5}");
Console.WriteLine($"{name1, 10} | {score1, -5}");
Console.WriteLine($"{name2, 10} | {score2, -5}");
```



```
      Name | Score
     Alice | 95
       Bob | 100
```

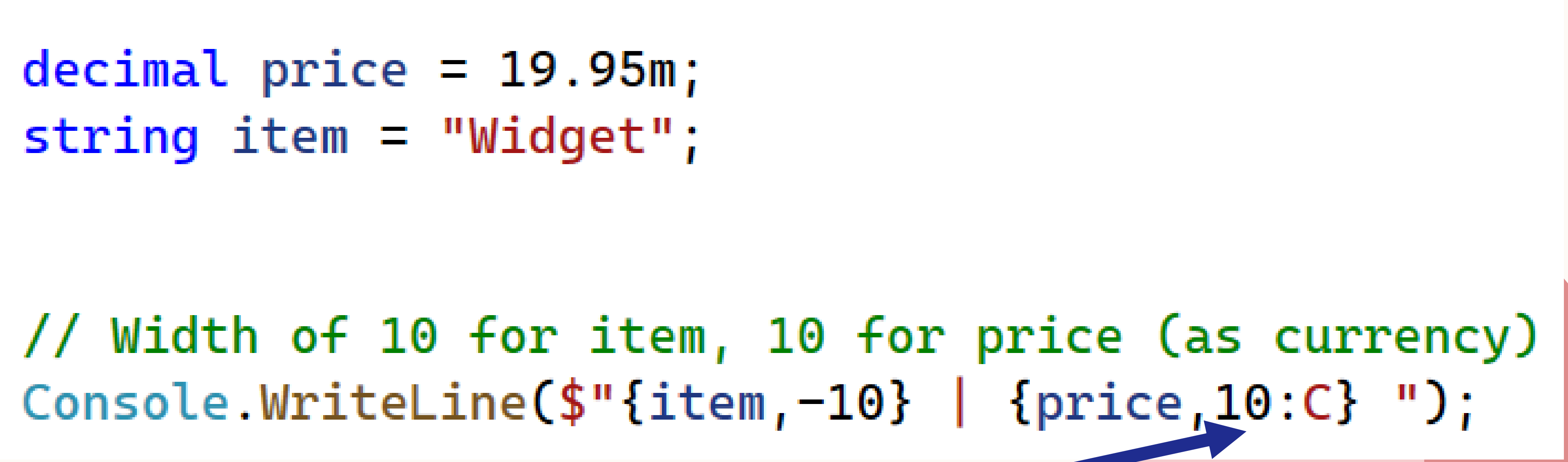
Sharad K Singh

OUTPUT NUMERIC FORMATTING



Format	Name	Example Output (for 123.456)
`:C`	Currency	`$123.46` (based on system locale)
`:N2`	Number	`123.46` (with 2 decimal places)
`:P0`	Percent	`12,346%`

OUTPUT NUMERIC FORMATTING

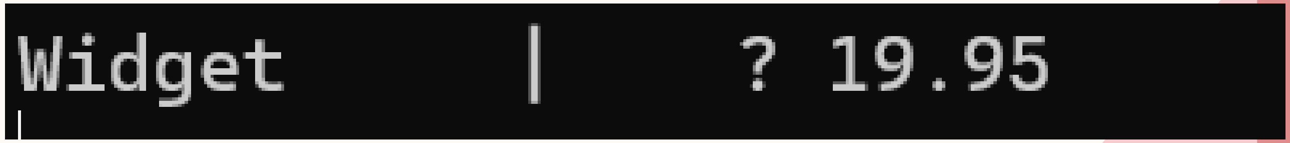


```
decimal price = 19.95m;
string item = "Widget";



// Width of 10 for item, 10 for price (as currency)
Console.WriteLine($"{item,-10} | {price,10:C} ");
```

The **width** always comes before the **format.**

OUTPUT NUMERIC FORMATTING

If you are seeing a question mark (?) instead of the Rupee symbol (₹), it is because the Console's output encoding is set to a restricted format like ASCII



```
Widget      |    ? 19.95
```

To fix this, you must explicitly output using UTF-8 encoding

Console.OutputEncoding = Encoding.UTF8;

STRING FORMATTING

$ String (Interpolated String)

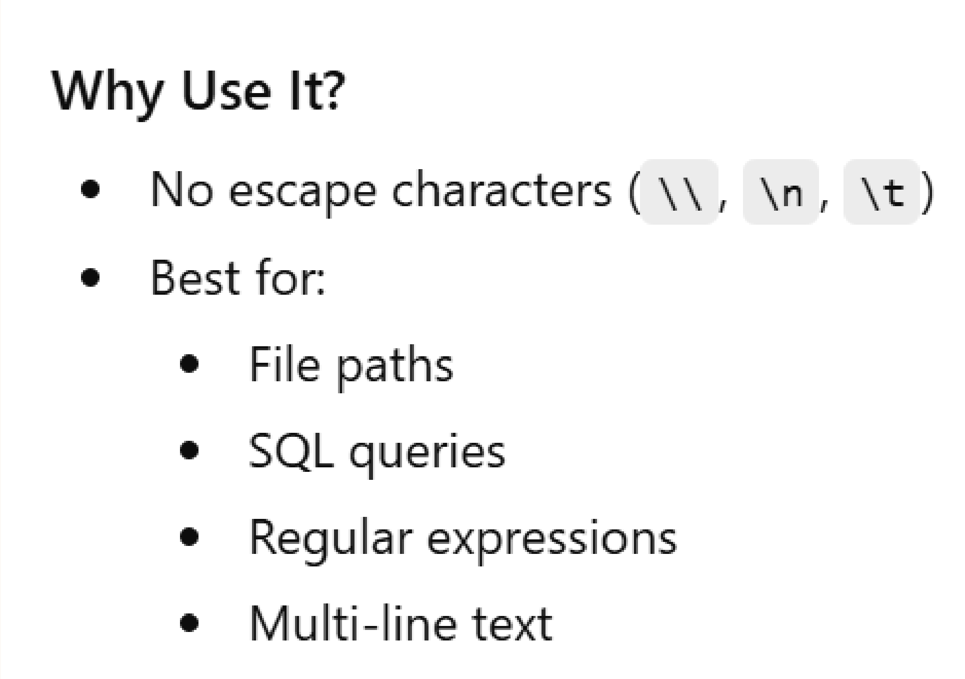
- A $ string allows you to **embed expressions directly inside the string.**

@ String (Verbatim String)

- An @ string treats text **exactly as typed**.

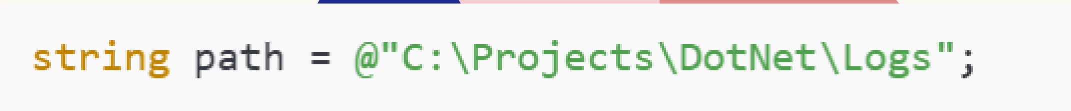
STRING FORMATTING

@ String (Verbatim String)

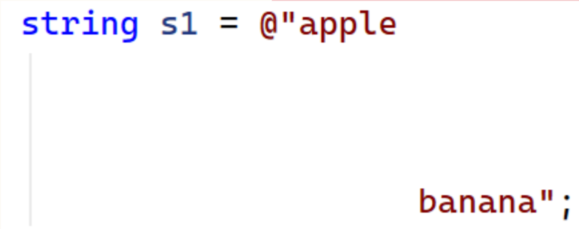


Why Use It?

- No escape characters (`\\`, `\n`, `\t`)
- Best for:
  - File paths
  - SQL queries
  - Regular expressions
  - Multi-line text



```
string path = @"C:\Projects\DotNet\Logs";
```



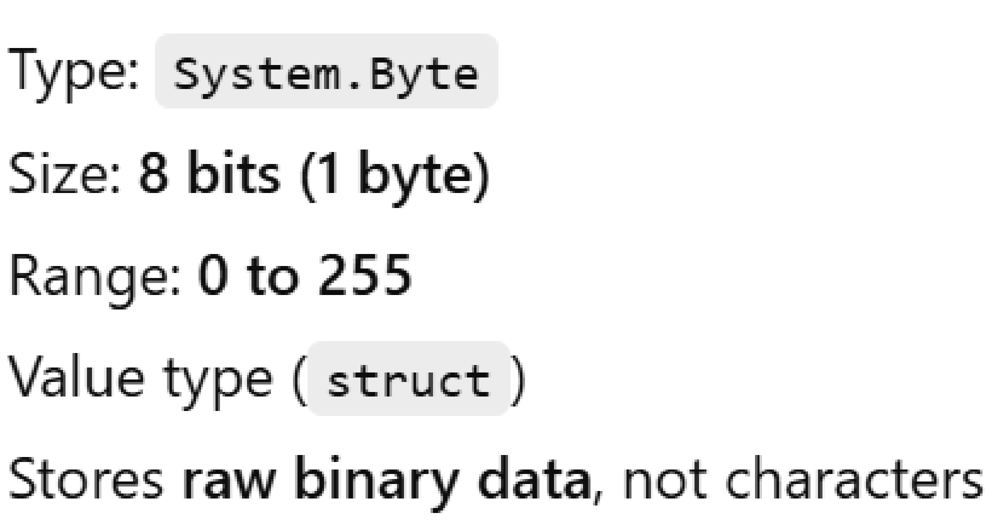
```
string s1 = @"apple



                    banana";
```

Sharad K Singh

.NET BYTE

byte is an 8-bit unsigned integer value type

byte Is NOT for Text



Type: `System.Byte`

Size: **8 bits (1 byte)**

Range: **0 to 255**

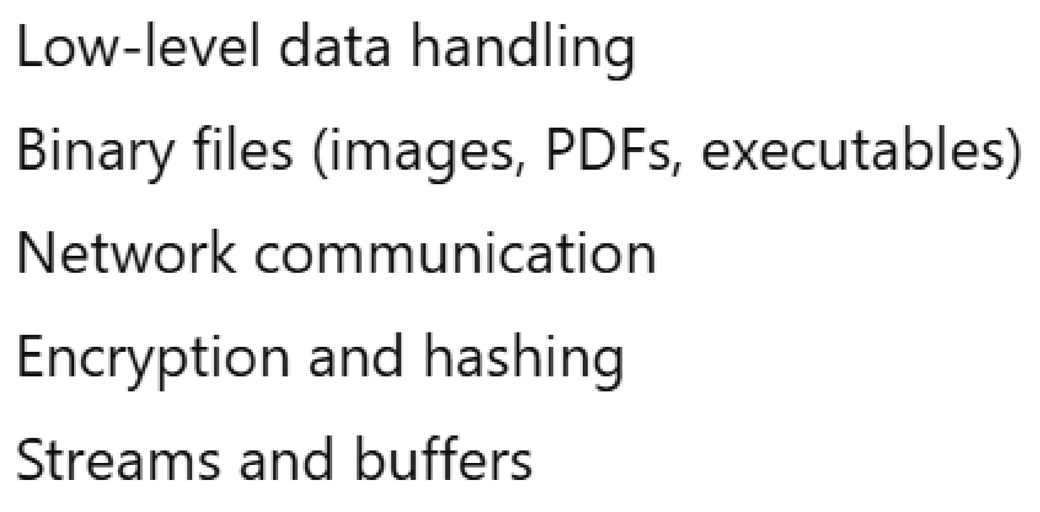
Value type (`struct`)

Stores **raw binary data**, not characters

Sharad K Singh

.NET BYTE

Why byte Exists



Low-level data handling

Binary files (images, PDFs, executables)

Network communication

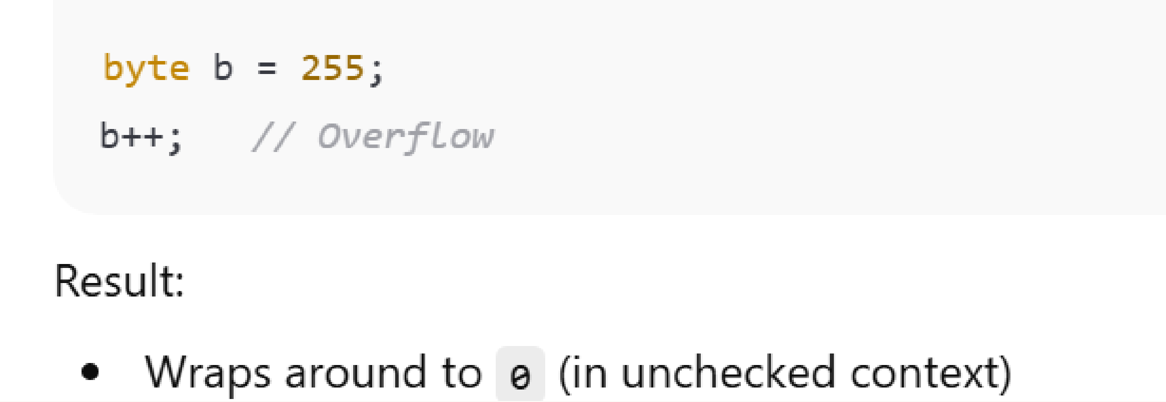
Encryption and hashing

Streams and buffers

Sharad K Singh

.NET BYTE

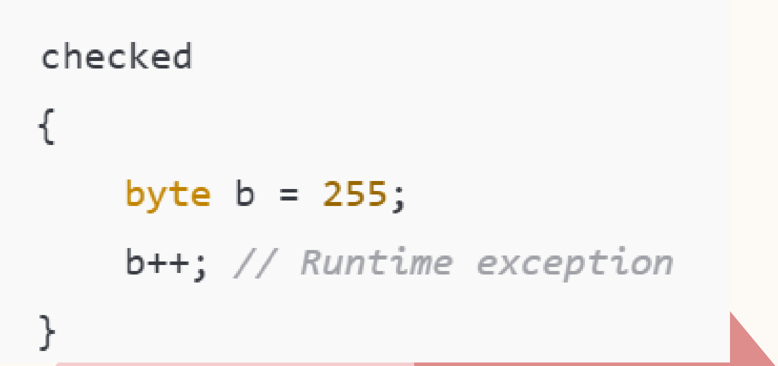
Range and Overflow



```
byte b = 255;
b++;   // Overflow
```

Result:

- Wraps around to `0` (in unchecked context)

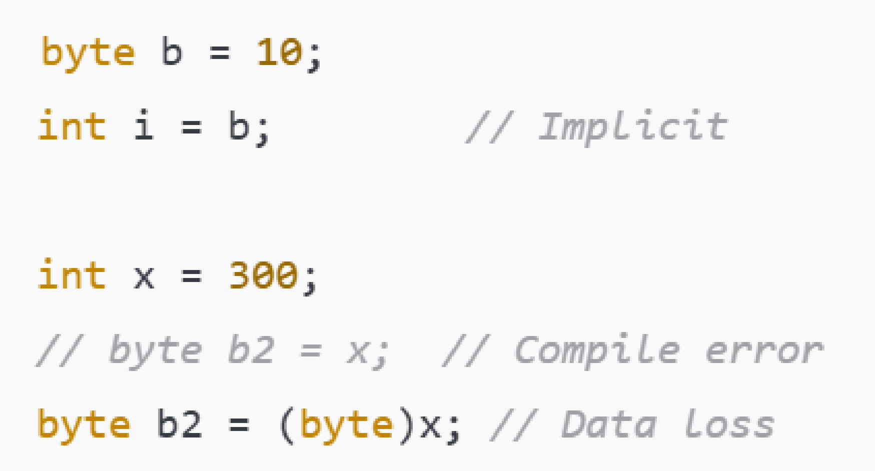


```
checked
{
    byte b = 255;
    b++; // Runtime exception
}
```

Sharad K Singh

.NET BYTE

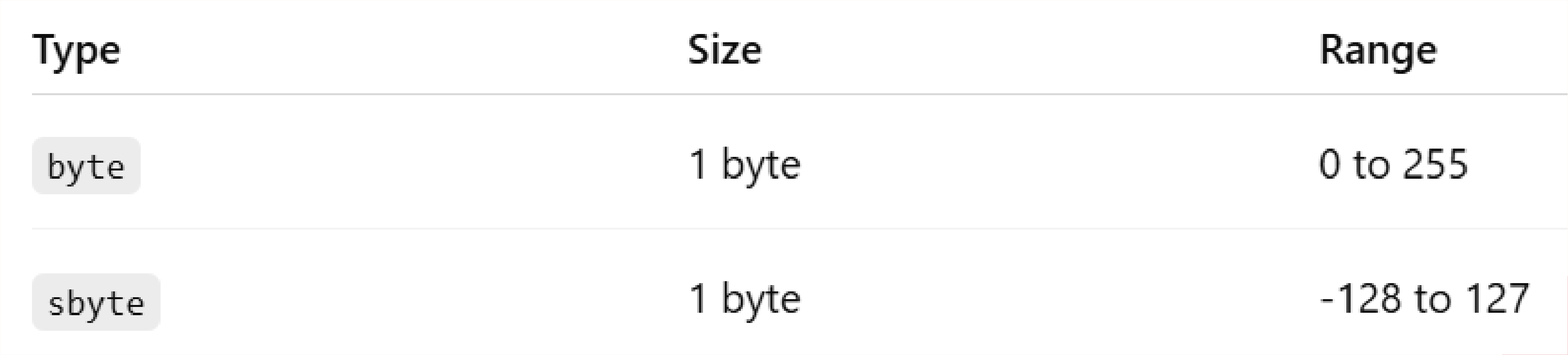
byte Conversion Rules



```
byte b = 10;
int i = b;        // Implicit

int x = 300;
// byte b2 = x;  // Compile error
byte b2 = (byte)x; // Data loss
```

Sharad K Singh

.NET BYTE



Type	Size	Range
`byte`	1 byte	0 to 255
`sbyte`	1 byte	-128 to 127

Sharad K Singh

.NET CHAR

Value Type

char represents a single UTF-16 code unit

Type name: System.Char

Size: 16 bits (2 bytes)

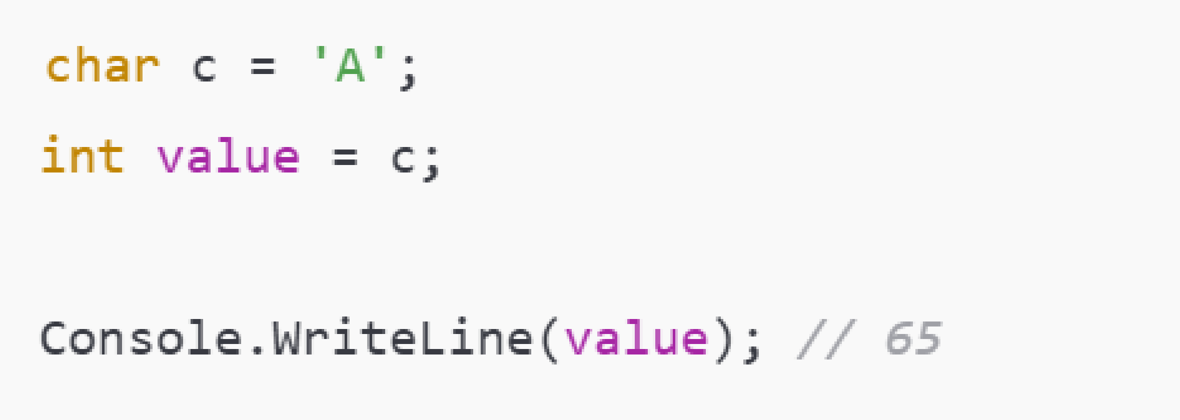
Stores a UTF-16 encoded value

.NET CHAR

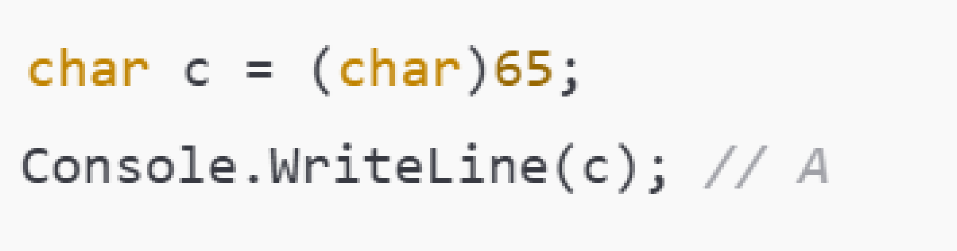
```
char c = '9';

char.IsDigit(c);      // true
char.IsLetter(c);     // false
char.IsWhiteSpace(c); // false
char.ToUpper('a');    // 'A'
```

Sharad K Singh

.NET CHAR



```
char c = 'A';
int value = c;

Console.WriteLine(value); // 65
```



```
char c = (char)65;
Console.WriteLine(c); // A
```

Sharad K Singh

.NET STRING API – OVERVIEW

String is an immutable reference type

Defined in System namespace

Stores Unicode characters

Widely used in enterprise applications

UNICODE

In many other programming languages, a string is an array of characters. This is not the case with C#. In C#, strings are objects. Thus, string is a reference type.

CREATION

The easiest way to construct a string is to use a string literal.

You can also create a string from a char array.

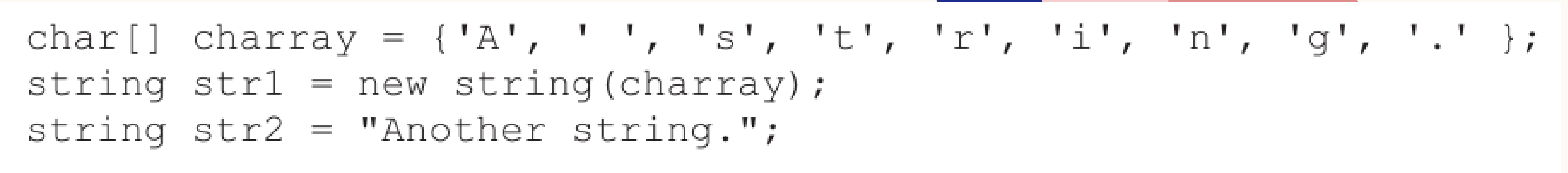
STRING CREATION

Literal assignment: string s = "Hello"

Using constructor: new string()

string.Empty

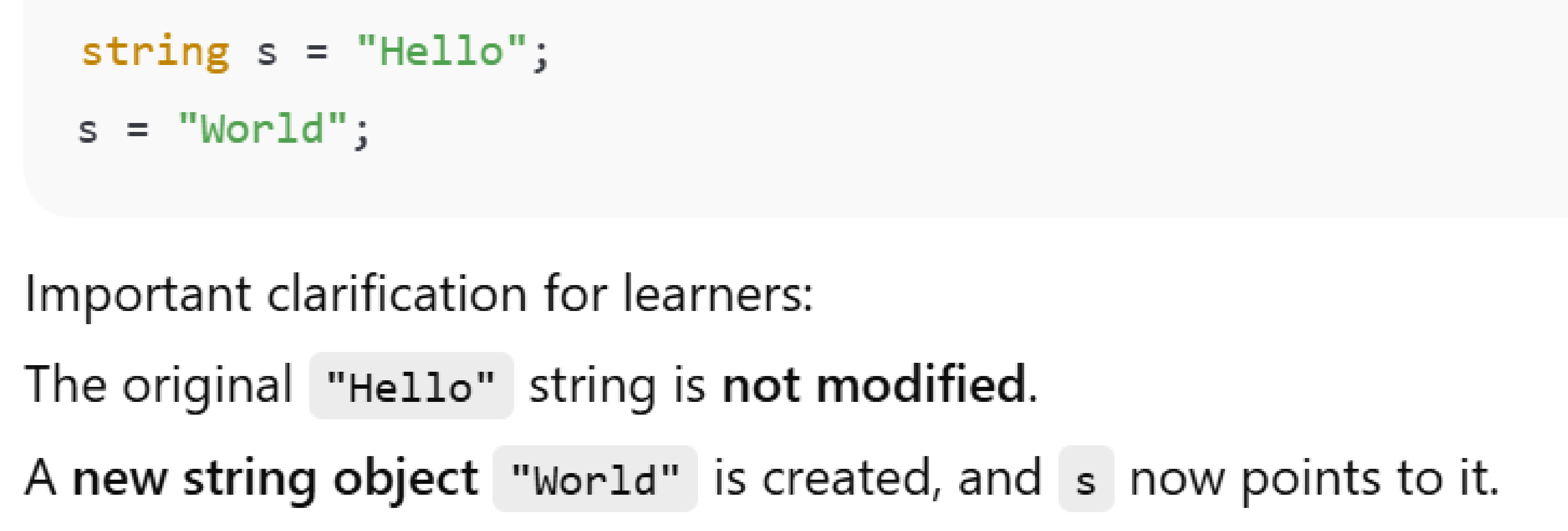
From char arrays



```
char[] charray = {'A', ' ', 's', 't', 'r', 'i', 'n', 'g', '.' };
string str1 = new string(charray);
string str2 = "Another string.";
```

IMMUTABLE

Once an object is created, its value **cannot be changed.**



```
string s = "Hello";
s = "World";
```

Important clarification for learners:

The original `"Hello"` string is **not modified**.

A **new string object** `"World"` is created, and `s` now points to it.

IMMUTABLE

Performance (String Interning)

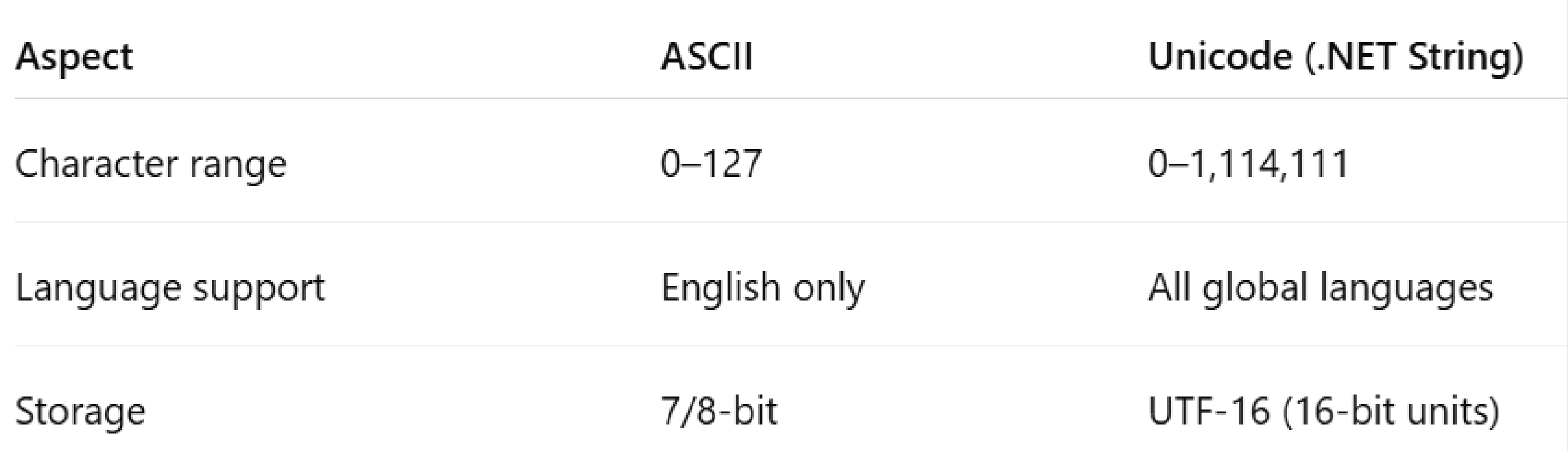
.NET stores identical string literals **only once**

Multiple variables can safely share the same string

UNICODE

.NET strings are implemented using UTF-16 encoding

The underlying type is System.Char (16-bit)



Aspect	ASCII	Unicode (.NET String)
Character range	0–127	0–1,114,111
Language support	English only	All global languages
Storage	7/8-bit	UTF-16 (16-bit units)

Sharad K Singh

KEY CHARACTERISTICS OF STRING

Immutable – modification creates a new instance

Thread-safe by default

Zero-based index access

Rich API support

KEY CHARACTERISTICS OF STRING

Thread Safety

Immutable objects are **inherently thread-safe**

Multiple threads can read the same string safely

No locking required

IMPORTANT PROPERTIES AND METHODS

Length

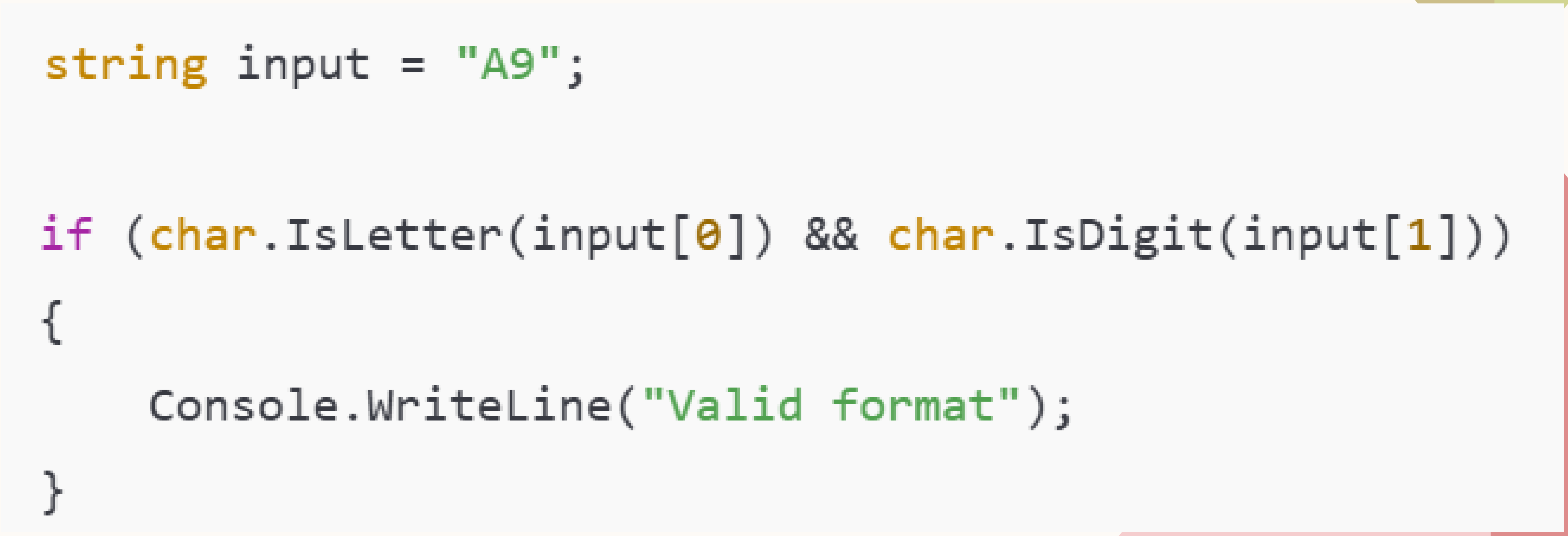
string[index]

IsNullOrEmpty

IsNullOrWhiteSpace

Sharad K Singh

USING INDEX



```
string input = "A9";

if (char.IsLetter(input[0]) && char.IsDigit(input[1]))
{
    Console.WriteLine("Valid format");
}
```

STRING.ISNULLOREMPTY

Returns true if:

The string is null, **or**

The string is empty ("")

Sharad K Singh

STRING. ISNULLORWHITESPACE

Returns true if:

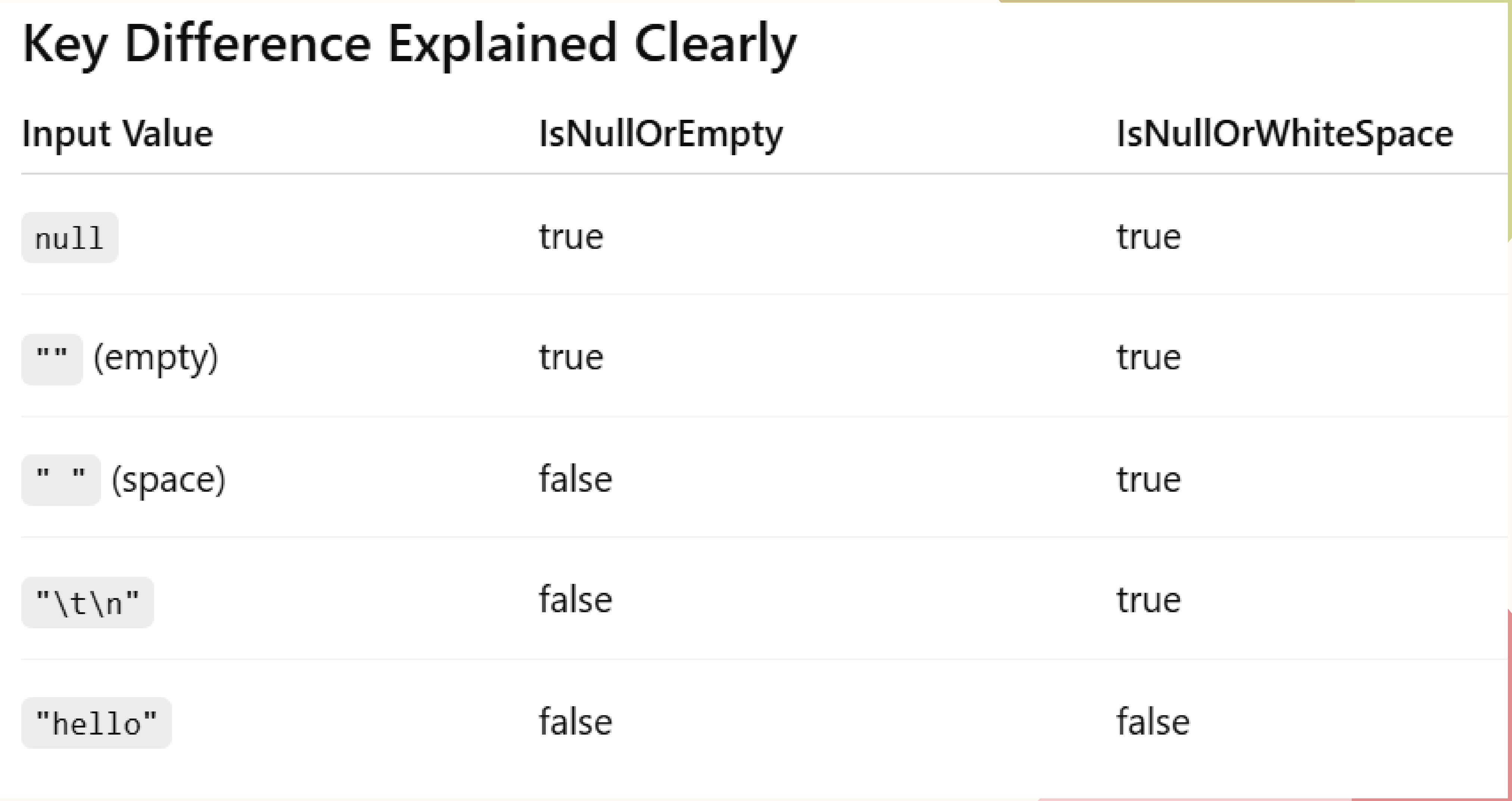
The string is null, **or**

The string is empty (""), **or**

The string contains **only whitespace characters**

Including space, tab ‘\t\ , new line ‘\n’

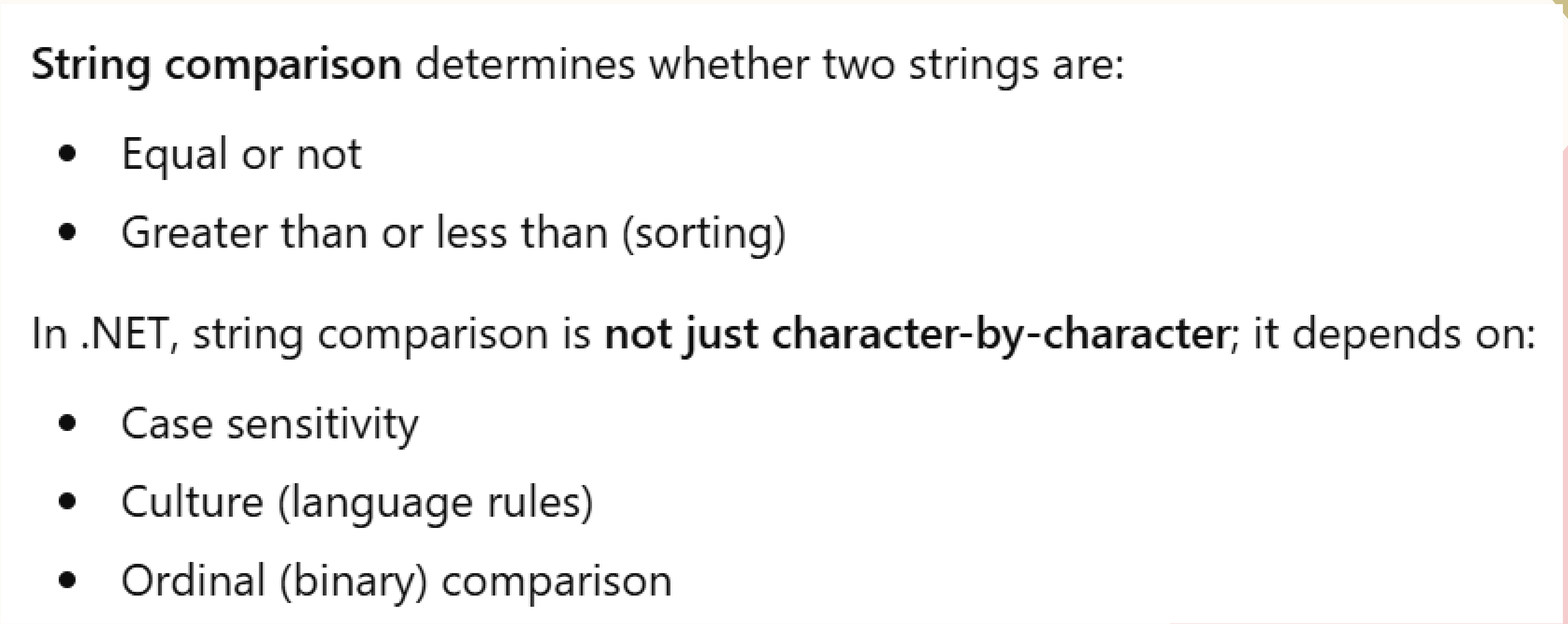
Sharad K Singh



Key Difference Explained Clearly

Input Value	IsNullOrEmpty	IsNullOrWhiteSpace
`null`	true	true
`""` (empty)	true	true
`" "` (space)	false	true
`"\t\n"`	false	true
`"hello"`	false	false

STRING COMPARISON



String comparison determines whether two strings are:

- Equal or not
- Greater than or less than (sorting)

In .NET, string comparison is **not just character-by-character**; it depends on:

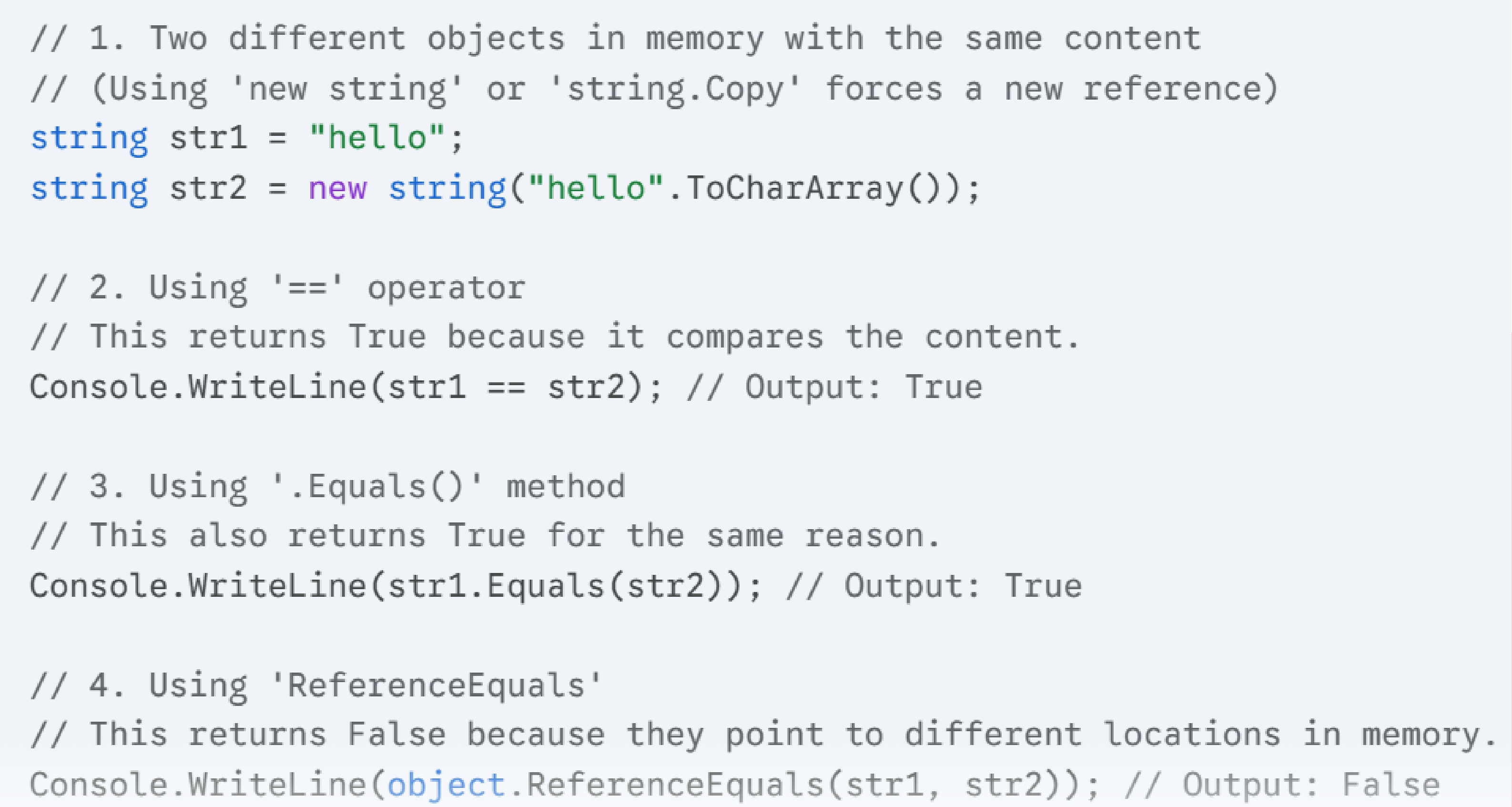
- Case sensitivity
- Culture (language rules)
- Ordinal (binary) comparison

STRING COMPARISON

In .NET, the == operator for the string class has been overloaded to compare the actual characters (the values) rather than the memory addresses (the references).

Sharad K Singh

STRING COMPARISON



```
// 1. Two different objects in memory with the same content
// (Using 'new string' or 'string.Copy' forces a new reference)
string str1 = "hello";
string str2 = new string("hello".ToCharArray());

// 2. Using '==' operator
// This returns True because it compares the content.
Console.WriteLine(str1 == str2); // Output: True

// 3. Using '.Equals()' method
// This also returns True for the same reason.
Console.WriteLine(str1.Equals(str2)); // Output: True

// 4. Using 'ReferenceEquals'
// This returns False because they point to different locations in memory.
Console.WriteLine(object.ReferenceEquals(str1, str2)); // Output: False
```

Sharad K Singh

STRING COMPARISON

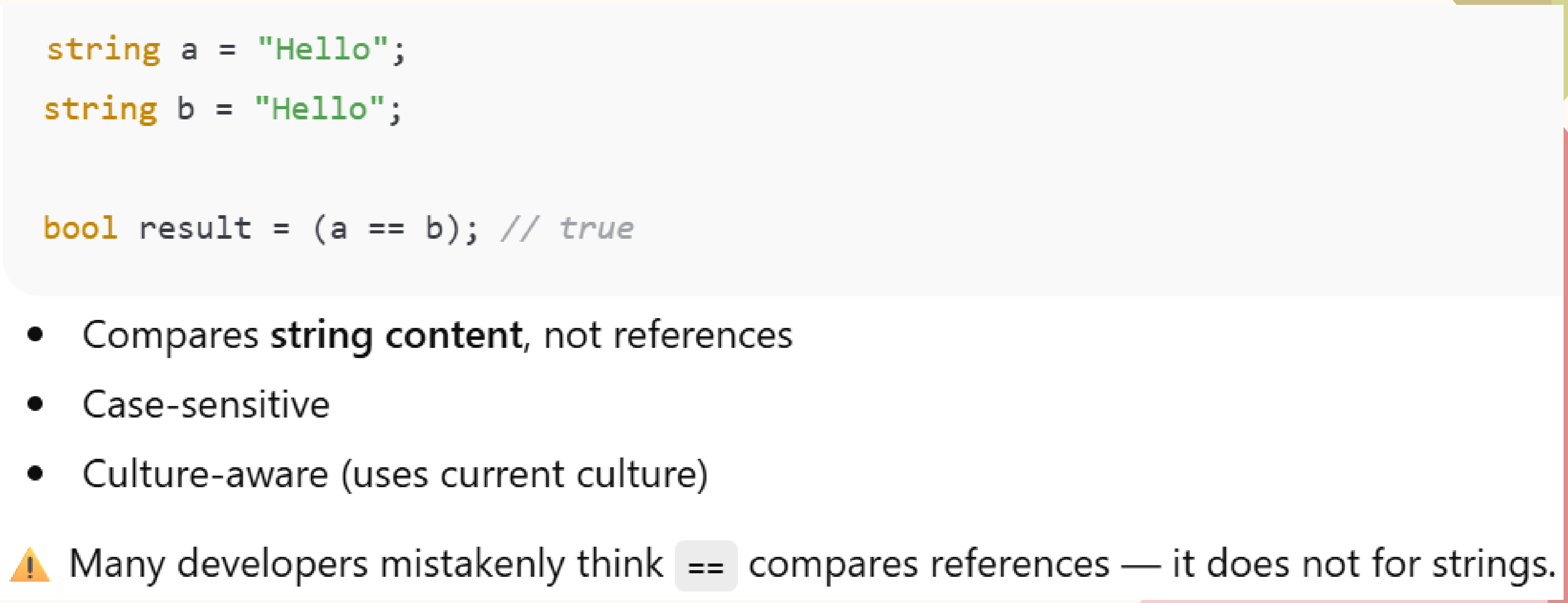
Equals() method

== operator

In .NET, == and .Equals() for strings often seem identical because the string class overloads both to perform **content comparison** rather than just checking memory addresses.

Sharad K Singh

STRING COMPARISON



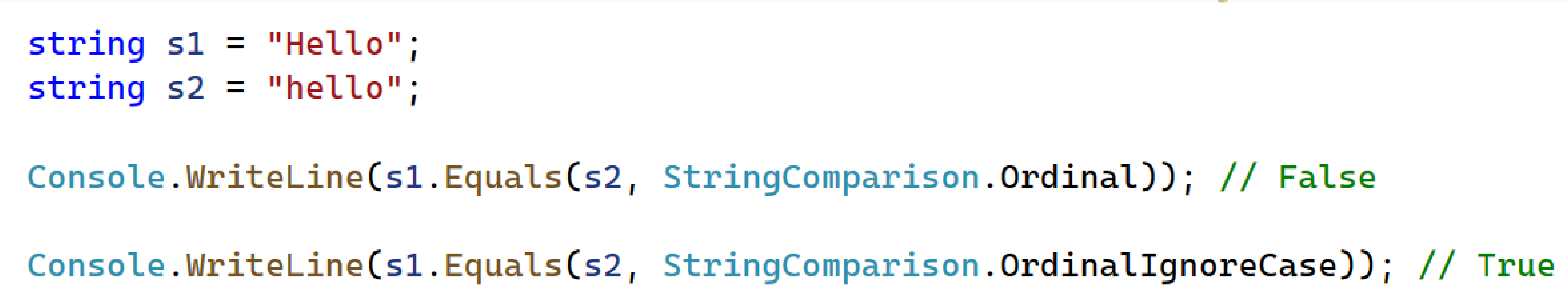
```
string a = "Hello";
string b = "Hello";

bool result = (a == b); // true
```

- Compares **string content**, not references
- Case-sensitive
- Culture-aware (uses current culture)

⚠️ Many developers mistakenly think `==` compares references — it does not for strings.

Sharad K Singh

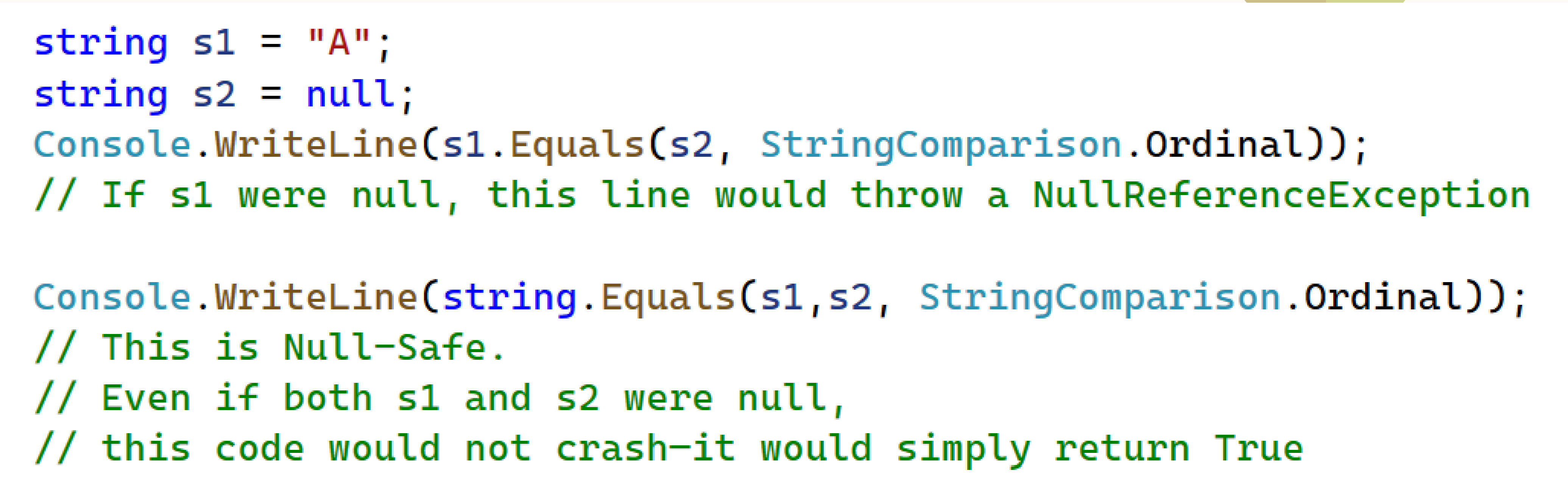
STRING EQUAL OVERLOAD



```
string s1 = "Hello";
string s2 = "hello";

Console.WriteLine(s1.Equals(s2, StringComparison.Ordinal)); // False

Console.WriteLine(s1.Equals(s2, StringComparison.OrdinalIgnoreCase)); // True
```

Sharad K Singh

STRING.EQUALS() STATIC



```
string s1 = "A";
string s2 = null;
Console.WriteLine(s1.Equals(s2, StringComparison.Ordinal));
// If s1 were null, this line would throw a NullReferenceException

Console.WriteLine(string.Equals(s1,s2, StringComparison.Ordinal));
// This is Null-Safe.
// Even if both s1 and s2 were null,
// this code would not crash-it would simply return True
```

SEARCHING STRINGS

Contains()

IndexOf() and LastIndexOf()

StartsWith()

EndsWith()

Sharad K Singh

STRING MANIPULATION

ToUpper() and ToLower()

Trim(), TrimStart(), TrimEnd()

Substring()

Replace()

Sharad K Singh

SPLIT AND JOIN

Split() for tokenizing text

Join() for combining strings

Commonly used in CSV and logs

Sharad K Singh

STRING FORMATTING

string.Format()

Interpolated strings ($"{value}")

Standard and custom format specifiers

Sharad K Singh

STRING VS STRINGBUILDER

String is immutable

StringBuilder is mutable

Use StringBuilder for loops and heavy concatenation

BEST PRACTICES

Avoid string concatenation inside loops

Use StringBuilder where appropriate

Be careful with culture-specific comparisons

Prefer ordinal comparisons for performance

METHODS

Code reuse

Modularity

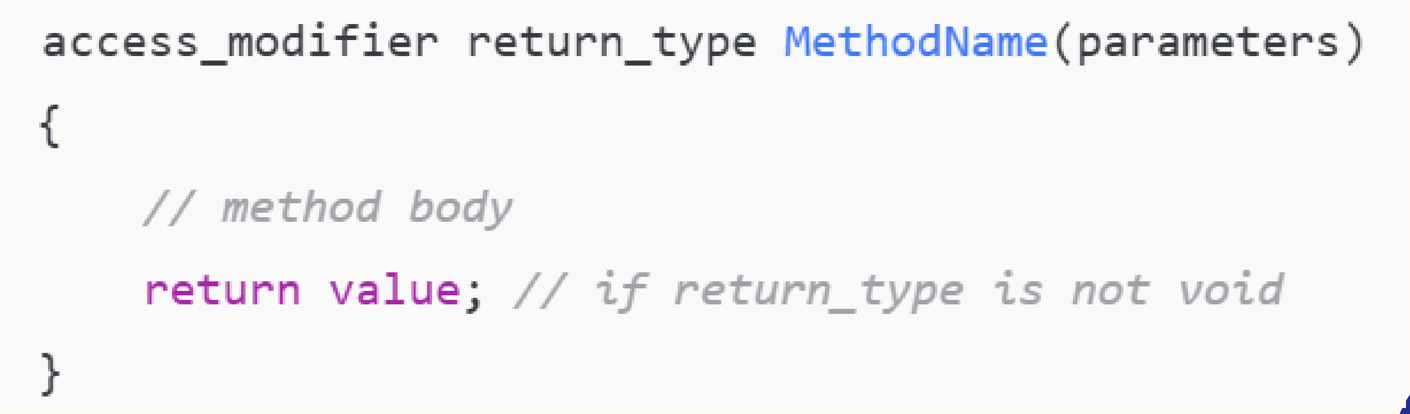
Readability

Testability

Maintenance

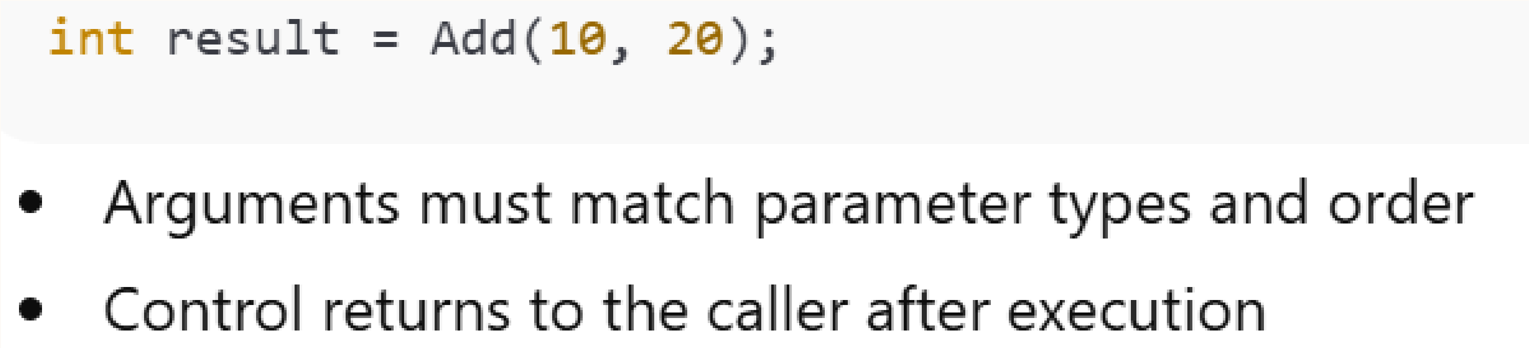
Sharad K Singh

METHODS



```
access_modifier return_type MethodName(parameters)
{
    // method body
    return value; // if return_type is not void
}
```

METHODS INVOCATION



```
int result = Add(10, 20);
```

- Arguments must match parameter types and order
- Control returns to the caller after execution

Sharad K Singh

METHOD TYPES

Instance Methods

Static Methods

Void vs Non-Void Methods

Sharad K Singh

PARAMETERS IN METHODS

Value Parameters (Default)

Reference Parameters (ref)

Output Parameters (out)

in Parameters (Read-Only Reference)

Sharad K Singh

METHOD OVERLOADING

Parameter count or type must differ

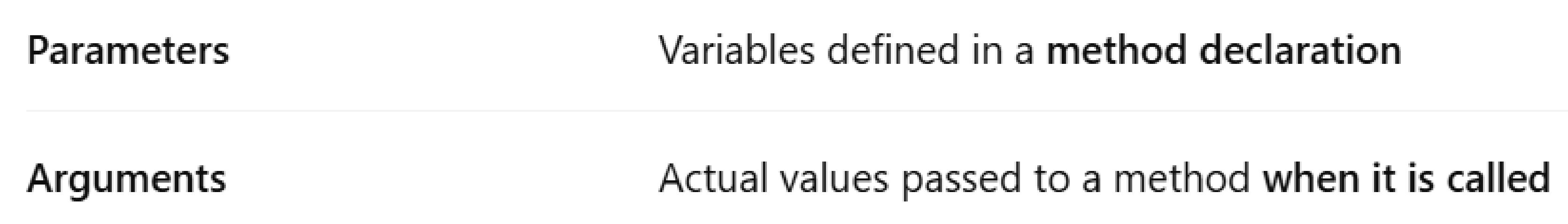
Return type alone cannot differentiate

Sharad K Singh

OPTIONAL AND NAMED PARAMETERS

Optional Parameters

Named Arguments



Parameters	Variables defined in a **method declaration**
Arguments	Actual values passed to a method **when it is called**

Sharad K Singh

THANK YOU

Sharad K Singh