
TRY C#

Level 1

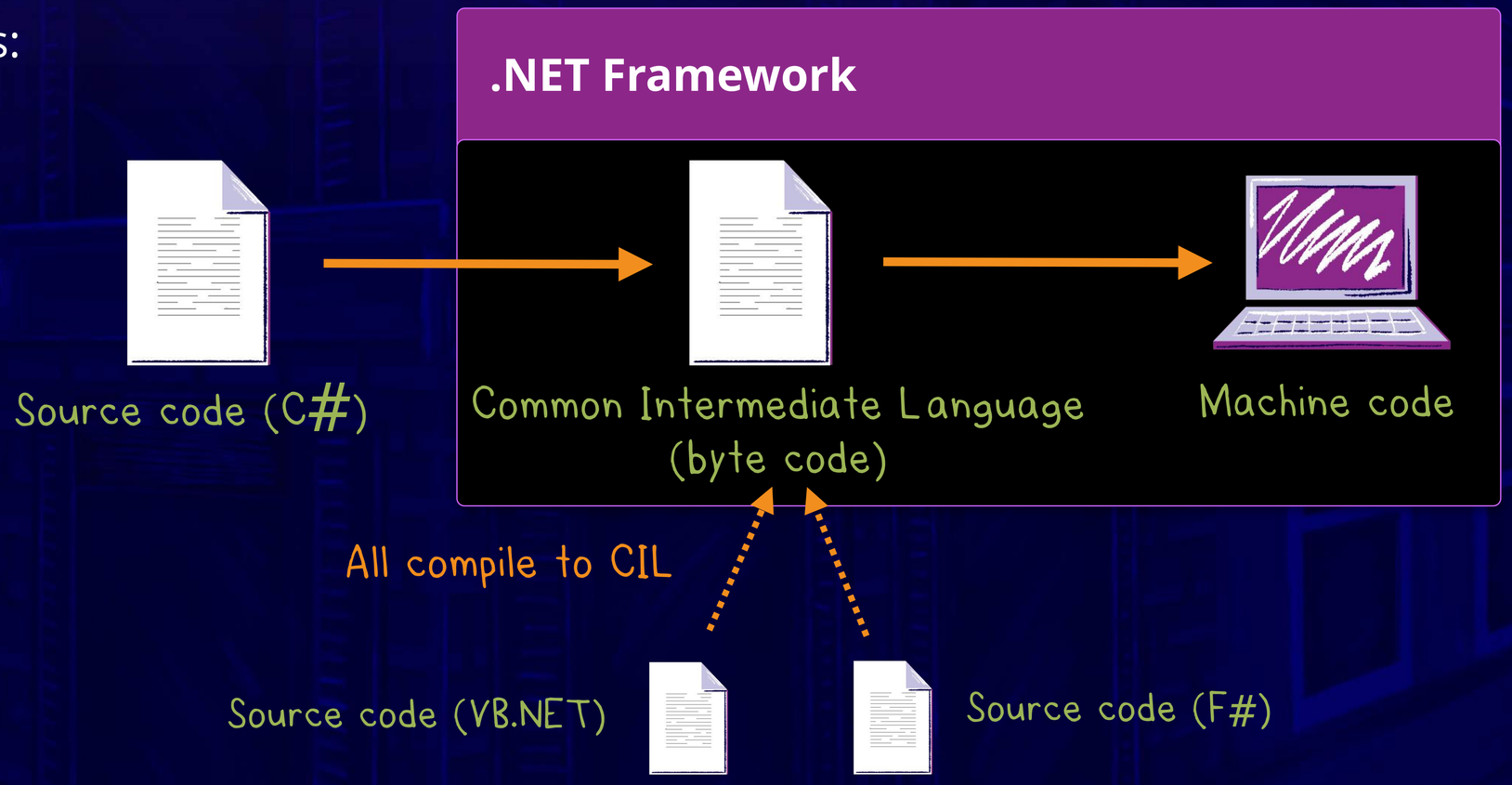
# Introduction

# What Is C#?

**C# is a general purpose object-oriented programming language released in 2002 by Microsoft.**

Some notable characteristics:

- Compiled
- Strongly typed
- .NET language





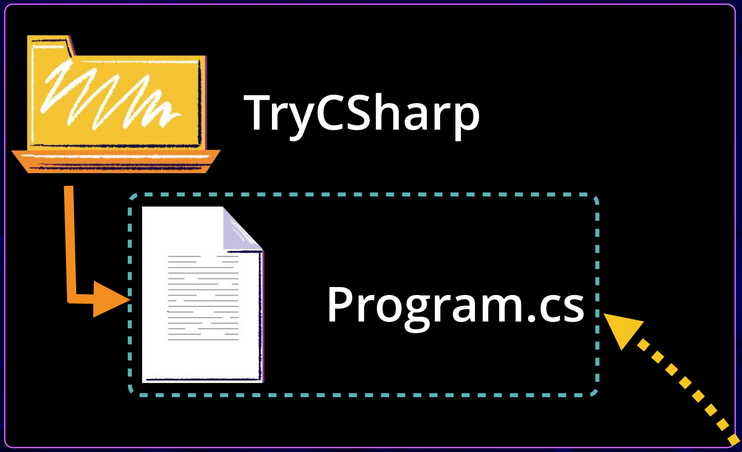
TRY C#

# Creating a New C# Application

**We can use the** dotnet new console **command to create a C# application with a** Program.cs **file.**

**Console**

```
>>> $ dotnet new console
The template "Console Application"
created successfully.
```




*All modern .NET applications start in the Program.cs file, so we'll start there*

For more information on installation, visit go.codeschool.com/install-dot-net

TRY C#

# The Program.cs File

This file is the entry point of our application. It's generated with the following code:

**Program.cs**

```
using System;

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

Classes allow us to separate our code into "objects"

Methods contain the executable code of our object

# Start of an Application

When our application is run, execution starts from the Main() method.

**Program.cs**

```
using System;

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

This is the first line of code executed

# Restoring Dependencies

Before we can run our application we need to use dotnet restore to restore our dependencies

**Program.cs**

```
using System;

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

dotnet restore needs to be run before you run the application the first time or anytime you change a dependency

**Console**

```
>>> $ dotnet restore

Restoring packages for TryCSharp…
```

# Running the Application for the First Time

When we run the application, it prints “Hello World!” to the console.

**Program.cs**

```
using System;

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

Strings are a collection of characters wrapped in double quotes

**Console**

```
>>> $ dotnet run

Hello World!
```

# Demo Application

Let's make our existing application read input from the user and use that as part of our output.

We'll need two things:

- Accept user input
- Concatenate strings

**Console**

```
>>> $ dotnet run
Type a message
>>> $ Hello World
You said Hello World
```

TRY C#

# Reading User Input

**The** Console.ReadLine **method reads user input from the console line and returns it as a** string**.**

**Program.cs**

```
using System;

class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Type a message");
        Console.WriteLine(Console.ReadLine());
    }
}
```

Reads user input as string

# String Concatenation

We can use the + character to concatenate multiple strings.

**Program.cs**

```
using System;

class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Type a message");
        Console.WriteLine("You said " + Console.ReadLine());
    }
}
```

The + will combine our strings

# Running the Demo Application

We can use the dotnet run command to run our program.

**Program.cs**

```
using System;

class Program
{
  ...
}
```

**Console**

```
>>> $ dotnet run
Type a message
>>> $ Hello World
You said Hello World
```

User types this

Program prints these

# Behind the Scenes of String Concatenation

This is what happens behind the scenes when using user input from Console.ReadLine.

Step 1.

```
Console.WriteLine("You said " + Console.ReadLine());
```

User input is read

Step 2.

```
Console.WriteLine("You said " + "Hello World");
```

Strings are combined

Step 3.

```
Console.WriteLine("You said Hello World");
```

# Quick Recap on Getting Started

We can use the dotnet commands to create, compile, and run applications.

```
>>> $ dotnet new console
The template "Console Application"
created successfully.
>>> $ dotnet run
Type a message
>>> $ Hello World
You said Hello World
```

Creates a new application

Runs the application