String Manipulation and Regex in C# 10

Introducing Strings and Regex

Steve Gordon

.NET Engineer and Microsoft MVP

@stevejgordon   www.stevejgordon.co.uk

Overview

Strings in C#

The importance of immutability

String encoding basics

Introduce regular expressions (Regex)

Later in This Course

Comparing and sorting strings

Searching strings

Combining and formatting strings

Efficient string manipulation using a StringBuilder

Later in This Course

```
(^A.*)
```

Regular expressions

Version Check

This version was created by using:

- .NET 6.0
- C# 10
- Visual Studio 2022

Version Check

This course is 98% applicable to:

- .NET Core 3.1
- .NET 5.0
- .NET Framework
- Visual Studio 2013 to 2019
- Future .NET versions

Relevant Notes

A note on frameworks and libraries:

- A new version of .NET releases each year
- Strings are a stable and mature concept
- Microsoft aim to maintain backward compatibility between releases
- Most concepts shown in this course can be applied regardless of .NET version

Course Prerequisites

Beginner to intermediate knowledge of .NET

Experience with C# fundamentals

String Manipulation in C#: Best Practices

Steve Gordon

app.pluralsight.com/library/courses/string-manipulation-c-sharp-best-practices

Follow Along

Follow along: Download the exercise files

The solution requires the latest .NET 6.0.x SDK
http://dot.net

An IDE such as Visual Studio Community Edition or an editor such as Visual Studio Code

Let’s Get Started

Introducing Strings

System.String

“A string represents text as a sequence of UTF-16 code units.”

Strings are used to represent textual information such as a username or hostname.

INSERT HANDWRITING VIDEO

Д

أَلِف

私

Aa

ב

```
string myString = "hello"
```

Properties of Strings

Built-in type of .NET and C#

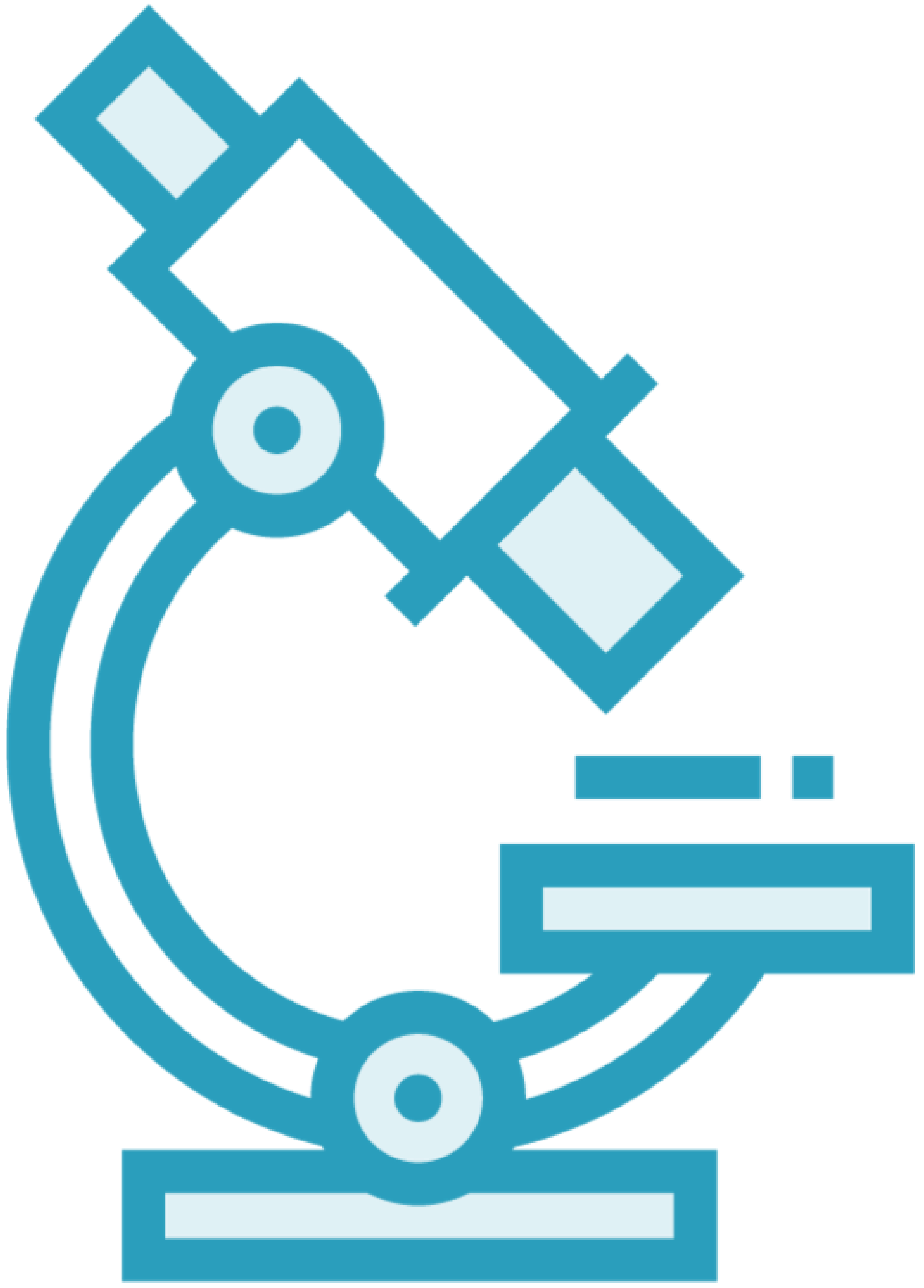
Their design is optimized for performance

Reference types stored on the heap

Have value-based equality semantics

Tightly integrated within the .NET CLR

Implemented with managed and unmanaged code



Immutability of Strings

Immutability

Internal state cannot be modified after initialization

Strings are immutable, read-only instances

Character buffer cannot grow

Methods and operators return a new string

Advantages

Strings can be safely shared without the need to defensively copy them

Easy to reason about

Thread-safe by default

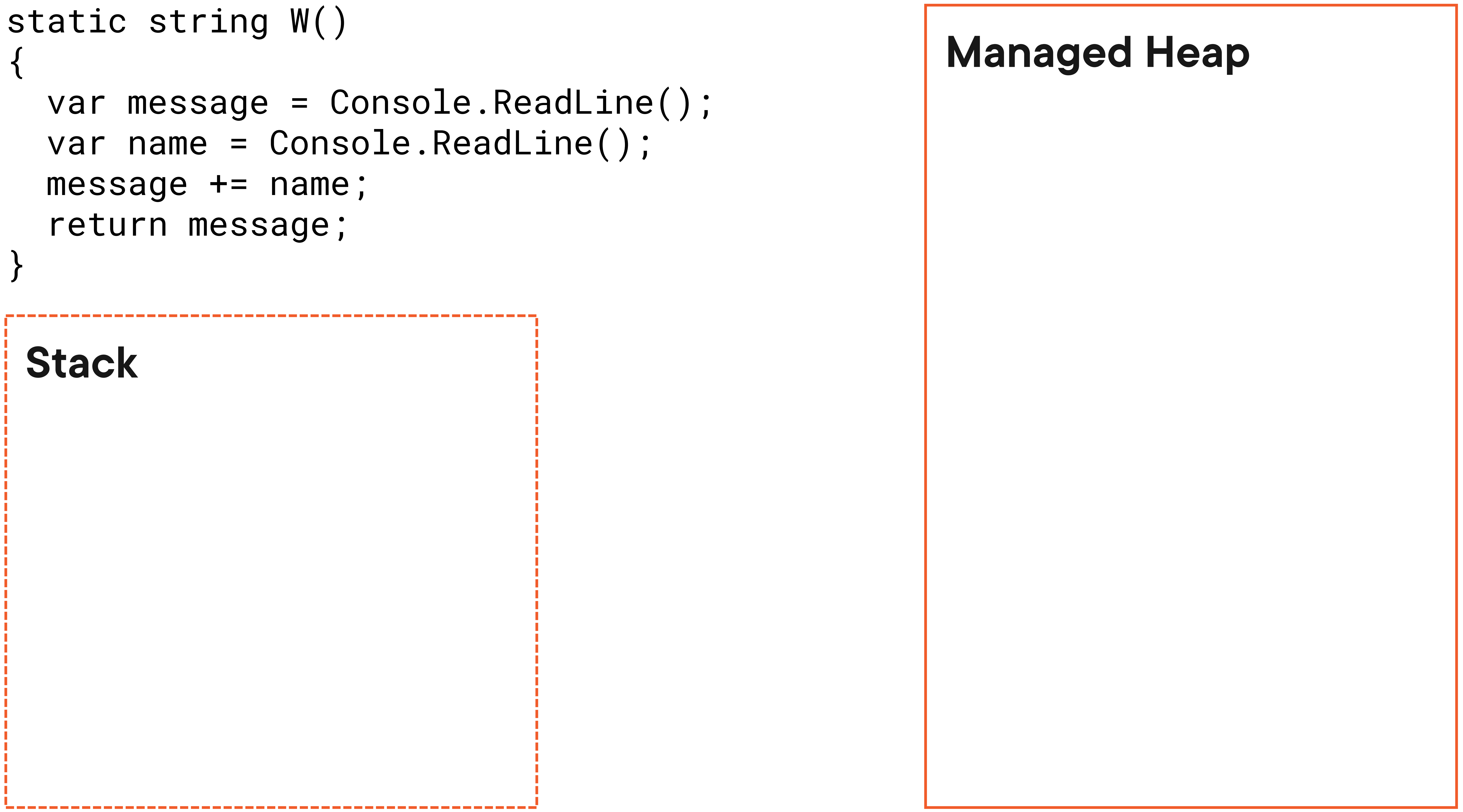
Can be optimized by the CLR

Immutability increases the number of objects in memory, which introduces a performance overhead.

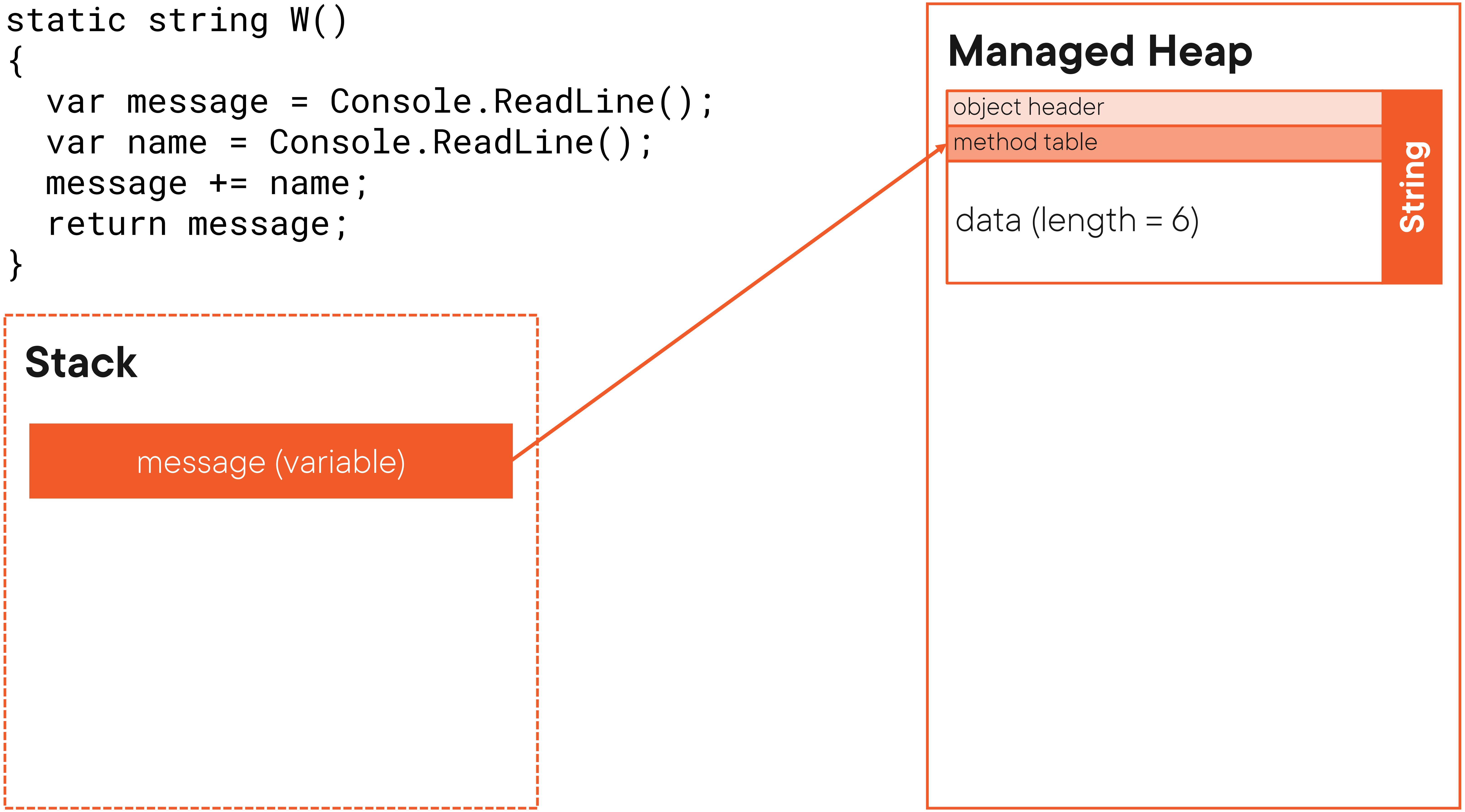
Important!

Remember that operations do not modify the original string and cause at least one new string to be created.

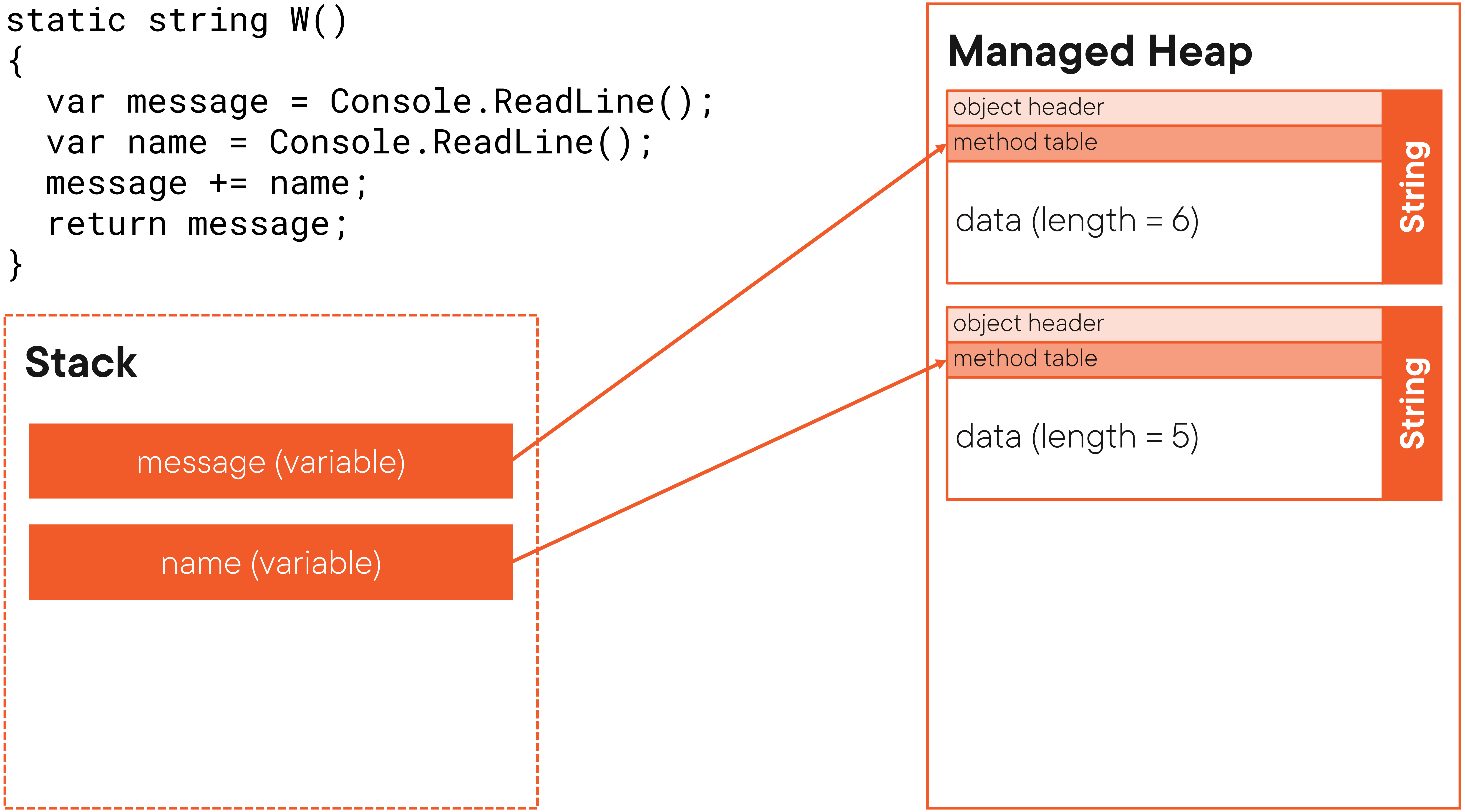
```
static string W()
{
  var message = Console.ReadLine();
  var name = Console.ReadLine();
  message += name;
  return message;
}
```



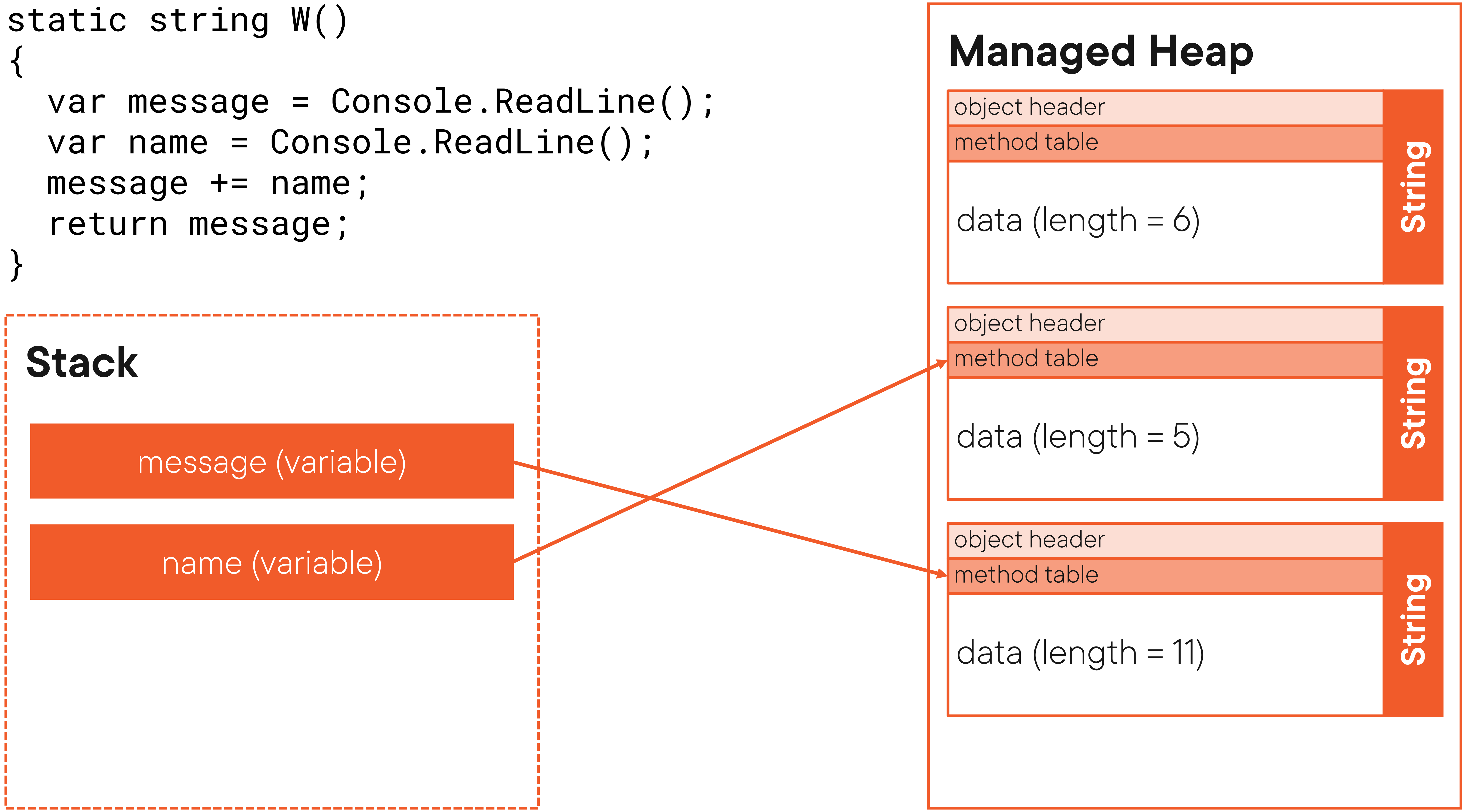
```
static string W()
{
  var message = Console.ReadLine();
  var name = Console.ReadLine();
  message += name;
  return message;
}
```



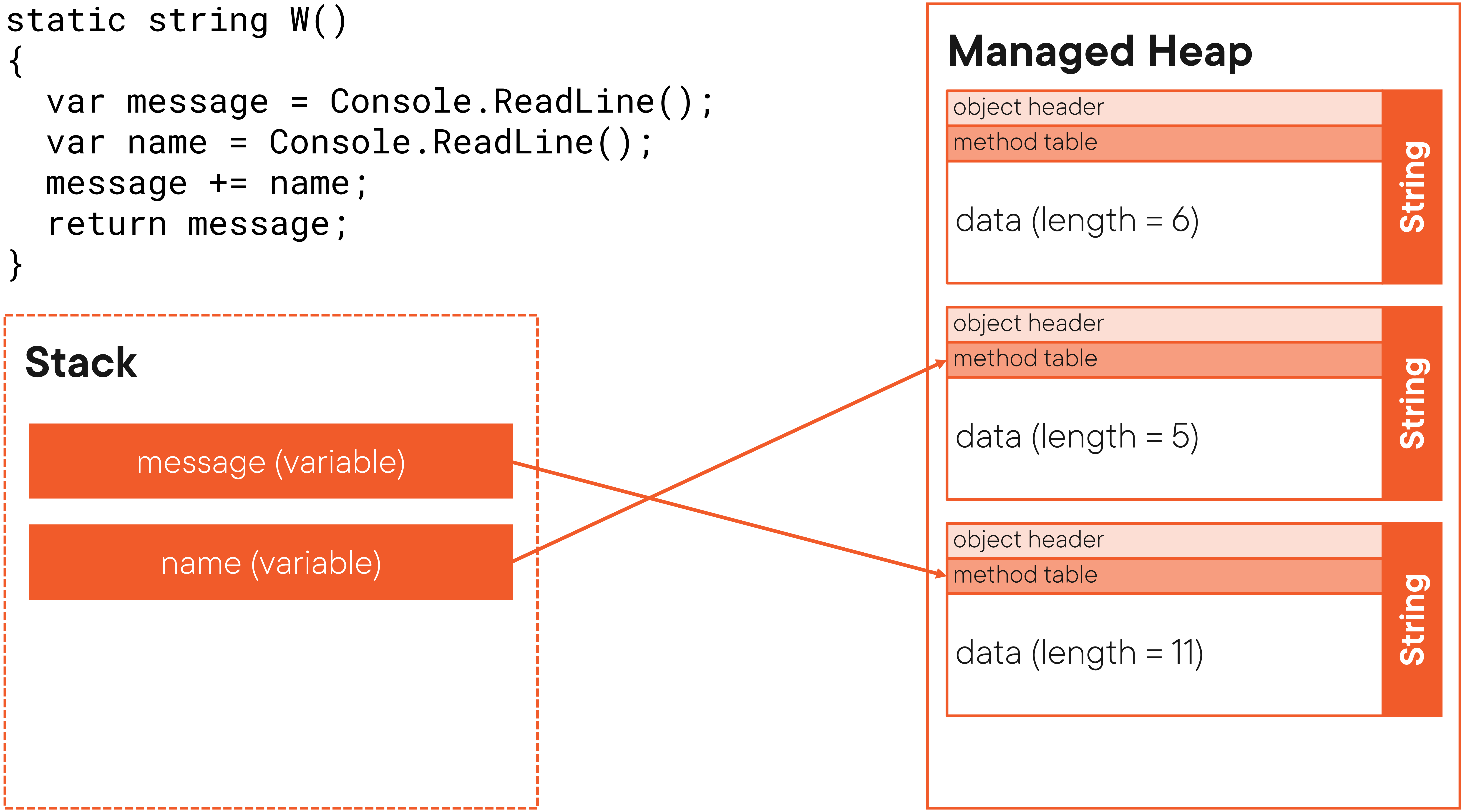
```
static string W()
{
  var message = Console.ReadLine();
  var name = Console.ReadLine();
  message += name;
  return message;
}
```



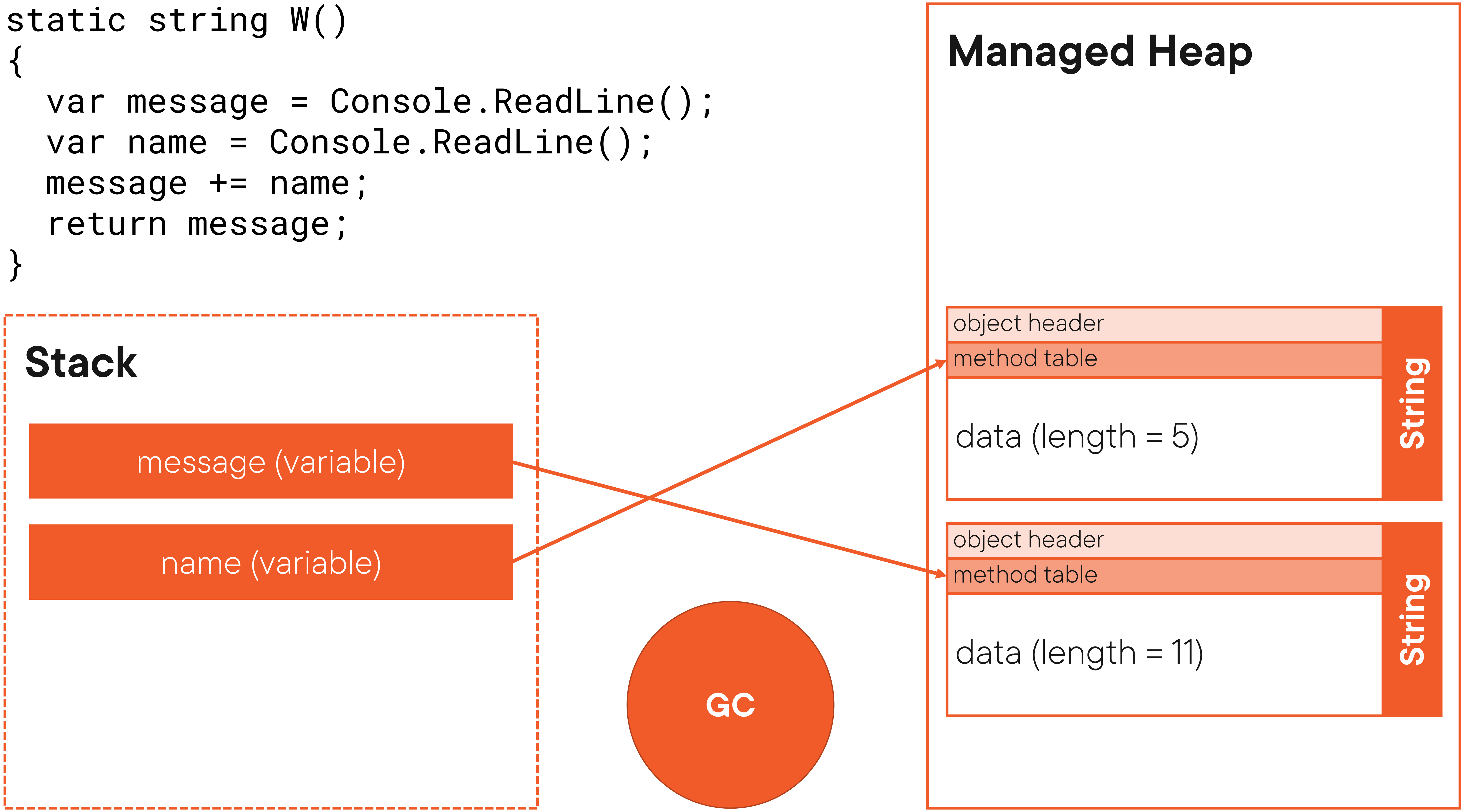
```
static string W()
{
  var message = Console.ReadLine();
  var name = Console.ReadLine();
  message += name;
  return message;
}
```



```
static string W()
{
  var message = Console.ReadLine();
  var name = Console.ReadLine();
  message += name;
  return message;
}
```

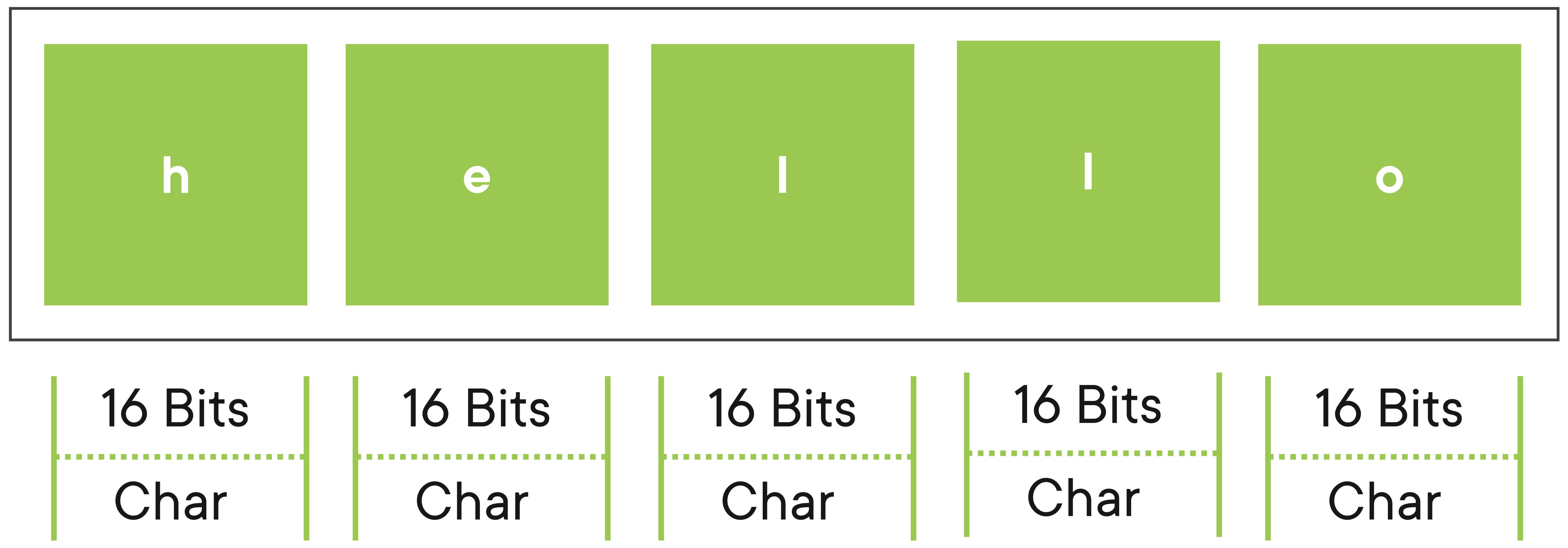


```
static string W()
{
  var message = Console.ReadLine();
  var name = Console.ReadLine();
  message += name;
  return message;
}
```

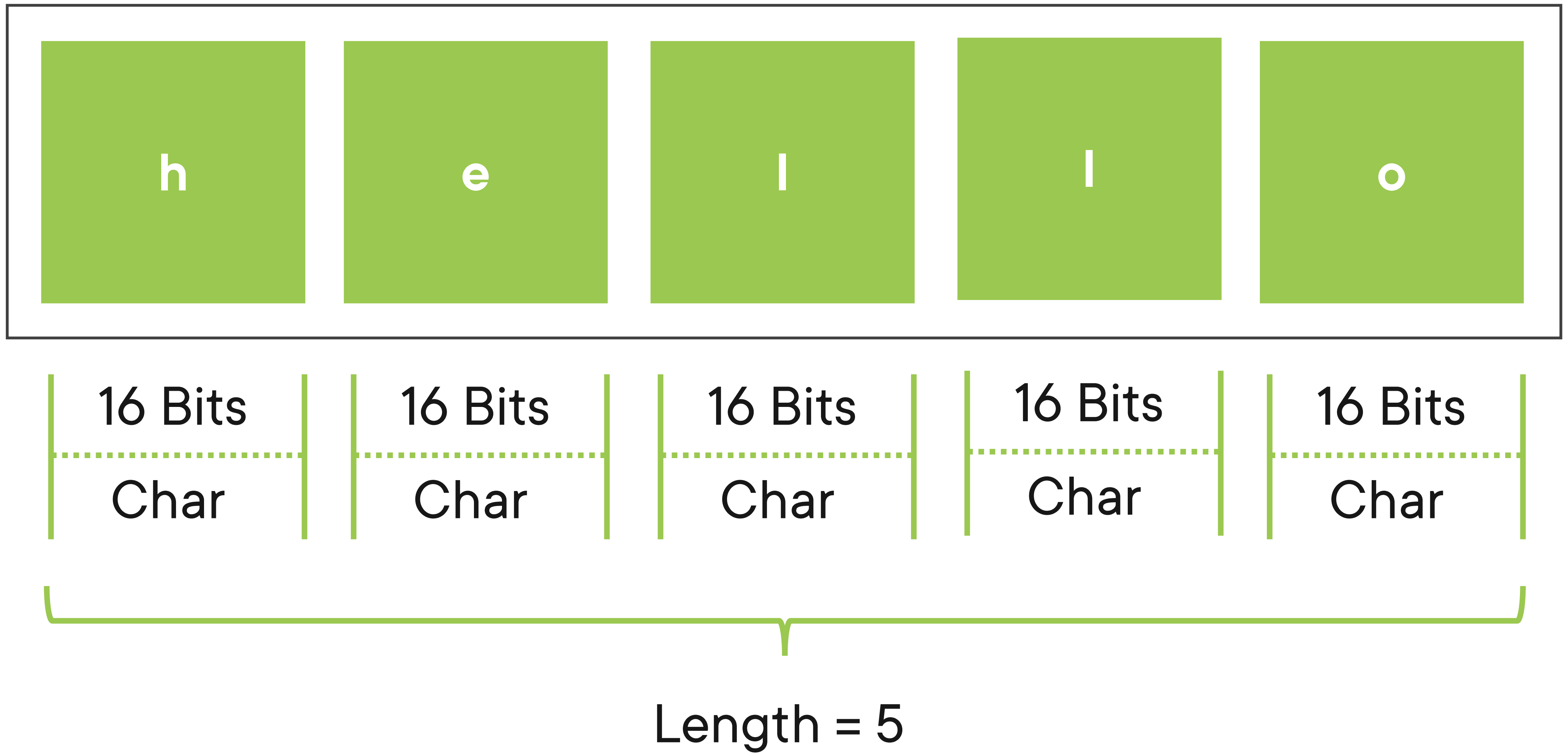


Character Encoding

Strings


h
e
l
l
o
16 Bits
Char
16 Bits
Char
16 Bits
Char
16 Bits
Char
16 Bits
Char

Strings


h
e
l
l
o
16 Bits
Char
16 Bits
Char
16 Bits
Char
16 Bits
Char
16 Bits
Char
Length = 5

Encoding

The means by which characters are mapped to bytes of binary data stored in memory, on disk or transmitted over a network.

ASCII

American Standard Code for Information Interchange

Developed in the 1960’s

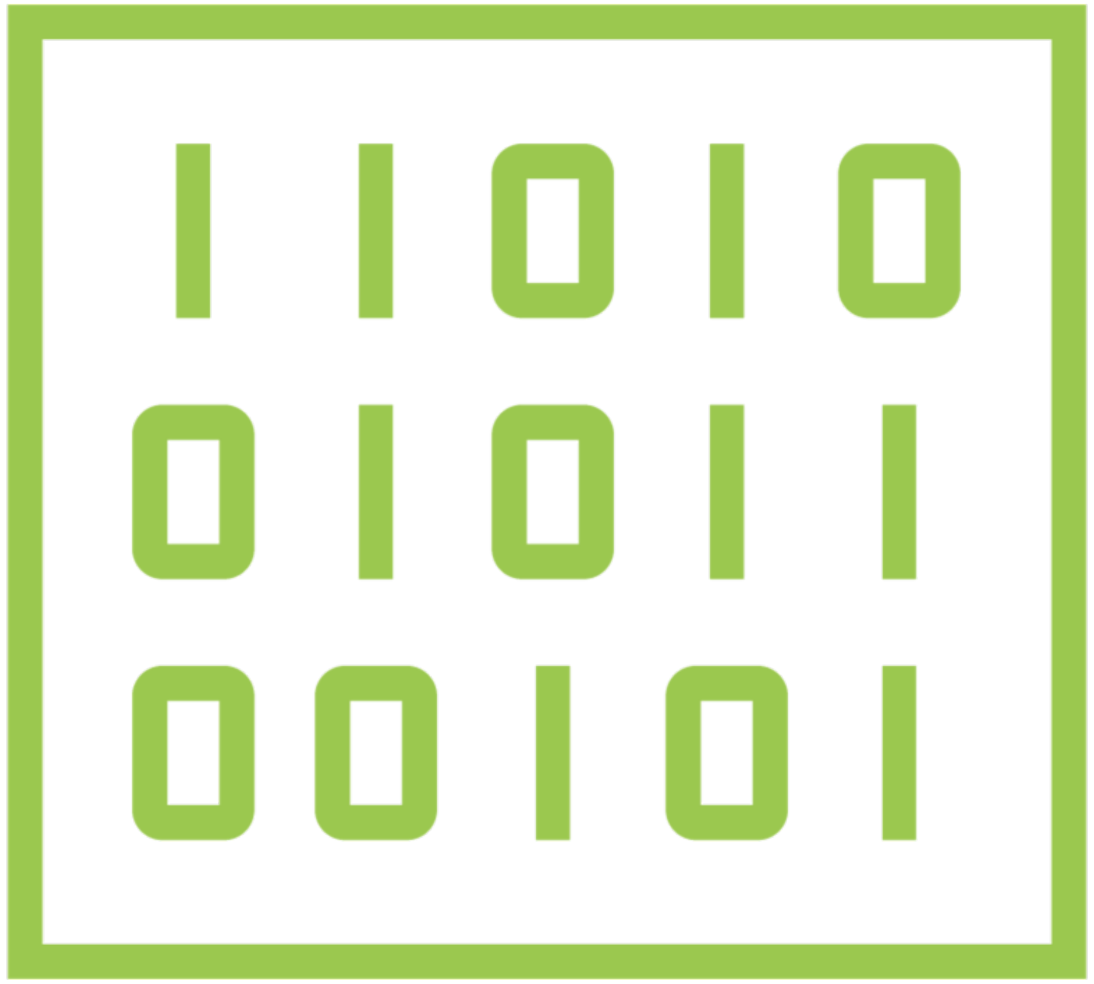
Uses 7-bits to encode up to 128 characters

Includes non-printing, control characters

Limited to only English characters

Used for HTTP/1 request line and headers

Unicode



Origins dating back to 1987

Uses a 16-bit character model

Original proposal had some limitations

- Version 2.0 removed the 16-bit limit with surrogate pairs

Continues to evolve with additional characters

Maintained by the Unicode Consortium

Contains 143,859 characters as of v13

Unicode Code Points

Integer value ranging from 0 to 1,114,111

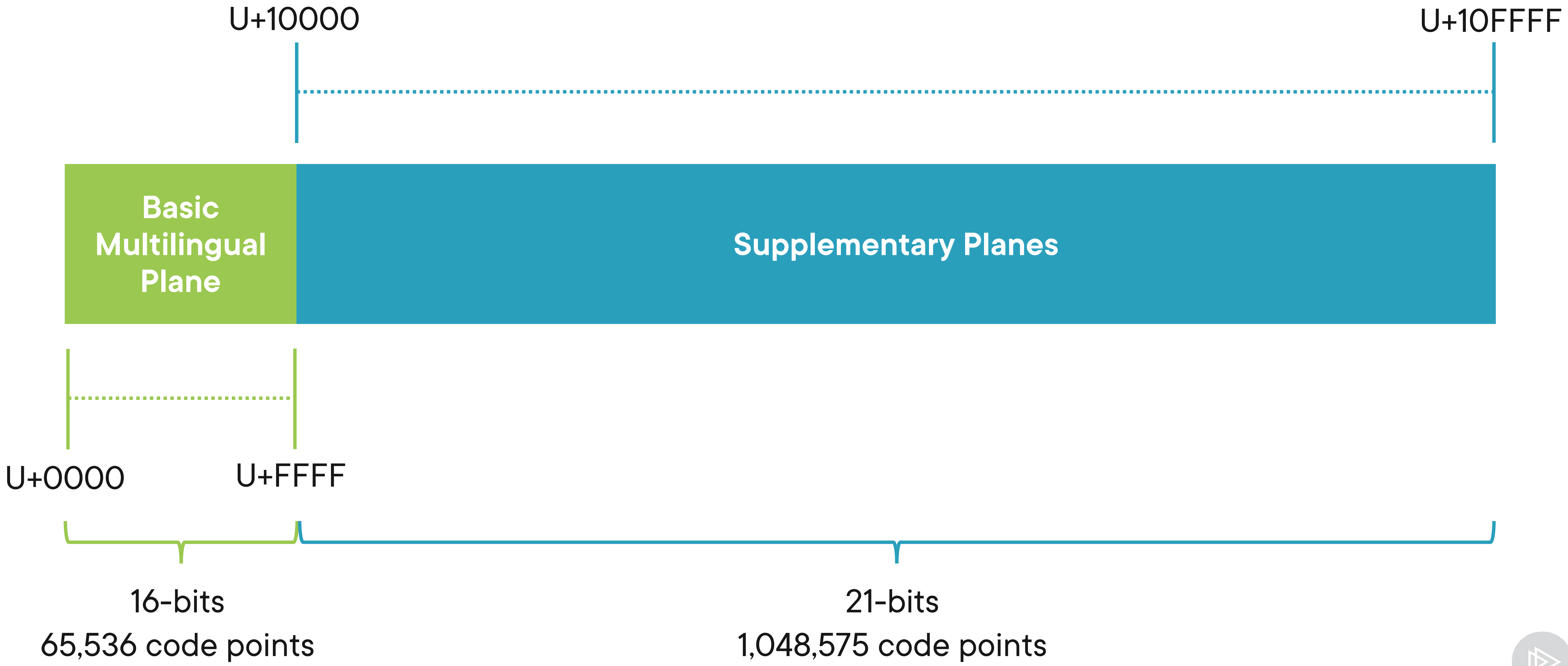
Assigned to letters, symbols and control characters

Many code points are reserved for future use

Code Point Examples

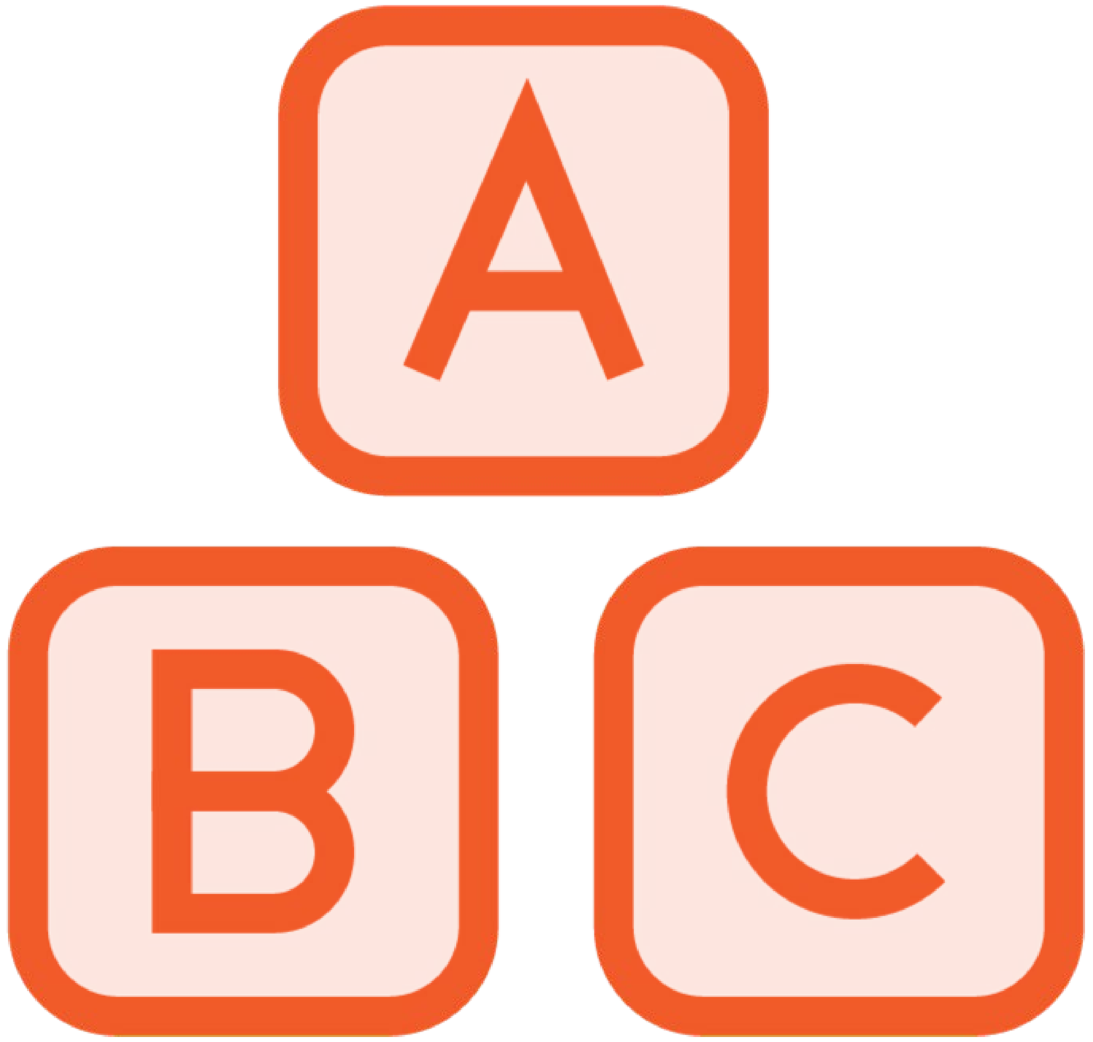
Decimal	Hex	Example	Description
13	U+000D	n/a	Carriage return
65	U+0041	A	Latin capital letter A
1590	U+0636	ض	Arabic letter Ḍād
128512	U+1F600	😀	Grinning face emoji

Code Point Ranges



A Unicode Transformation Format (UTF) encodes code points to and from binary data.

UTF-16 Encoding



.NET uses UTF-16 encoding for strings

Requires at least two bytes per character (char)

Higher code points may require two chars

- Known as a surrogate pair
- Requires four bytes

UTF-8 and the Internet

UTF-8 is the most prevalent encoding on the web. Most sites accept UTF-8 encoded requests and return UTF-8 encoded data (HTML/JSON) in responses.

UTF-8 Encoding

Variable width character encoding

Can encode all Unicode code points

Length varies between 1 and 4 bytes

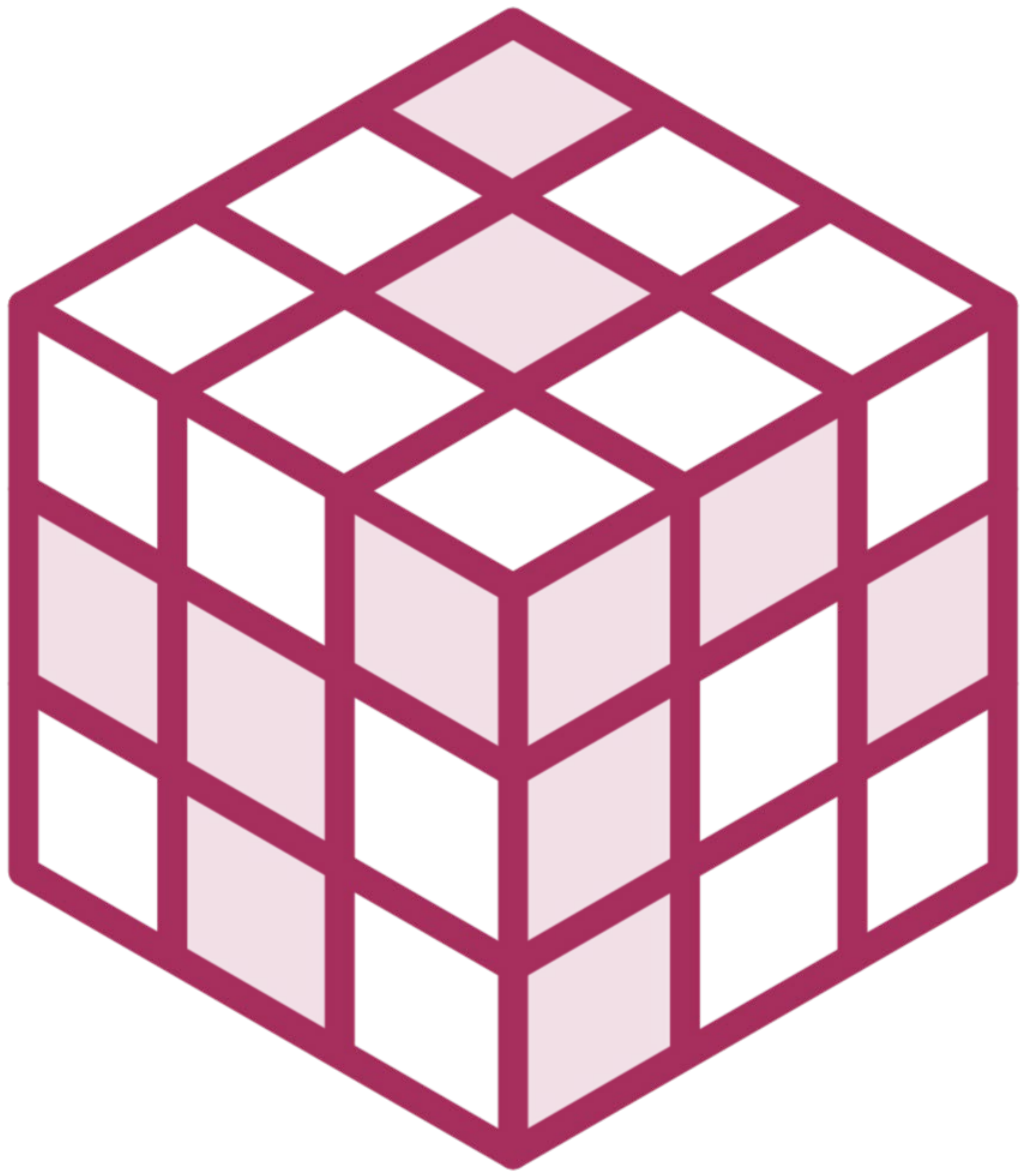
- 1 byte: Sufficient for US-ASCII
- 2 bytes: Supports an extra 1,920 characters
- 3 bytes: Sufficient for the whole BMP range
- 4 bytes: Supports the supplementary range

For most text, UTF-8 results in less data to transmit, vs. UTF-16 encoding

Variable Length Encoding

A	41	1-byte
Ä	C3-84	2-bytes
诶	E8-AF-B6	3-bytes
😀	F0-9F-98-80	4-bytes

.NET Encoding Classes



.NET includes classes to work with different encoding schemes

Derive from System.Text.Encoding

Encoding classes are accessible from static properties (e.g. ASCII and UTF8)

Encode characters/strings to bytes

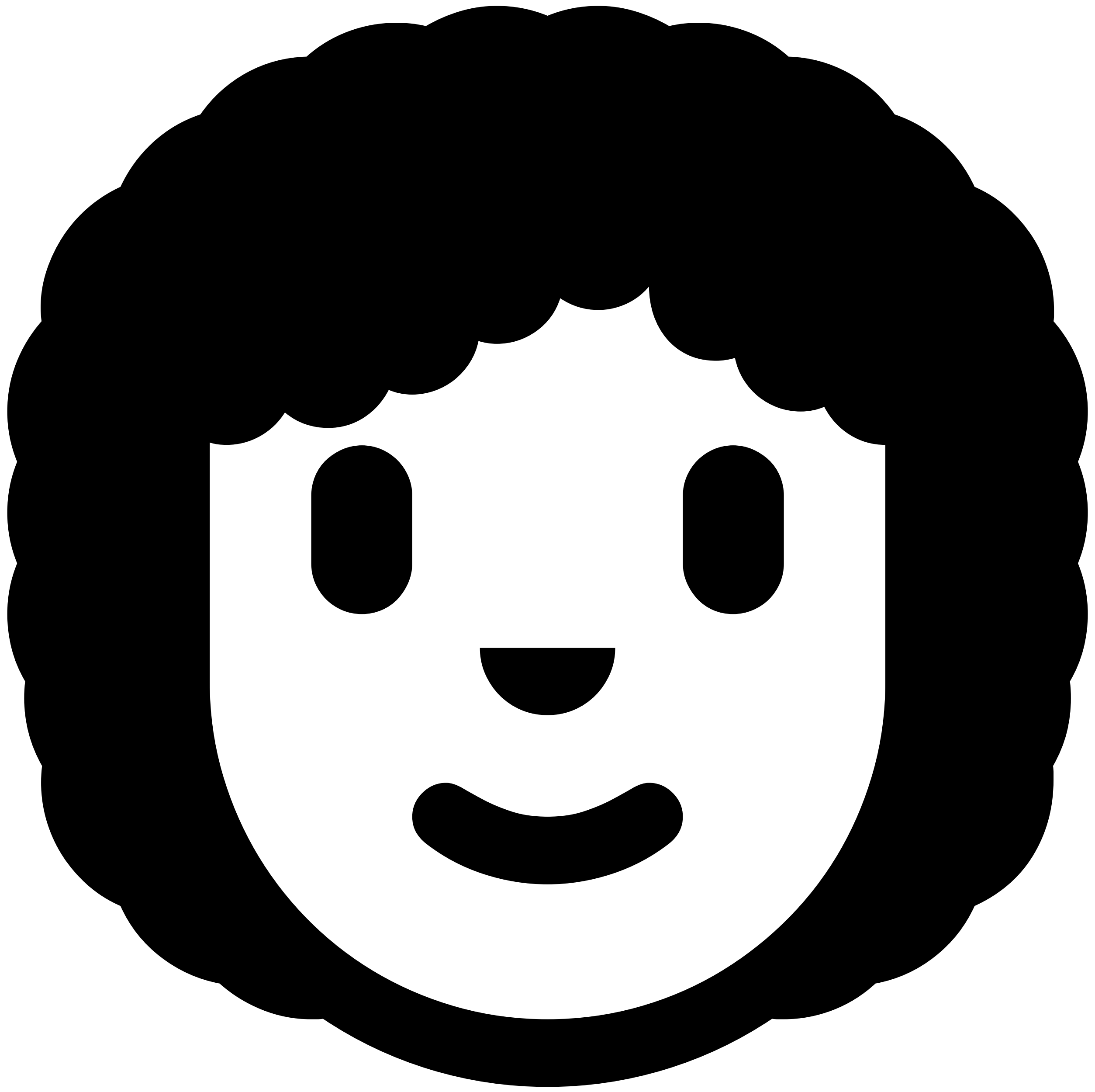
Decode from bytes into characters/strings

```csharp
string myString = "Treble Clef: 𝄞";

int length = myString.Length; // 15

byte[] utf16Bytes = System.Text.Encoding.Unicode.GetBytes(myString);

int utf16Length = utf16Bytes.Length; // 30

byte[] utf8Bytes = System.Text.Encoding.UTF8.GetBytes(myString);

int utf8Length = utf8Bytes.Length; // 17

string roundTrippedString = System.Text.Encoding.UTF8.GetString(utf8Bytes);

bool equal = myString == roundTrippedString; // true
```

Using the System.Text Encoding Classes

This sample code encodes a UTF-16 string to UTF-16 and UTF-8 bytes and then decodes the UTF-8 bytes back into a string representation (roundtripping).

Grapheme Cluster



Woman: Medium Skin Tone, Curly Hair

Combines:

Woman

Medium Skin Tone

Zero Width Joiner

Curly Hair

```
string womanEmoji = "👩🏻‍🦱"; // NOTE: VS does not render this as a single element

int womanEmojiLength = myString.Length; // 7

byte[] womanEmojiBytes = System.Text.Encoding.Unicode.GetBytes(womanEmoji);

int womanEmojiBytesLength = womanEmojiBytes.Length; // 14

var womanEmojiInfo = new System.Globalization.StringInfo(womanEmoji);

int womanEmojiTextElements = womanEmojiInfo.LengthInTextElements; // 1
```

String Length

The length of a string is the number of UTF-16 chars that it contains. This may not always align to the number of visible text elements.

Introducing Regular Expressions

Regular Expressions

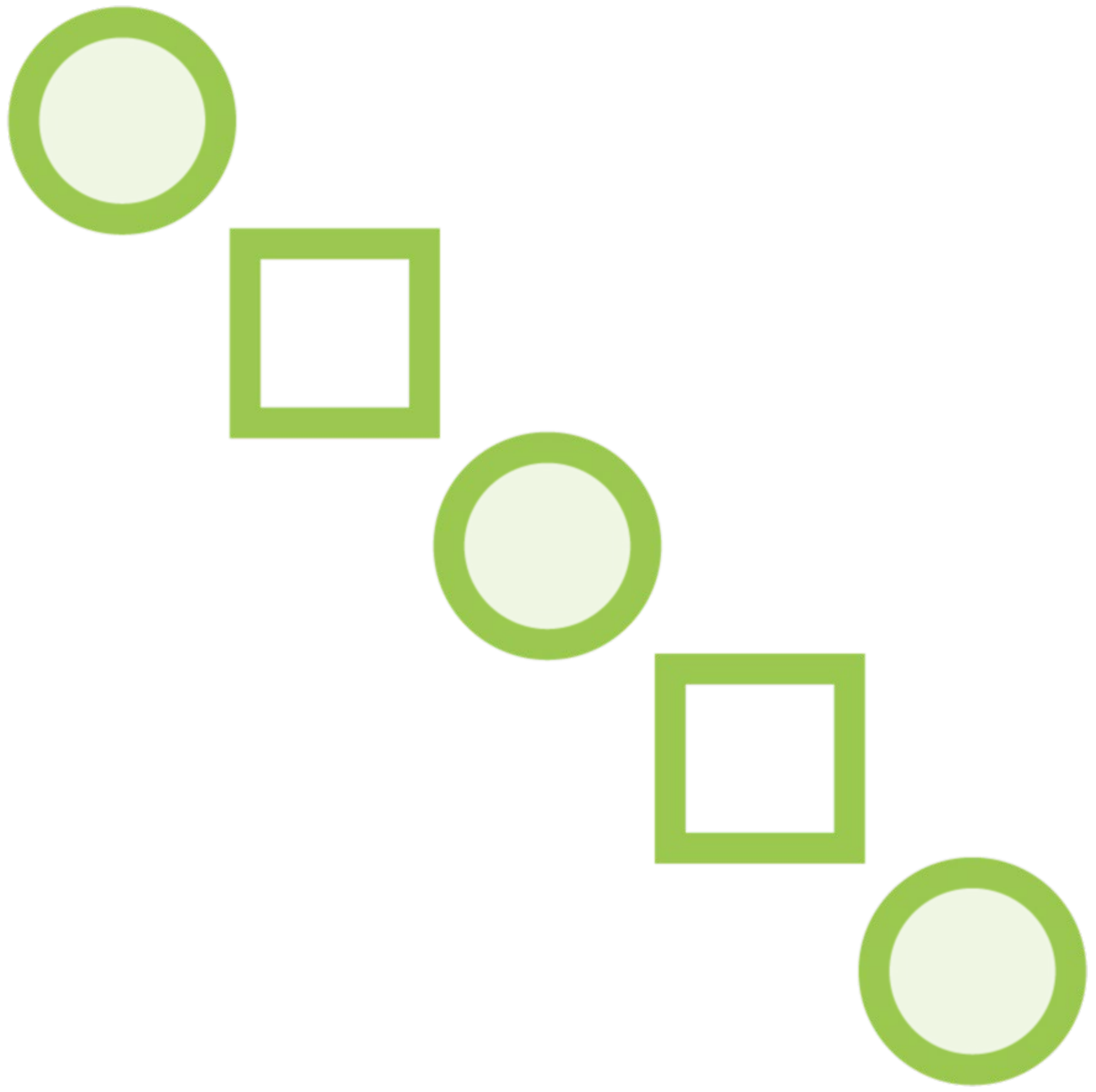
An often indispensable tool for parsing complex strings.

Validating text input

Complex search matches

Removing or replacing portions of strings

Regex Patterns



Formed by a sequence of characters

Characters can represent:

- A literal character
- A metacharacter

Regex patterns can be complex and difficult to understand

Regex patterns can be used across languages

Regex Engines

Central component for processing regular expressions

.NET uses a backtracking, non-deterministic finite automaton (NFA) engine

Backtracking allows the engine to save state

- Offers greater control over string matching
- Matches can run slowly if they require lots of backtracking

Pattern Example

```
string pattern = "^A.*";
```

Character	Description
^	Positional anchor - Requires the match to start at the beginning of the string.
A	The literal Latin uppercase letter A.
.	Matches any single character except \n.
*	Greedy quantifier – Match the previous token zero or more times.

Preferring static Regex methods is recommended because it benefits from caching and may perform better.

```
string input = "A sample string.";
string pattern = "^A.*";
bool match = Regex.IsMatch(input, pattern);
```

Matching a Regular Expression

The IsMatch method indicates whether the regular expression finds a match in the input string.

IsMatch Results

```
string pattern = "^A.*";
```

Input	Result
“a sample string.”	False
“A string. Another string”	True
“The string. A string”	False
“A”	True

IsMatch Results

```
string pattern = "^A.+";
```

Input	Pattern “^A.+”	Pattern “^A.*”
“A sample string.”	True	True
“a sample string.”	False	False
“A string. Another string”	True	True
“The string. A string”	False	False
“A”	False	True

Simplified Pattern

```
string pattern = "^A.";
```

Input	Regex.IsMatch(…)	Regex.Match(…).Value
“A sample string.”	True	“A ”
“a sample string.”	False	N/A
“A string. Another string”	True	“A ”
“The string. A string”	False	N/A
“A”	False	N/A

Regular Expressions are a deep topic.

https://docs.microsoft.com/en-us/dotnet/standard/base-types/regular-expression-language-quick-reference

Microsoft documentation

Regular Expression Fundamentals

Juliette Reinders Folmer
app.pluralsight.com/library/courses/
regular-expressions-fundamentals

Introducing the Sample Domain

GLOBOMANTICS

Business Requirements

Console application to read, parse and process source data

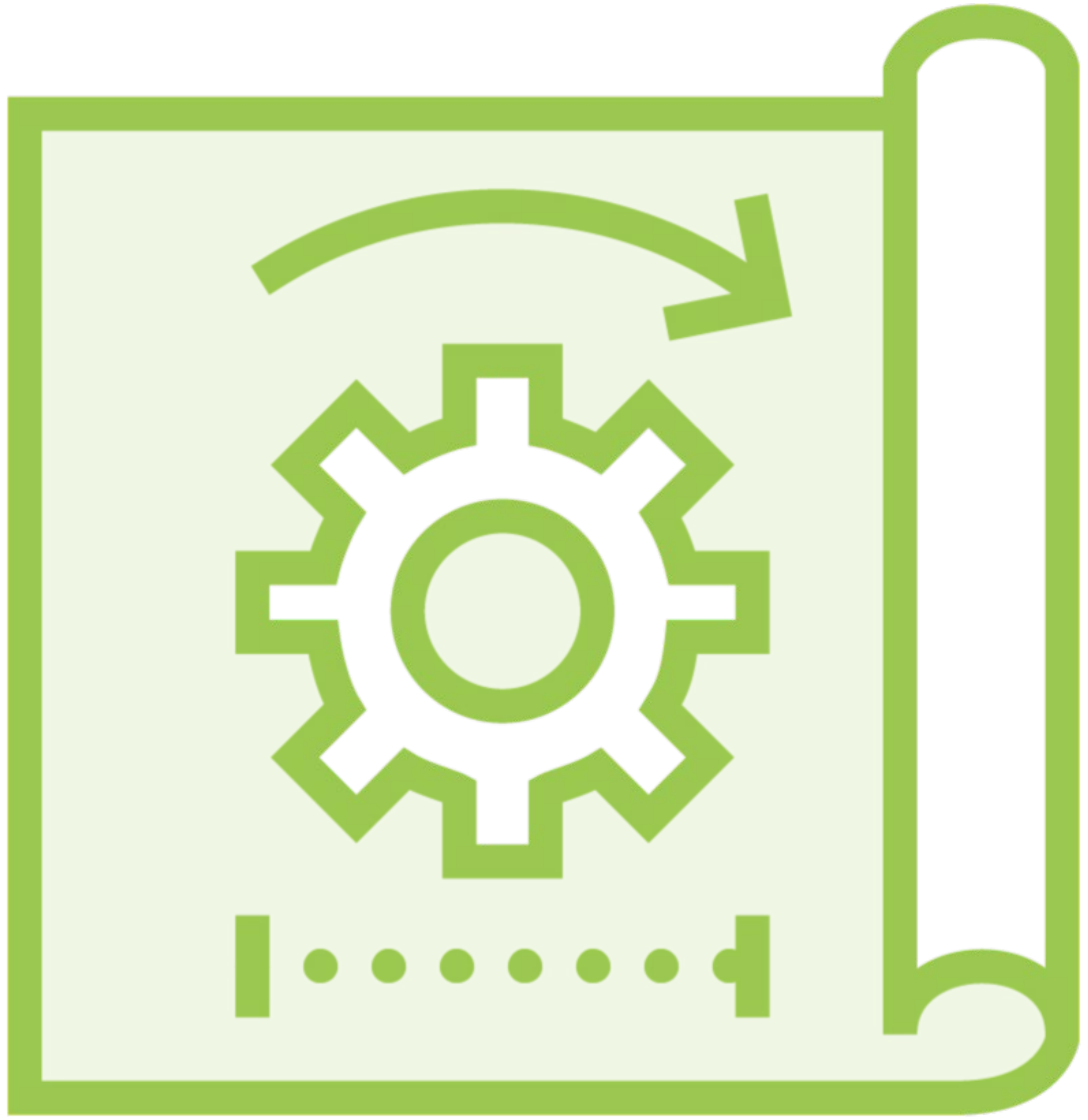
Ingest historical data from legacy systems

Source data is text-based

Data formats vary across source data files

Data consistency and validity is not guaranteed

Plan of Attack



Use sample data files during initial development phase

Build a prototype data processing application

Tackle various scenarios from the business requirements

String Manipulation in C#: Best Practices

Steve Gordon

app.pluralsight.com/library/courses/string-manipulation-c-sharp-best-practices

Demo

Introducing the sample application

Up Next: Working with Strings