C# Fundamentals

Principles of Computer Programming I

Spring/Fall 20XX

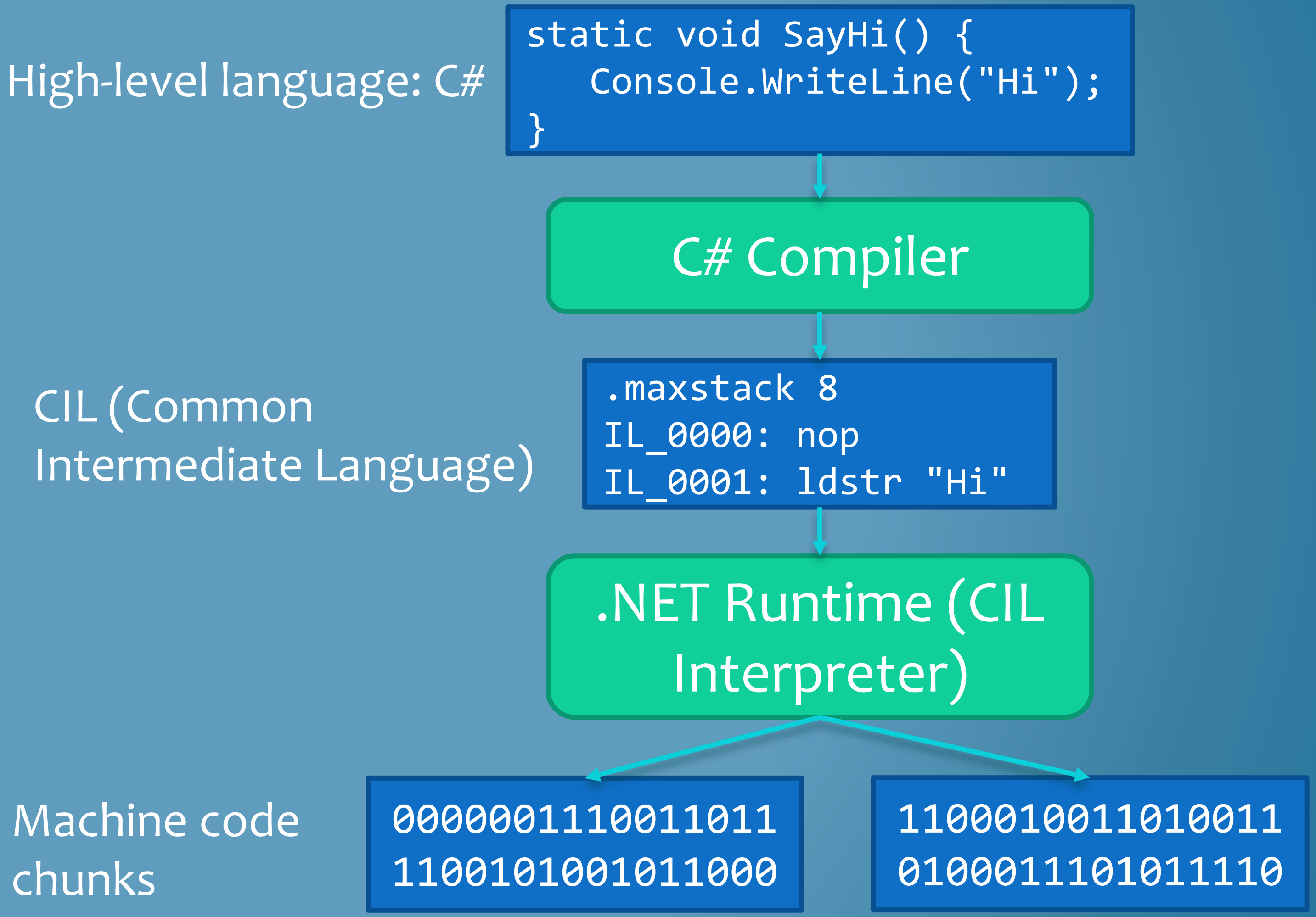
AUGUSTA UNIVERSITY

Outline

- C# Introduction
- Example C# program
- Rules vs. Conventions of C#
- Reserved words and identifiers
- Console.Write and Console.WriteLine
- Escape sequence

C#: A Managed Language

- More portable and safe than C or C++, older "unmanaged" languages
- Similar to Java
  - JVM is the runtime for Java Bytecode, the IL
- Comes with large standard library (.NET Framework)


High-level language: C#
C# Compiler
CIL (Common Intermediate Language)
.NET Runtime (CIL Interpreter)
Machine code chunks

```
static void SayHi() {
   Console.WriteLine("Hi");
}
```

```
.maxstack 8
IL_0000: nop
IL_0001: ldstr "Hi"
```

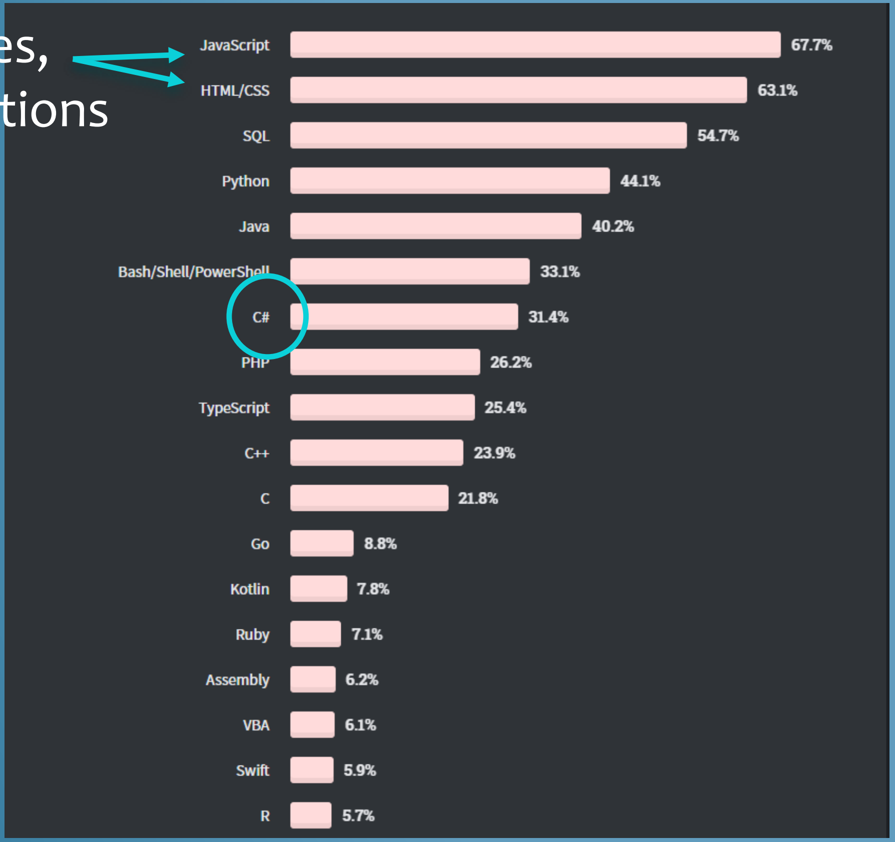
```
0000001110011011
1100101001011000
```

```
1100010011010011
0100011101011110
```

C#: Widespread and Popular

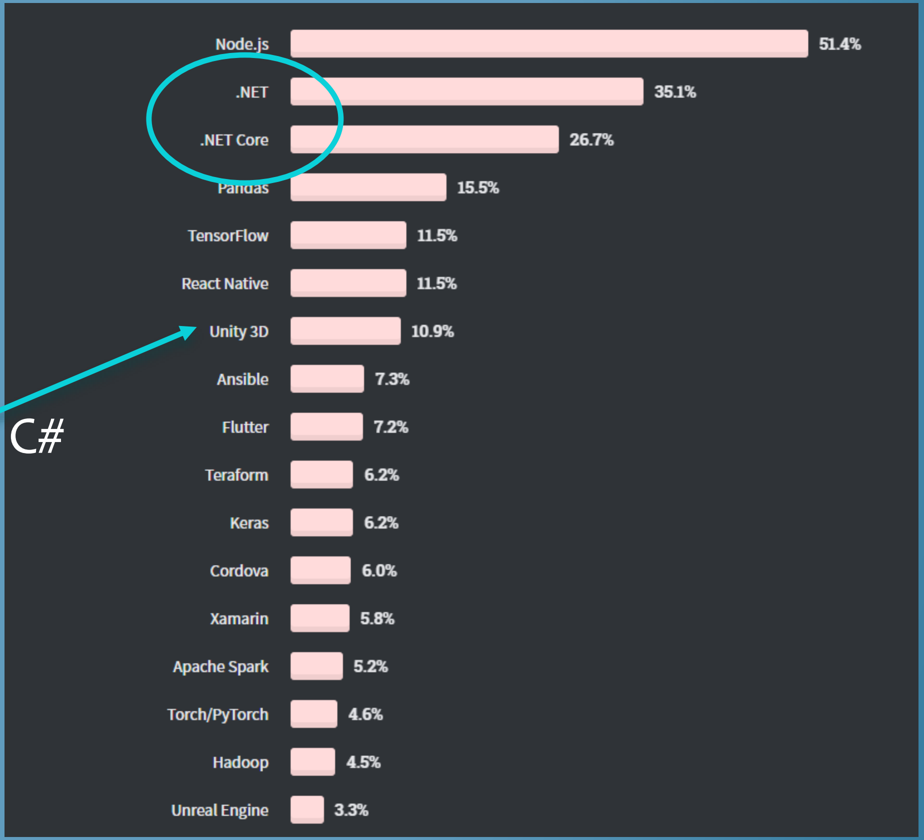
Most Used Languages 2020

For websites, not applications



Most Used Libraries/Frameworks

Written in C#

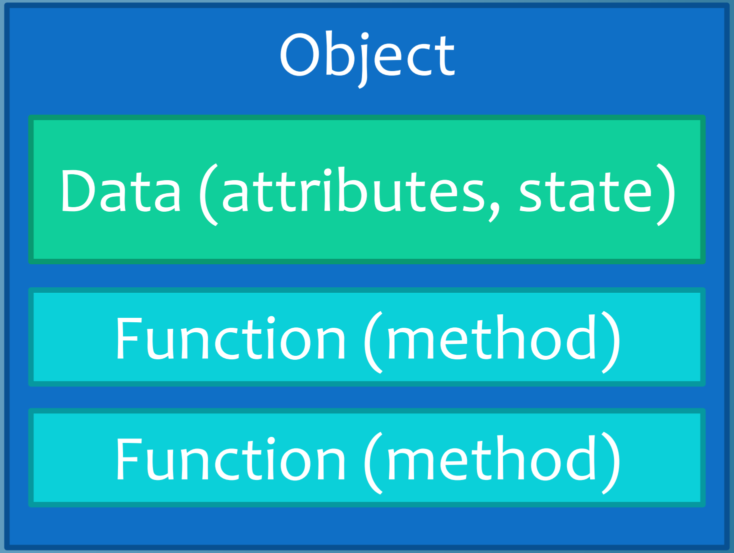


StackOverflow Developer Survey 2020

AUGUSTA UNIVERSITY

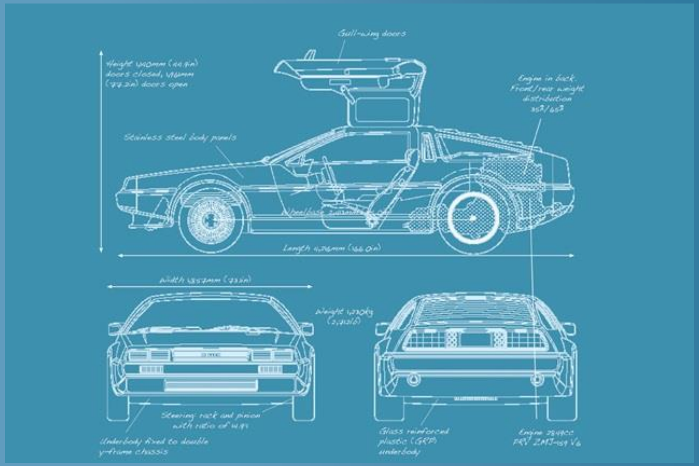
C#: Object-Oriented

- Paradigms for programming languages: Functional, procedural, logical, object-oriented, event-based
  - Philosophies for organizing code and expressing ideas in code
- C# is primarily object-oriented
- Program mostly consists of *objects*, reusable modules of code
  - Set of data (attributes)
  - Functions related to that data (methods)



Object Terminology

- Class = blueprint, template for object
  - Code that describes an object
- Object = single instance of class
  - Running code, with specific values/state
  - Each object is a “copy” of the class
- Method = function that modifies object
  - Defined in class, but executed on specific object (usually)
- Attribute = data stored in object



AUGUSTA UNIVERSITY

Object Examples

Car object:

- Attributes:
  - Engine status (off, idle, accelerating)
  - Transmission/gear position
- Methods:
  - Press/release pedals
  - Turn steering wheel
  - Shift transmission

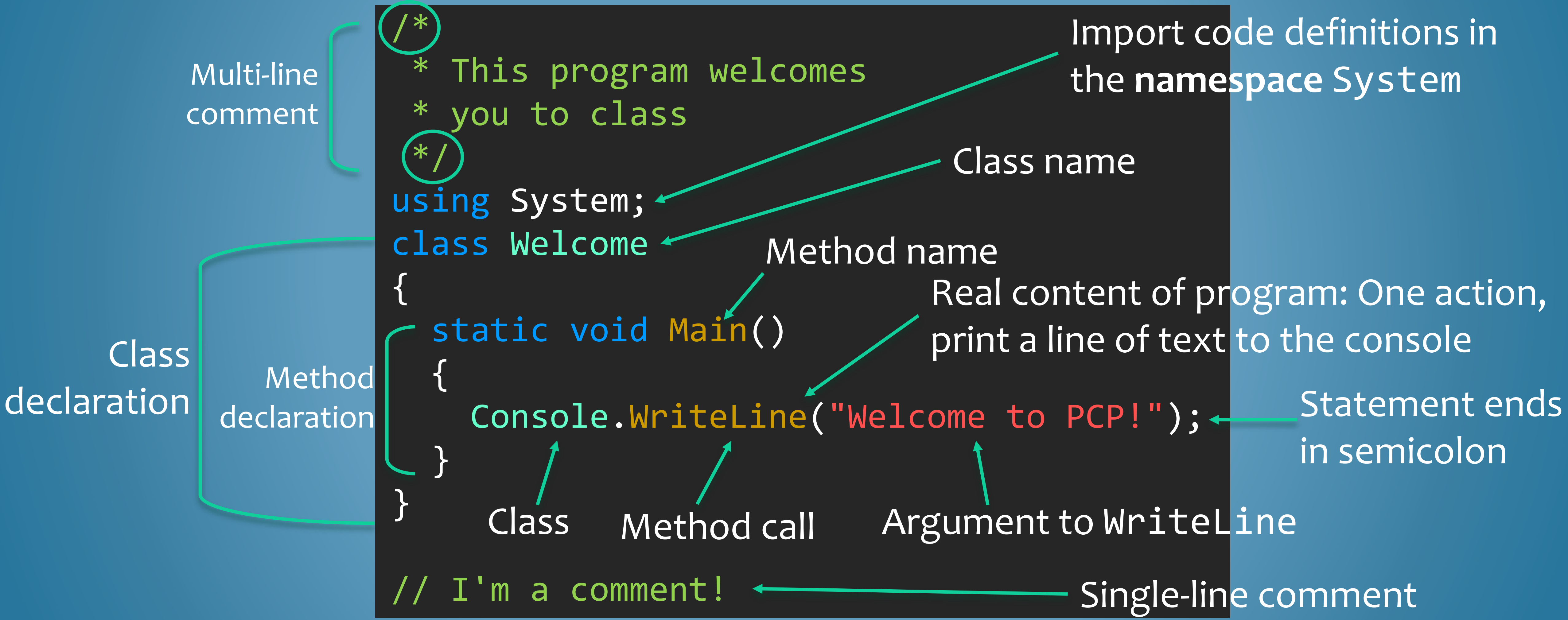
Audio object:

- Attributes:
  - Sound data
  - Current playback position
  - Target playback device
- Methods:
  - Play, pause, stop
  - Fast forward, rewind

Outline

- C# Introduction
- **Example C# program**
- Rules vs. Conventions of C#
- Reserved words and identifiers
- Console.Write and Console.WriteLine
- Escape sequence

A Simple Program



```
/*
 * This program welcomes
 * you to class
 */
using System;
class Welcome
{
  static void Main()
  {
    Console.WriteLine("Welcome to PCP!");
  }
}

// I'm a comment!
```

Multi-line comment
Import code definitions in the **namespace** System
Class name
Method name
Real content of program: One action, print a line of text to the console
Class declaration
Method declaration
Statement ends in semicolon
Class
Method call
Argument to WriteLine
Single-line comment

AUGUSTA UNIVERSITY

Outline

- C# Introduction
- Example C# program
- **Rules vs. Conventions of C#**
- Reserved words and identifiers
- Console.Write and Console.WriteLine
- Escape sequence

Basic C# Syntax Rules

- Each **statement** must end in a semicolon ;
    - Class and method declarations are not statements – they contain statements
- All words are case-sensitive
    - A class named `welcome` is **not** the same as one named `Welcome`
- Braces { } and parentheses ( ) must be matched
    - Finish what you start: once you start a class definition with {, you must end it with }

More C# Rules

- Whitespace – spaces, tabs, newlines – has (almost) no meaning
  - **Unless** it is within string data, like `"Welcome to PCP!"`
  - Must have 1 space between words

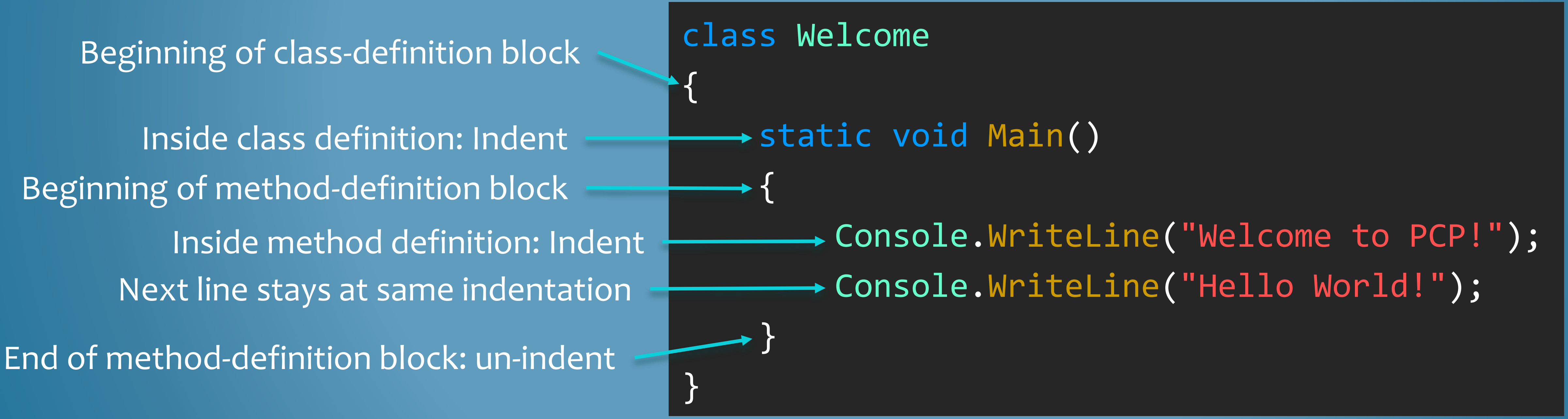
```
using System;class Welcome{static void
Main(){Console.WriteLine("Welcome to PCP!");}}
```

Same program as before

  - Note: Colors also don't matter – they're added by Visual Studio
- All C# applications must have a method named `Main`
  - When the application starts, the `Main` method is the first code to run

C# Conventions

- Not enforced by compiler, but expected by humans
- Indentation: Each time you start a block with {, indent 4 spaces



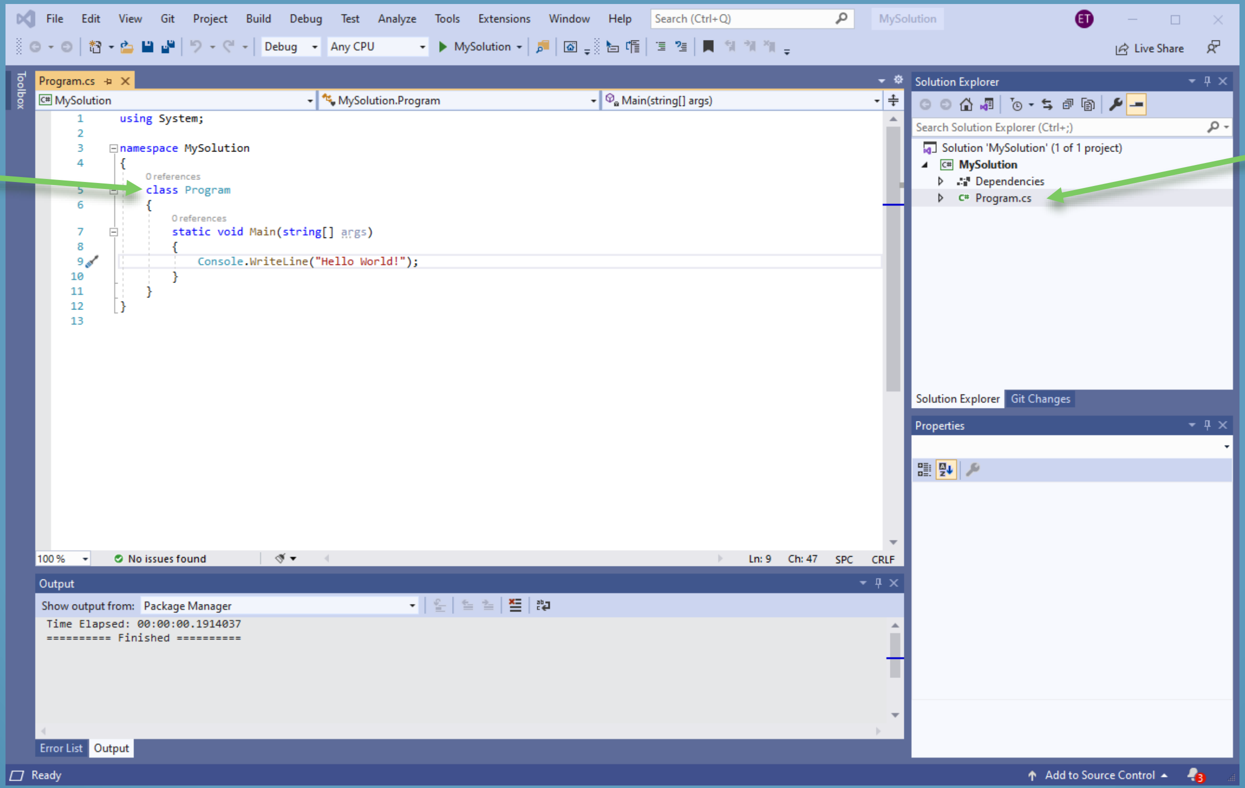
```
class Welcome
{
    static void Main()
    {
        Console.WriteLine("Welcome to PCP!");
        Console.WriteLine("Hello World!");
    }
}
```

Beginning of class-definition block
Inside class definition: Indent
Beginning of method-definition block
Inside method definition: Indent
Next line stays at same indentation
End of method-definition block: un-indent

C# Conventions

- Each .cs file contains one class
- The .cs file has the same name as the class

Class Program



```
using System;

namespace MySolution
{
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

```
Time Elapsed: 00:00:00.1914037
========== Finished ==========
```

File program.cs

AUGUSTA UNIVERSITY

Outline

- C# Introduction
- Example C# program
- Rules vs. Conventions of C#
- **Reserved words and identifiers**
- Console.Write and Console.WriteLine
- Escape sequence

AUGUSTA UNIVERSITY

Reserved Words in C#

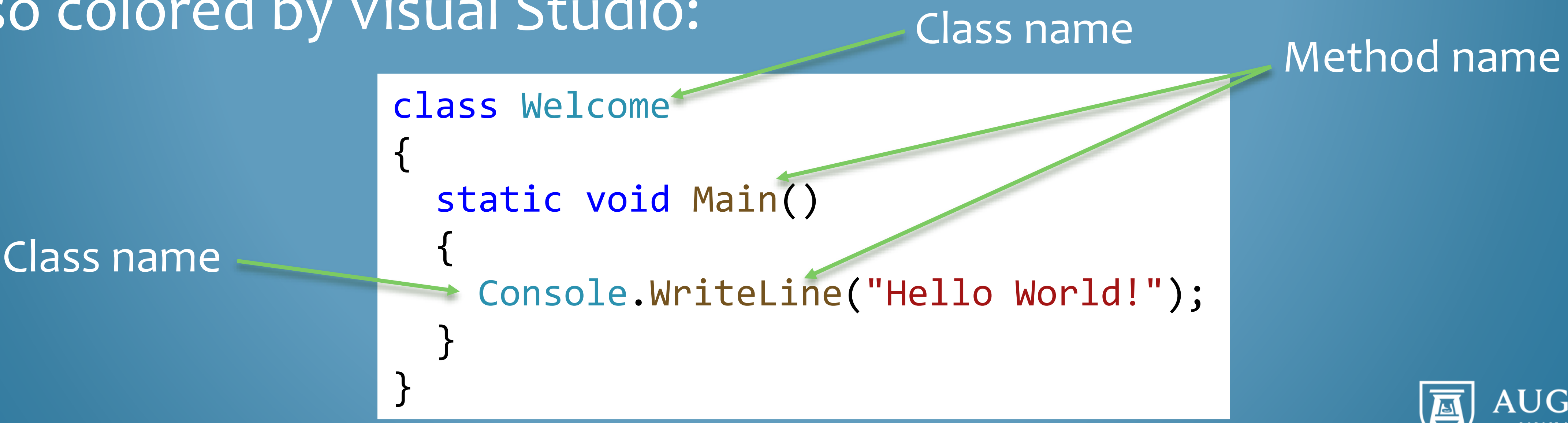
- Keywords in the language, colored blue by Visual Studio

`using`	`if`	`bool`	`string`
`class`	`else`	`byte`	`object`
`public`	`for`	`short`	`double`
`private`	`while`	`long`	`float`
`namespace`	`do`	`void`	`decimal`
`this`	`return`	`int`	`char`

- Can only be used for one specific purpose; cannot be changed or re-used as a name for something

Identifiers in C#

- Names a programmer chooses
- For classes, variables, methods, namespaces, etc.
- Some have already been chosen by other programmers: System, Console, WriteLine…
- Also colored by Visual Studio:



```
class Welcome
{
  static void Main()
  {
    Console.WriteLine("Hello World!");
  }
}
```

Identifier Rules

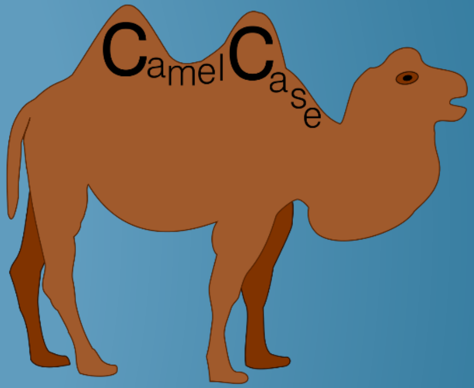
- Must not be a reserved word
- Must contain only letters, numbers, and _ (underscore)
  - No spaces
- Must not begin with a number
- Remember: Case sensitive, like everything in C#
- Are these valid identifiers?

```
My_class   class1   class   Class   thisClass   this
```

AUGUSTA UNIVERSITY

Identifier Conventions

- Should be descriptive
  - `AudioFile` is a better class name than `a`
- Should be easy to read and type
  - `ClAsS` is valid, but not a good idea
- Multi-word names should use CamelCase
- Class and method names should start with capitals: `AudioFile`
  - A.k.a. UpperCamelCase or PascalCase
- Variable names should start with lowercase: `myFile`



Outline

- C# Introduction
- Example C# program
- Rules vs. Conventions of C#
- Reserved words and identifiers
- **Console.Write and Console.WriteLine**
- Escape sequences

The WriteLine Function

Code in welcome.cs

```
class Welcome
{
  static void Main()
  {
    Console.WriteLine("Hello World!");
  }
}
```

Program output in terminal

```
Hello World!
```

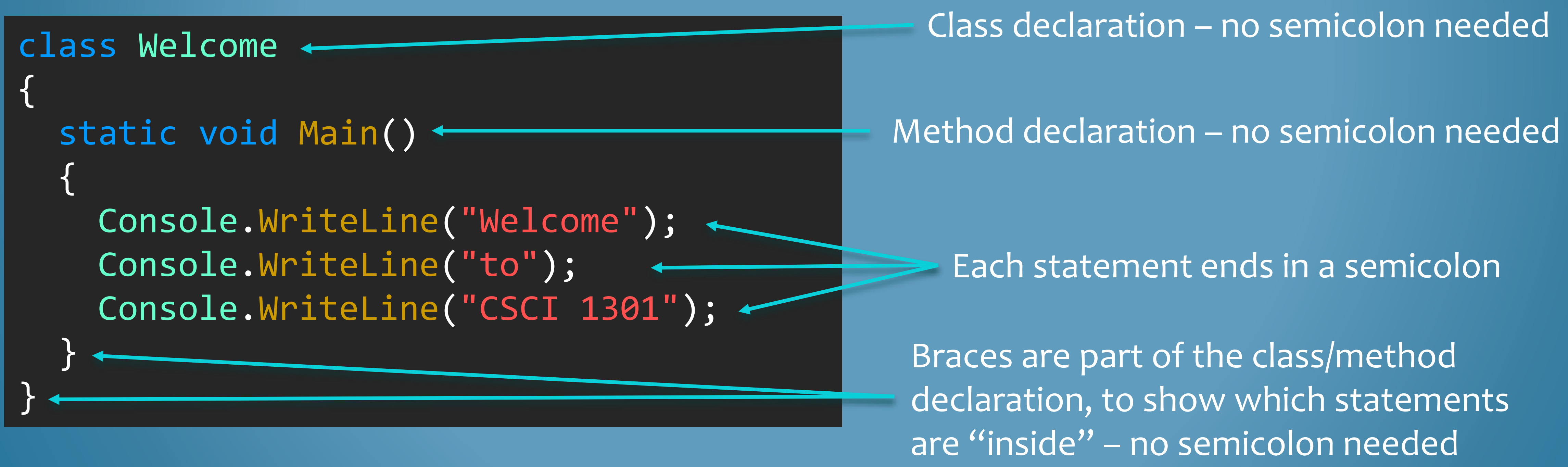
```
class Welcome
{
  static void Main()
  {
    Console.WriteLine("Hello");
    Console.WriteLine("World!");
  }
}
```

```
Hello
World!
```

AUGUSTA UNIVERSITY

Statements in a Method

- C# rule: Each **statement** must end in a semicolon
  - Statement ≠ line of code in your .cs file
  - Class and method declarations are not statements



```
class Welcome
{
  static void Main()
  {
    Console.WriteLine("Welcome");
    Console.WriteLine("to");
    Console.WriteLine("CSCI 1301");
  }
}
```

Class declaration – no semicolon needed

Method declaration – no semicolon needed

Each statement ends in a semicolon

Braces are part of the class/method declaration, to show which statements are “inside” – no semicolon needed

Write vs. WriteLine

- `Console.WriteLine`: Print text and then start a new line
- `Console.Write`: Just print the text, no newline

```
class Welcome
{
  static void Main()
  {
    Console.Write("Hello");
    Console.Write("World!");
  }
}
```

```
HelloWorld!
```

How could we fix this to print "Hello World!" (with a space)?

AUGUSTA UNIVERSITY

Write and WriteLine

- `Console.WriteLine` puts "cursor" at start of next line
- A subsequent `Console.Write` will start there

```
class Welcome
{
  static void Main()
  {
    Console.Write("Hello ");
    Console.WriteLine("World!");
    Console.Write("Welcome to ");
    Console.WriteLine("CSCI 1301!");
  }
}
```

```
Hello World!
Welcome to CSCI 1301!
```

AUGUSTA UNIVERSITY

Outline

- C# Introduction
- Example C# program
- Rules vs. Conventions of C#
- Reserved words and identifiers
- Console.Write and Console.WriteLine
- **Escape sequences**

Making the Newline "Visible"

- These programs print the same output:

```
class Welcome
{
  static void Main()
  {
    Console.WriteLine("Hello");
    Console.WriteLine("World!");
  }
}
```

```
class Welcome
{
  static void Main()
  {
    Console.Write("Hello\nWorld!\n");
  }
}
```

"\n": The **escape sequence** for "newline"

Means "insert a newline character here"

Special, Hard-to-Type Characters

- Escape sequences: use "normal" letters to represent "special" characters
- \n = newline character, i.e. result of pressing "enter"
- \t = tab character, a single extra-wide space

```
class Welcome
{
  static void Main()
  {
    Console.Write("Hello\tWorld!");
  }
}
```

```
Hello   World!
```

Quotes Inside Strings

```
class Welcome
{
  static void Main()
  {
    Console.WriteLine("This is "in quotes"");
  }
}
```

String ends

Invalid code!

Compile error!

- \" = double-quote character, without ending the string in C#

```
class Welcome
{
  static void Main()
  {
    Console.WriteLine("This is \"in quotes\"");
  }
}
```

```
This is "in quotes"
```

What If You Need a Backslash?

- All escape sequences start with \
- If C# sees a \ in your string, it assumes the next letter is for an escape sequence

```
Console.WriteLine("Go to C:\Users\Edward");
```

Compile error!

Invalid escape sequence: \U

- Solution: The escape sequence \\ = a single \ character

```
Console.WriteLine("Go to C:\\Users\\Edward");
```

```
Go to C:\Users\Edward
```

AUGUSTA UNIVERSITY