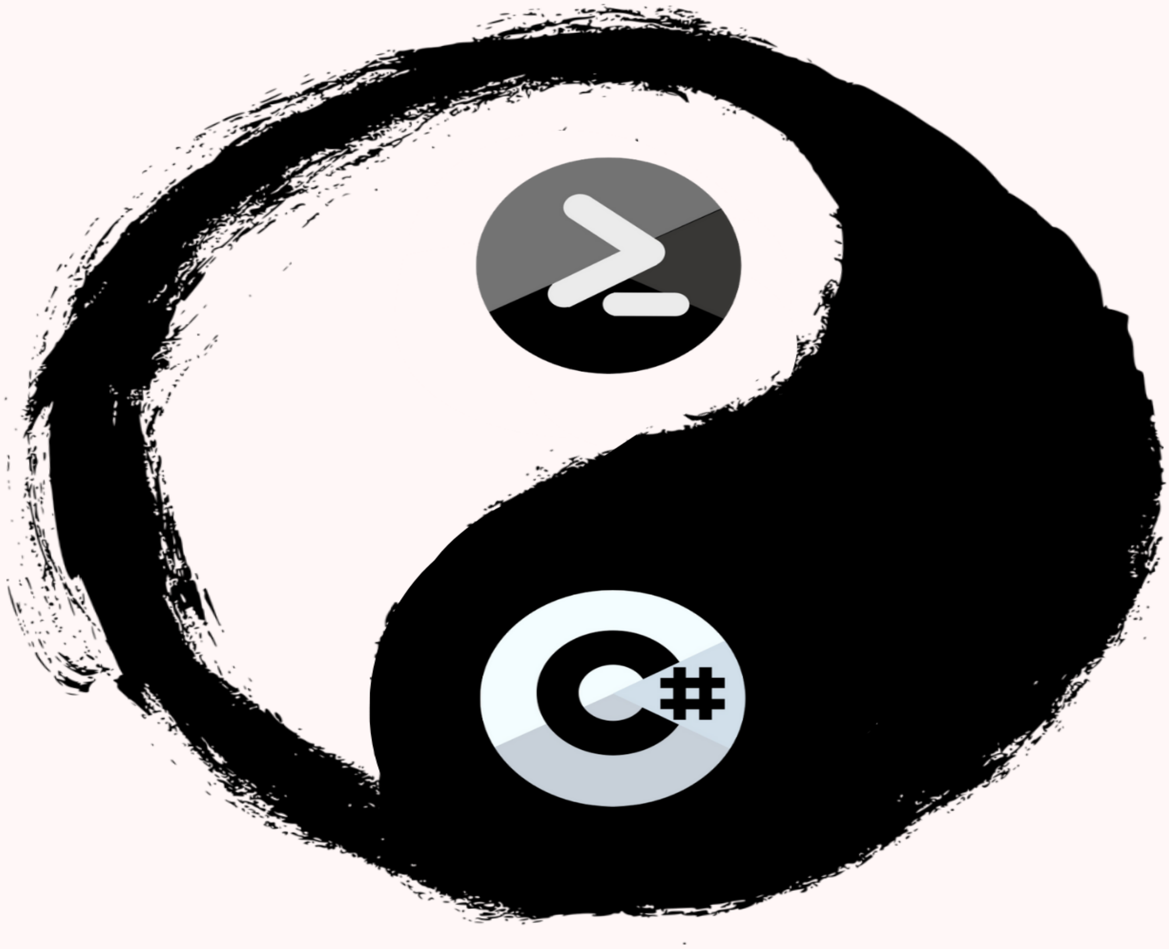
DEEPAK DHAMI & PRATEEK SINGH



# POWERSHELL TO C# AND BACK

LEARN C# AND EXCEL AT POWERSHELL

# PowerShell to C# and back

A journey to C# and becoming better at PowerShell

Prateek Singh and Deepak Singh Dhami

This book is for sale at http://leanpub.com/powershell-to-csharp

This version was published on 2022-02-10

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

© 2020 - 2022 Prateek Singh and Deepak Singh Dhami

# Tweet This Book!

Please help Prateek Singh and Deepak Singh Dhami by spreading the word about this book on Twitter!

The suggested hashtag for this book is #pwshtocsharp.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

#pwshtocsharp

## Also By These Authors

### Books by Prateek Singh

PowerShell Guide to Python

Plain Perception

Learn C# in 30 minutes

### Books by Deepak Singh Dhami

Learn C# in 30 minutes

# Contents

# About the Book

## Why did we write this book

We started writing this book to document our journey to learn C#, to be honest we were scratching our own itch. There is no book or tutorial available today that can bridge the gap between a scripting language like PowerShell and modern programming language like C#, which is a natural language of choice, so we decided to write our own!

PowerShell cmdlets are mostly written in C# and very often you will see .Net classes utilized in PowerShell scripts, and it seems like both these worlds are very much interconnected but still there is a lot of friction, especially for IT professionals who lack understanding of object oriented programming and find it difficult to understand software design patterns.

Purpose of this book is to provide bite sized easily consumable chunks of knowledge and concepts to the reader, that will make it simple for them to learn and understand C# and improve their skill sets. More than that, leaving readers will tips, tricks and scenarios, that will develop a synergy with C# and PowerShell and the knowledge attained can be refed further to extend their capabilities by writing better and advanced scripts and Automation.

> "Your problem is to bridge the gap which exists between where you are now and the goal you intend to reach."
>
> ~ Earl Nightingale

## Who is the Target Audience

C# can be used to create almost anything from Windows desktop applications and games to develop powerful web applications and has become increasingly popular for mobile development too. Cross-platform tools such as Xamarin allow apps written in C# to be used on almost any mobile device. On top of that, since the announcement of .NET Core which is an open-source, general-purpose development platform now you can create .NET Core applications using C# for Windows, macOS, and Linux this just opens up a lot of opportunities and horizons for C#.

### System Administrators and IT Professionals

System Administrators who wants to upgrade their PowerShell scripting skills into C# Development or Programming roles, this book is written exactly keeping you in mind. The purpose of this book is to bridge the gap between PowerShell and C# and leave the readers with enough confidence, exposure and hands on experience, so that they can develop in C# comfortably. Microsoft's cloud platform 'Azure' is growing rapidly, and having C# and PowerShell in your skill arsenal, that compliment each other and align with Azure automation and development needs, will definitely give you competitive advantage as an IT professional.

### C# Developer

If you are C# developer and want to learn to write powerful automation and scripts using PowerShell, then you must read this book, because this book will help you associate concepts of modern programming language like C# with PowerShell and make it easy for you to learn PowerShell. Now a days most of the CI/CD platforms like Azure DevOps leverage some sort of scripting language to run tasks and background jobs in their pipelines, so learning PowerShell will start to impact your day job in a good way.

## Book updates and email notifications

This book has been published as we are writing it, a chapter at a time on Leanpub.com , which is a lean publishing platform and we will keep on updating and improving the book, so that you have the best version of the book available. This book is work in progress and it is recommend that you allow Leanpub to send you update emails when a new version of book manuscript is published.

By default this setting is not enabled and you have to turn it on by following below steps, otherwise you will miss any new chapters or updates we push to the book.

1. Log into your Leanpub.com account
2. From the top-right corner in the 'Account' drop-down menu, click on 'Library'
3. Click on this book in your library.
4. Now on the right-hand side you will get 'Email Settings'
5. Under 'Email Settings' select the checkbox next to 'New version available'

## How to provide Feedback

We would appreciate you sending us an e-mail to prateek@ridicurious.com with any feedback about how you are using the content, and how the book could be improved, but this is just a request, so that we can improve the book and track how it is being used. You can also use the "Email the Author(s)" option on the book webpage to reach out both of us!

# Prerequisites

Some prior experience and knowledge of PowerShell is recommended, but even if you are not experienced in PowerShell this book gives an opportunity to learn PowerShell basics while you commence your c# journey. Apart from that we recommend to install Visual Studio Code, .Net Core SDKs and Visual Studio (this is optional but it provide an Interactive C# command-line which can come handy).

## Setup and Tools

All code snippets and example in this book, is expected to run without any issues on all the versions of PowerShell v5.1 and onwards. Follow along the below steps to download and install the correct version of PowerShell on your machine before proceeding with the chapters in this book.

### Windows PowerShell

Windows users can check current version of PowerShell installed on their system by running the following one-liner from a PowerShell console. If the returned version is above `5.1.x` or above then we are good to go.

```
$PSVersionTable.PSVersion.ToString()
```

If the version is lower than `5.1`, then please download and install 'Windows Management Framework 5.1' from the Microsoft Download Center, which includes Windows PowerShell 5.1.

*NOTE: You can also install PowerShell core on Windows using the steps mentioned in this Microsoft's official documentation.*

## PowerShell Core

Linux and MacOS user can use the following steps to install latest and greatest PowerShell v7 on their systems.

- PowerShell v7 installation steps for Linux
- PowerShell v7 installation steps for MacOS

## Visual Studio Code, C# and Extension .Net Core SDK

- Download and install Visual Studio Code
- Install the C# extension for Visual Studio Code
- Download and install the .NET Core SDK

## Visual Studio (Optional)

We recommend you to download and install Visual Studio which is best in class IDE from Microsoft, but this is totally optional. Examples and code snippets in the following chapters of this book can be executed using Visual Studio 2019 v16.5 or later with the '.NET Core cross-platform development workload' installed. Unlike Visual Studio Code, the installation of Visual Studio has a little bigger resource foot print.

# How to Use this Book

The book is written keeping in mind, that most reading will be in bite size easily consumable chunks of one hour or less so each section of a chapter can be completed in one hour at the maximum and some will be much shorter. There are many different ways to use this book, and some of which are suggested in the following points:

- A straight read through of the book in a few weeks. All the chapters are structured to help you build concepts with good cadence as you progress through the book.
- Cherry pick specific chapters or section\sub-section that is useful for you, or seems interesting to you. For this reason, we provide 'Purpose of the chapter' section as a first thing in each chapter chapter of the book, so that before reading the chapter you can come to conclusion if this is something you should read now ord come back later.
- We encourage you to do the activities that follow most chapters or sections\subsections that will reinforce your takeaways from the chapter and will help build some confidence writing the code. You can also, refer the 'Reading recommendations' section at the end of the chapters to further enhance the knowledge on the topic which this book is unable to cover.

## Available Book Formats

The book can be download in any of the following formats and it is recommended to read it on a personal computer in PDF format for clarity in images and code samples. We might publish this book on Amazon through Kindle Direct Publishing (KDP) in digital and print in future but for now it is only digitally available on Leanpub.com.

| Format | Description |
|---|---|
| PDF | For laptops and personal computers |
| EPUB | For phones and tablets |
| MOBI | For Amazon Kindle and E-Readers |
| WEB | On the web browser |

## Structure of a Chapter

Each chapter begins with a set of learning goals for the chapter and the topics covered as a list, which gives a fair idea to the reader what to expect in the chapter. It enables the readers to with a choice to skip the chapter if the topic is familiar and can be revisited later stage. After this a small introduction of the chapter to make reader comfortable with topic, followed by the main content of the chapter, headings and sub-headings, figures, code samples/examples.

Towards the end of the book all chapters have a 'Key takeaway' section which summarizes key pointers of the chapter. Key takeaways can be used to reinforce the learning and even come handy during revisions without going deep dive into the whole chapter again. Finally chapters will end with some ExercisesAssignments that will further allow you to use the learnings from the chapter and implement that in code.

All chapters end with some reading recommendations, official documentations for the reader on the related topic, which readers can follow to gain knowledge on the topic which is out of the scope of the book.

Following is the structural pattern you will observe in all the chapters:

- Purpose of the chapter
- Introduction to the chapter
- Main chapter content
- Key Takeaways from the Chapter
- Exercises and Assignments
- Reading Recommendations

## Downloading Code samples and Examples

All the PowerShell and C# code samples, snippets and examples you come across in this book would be available to be download, from the 'Download Extras' section of the book. Please follow the below steps to get that:

1. Log into your Leanpub.com account
2. From the top-right corner in the 'Account' drop-down menu, click on 'Library'

3. Click on this book in your library.
4. Now on under 'Read this book' section, click on the 'Download Extras'.
5. Which will download all the all the code samples, Jupyter Notebooks in a compressed (.zip) file.

## Running code samples and examples

If you are reading this book, then it is highly likely you are coming from System Administration background and have intermediate to advanced PowerShell scripting skill set. Now keeping that in mind, it is highly recommended to all readers to run C# interactively, so that the users don't miss the interactive nature of PowerShell when they start learning C#. Idea here is reduce the friction while learning something new and build some confidence for you without getting overwhelmed with vastness, diversity and concepts C# as a programming language brings.

This was the biggest challenge we personally went through when we started learning C#, but following tools allow use to use C# interactively and give us a leverage to experiment expressions and code snippets without creating a full Project in IDEs like Visual Studio or Visual Studio Code and defining a namespace and then calling the `main()` method with the expressions or code snippets we wanted to test. Purpose is to make it faster to test your code snippets and reducing the feedback time.

Following are three ways you can run the code snippets in this book.

1. C# interactive Window in Visual Studio
2. `dotnet-script` extension for .Net Core CLI
3. PowerShell and C# Jupyter Notebooks

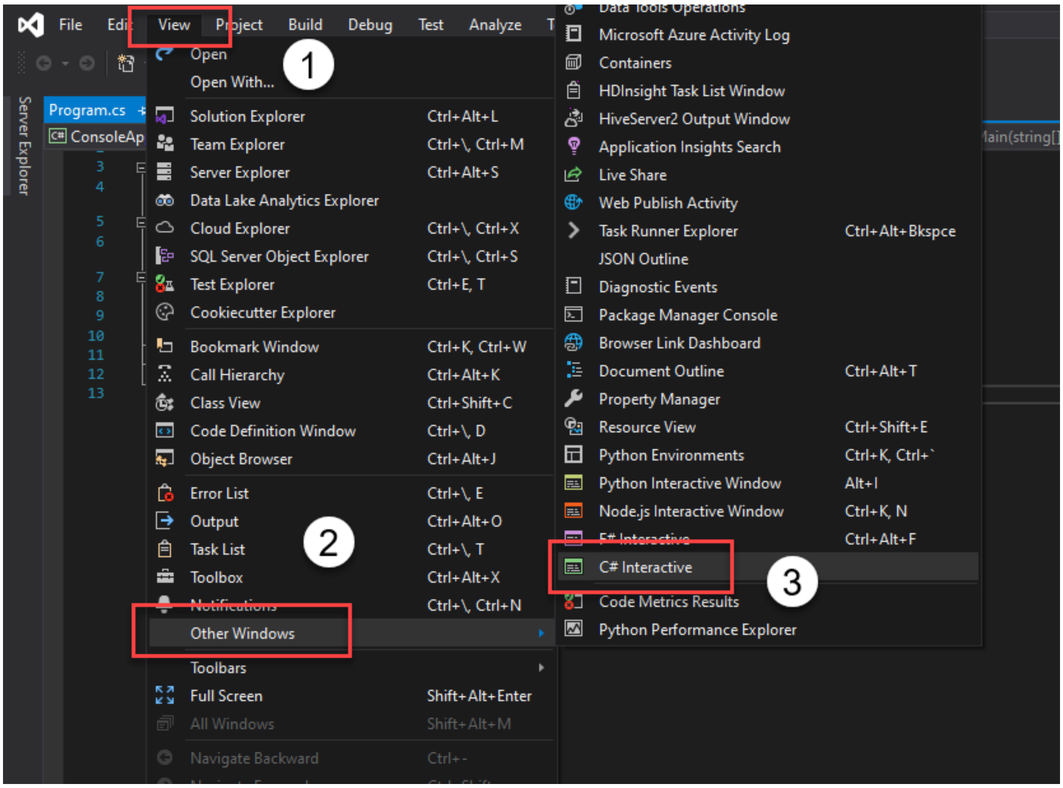
These are explained in detail in following sub-sections.

### C# Interactive Window in Visual Studio

C# interactive Window is a simple, REPL (`read-eval-print-loop`) interactive programming environment that takes one user input at a time in form of commands and expressions to let you play with APIs, learn new language features and experiment by enabling us to evaluate them directly with immediate feedback as results to the user. Alongside the productivity enhancement, it also provides advanced editor features like:

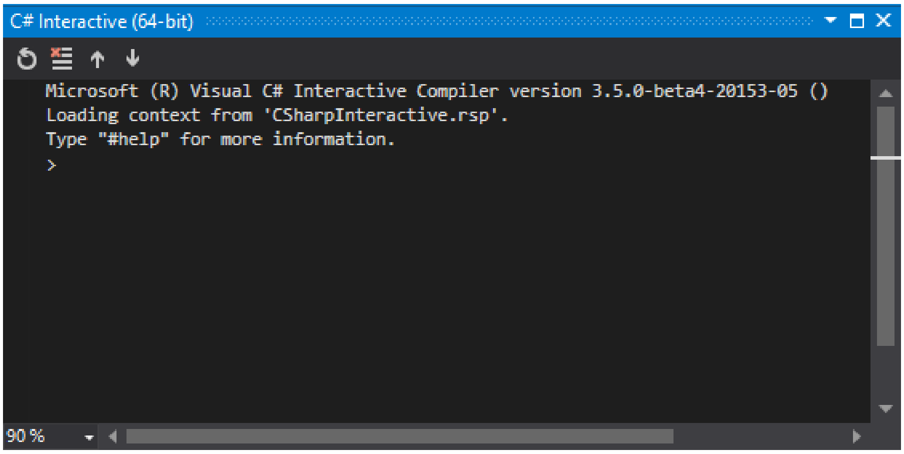
- Intellisense and code completion
- Syntax coloring
- Multiple line input
- History
- Find and replace

You must have latest version of Visual Studio installed to access the Interactive Window. Once you have one of the latest versions installed, then you can launch the C# Interactive Window, from the Menu bar, by navigating to View > Other Windows > C# Interactive.


File
Edit
View
Project
Build
Debug
Test
Analyze
1
Open
Open With...
Solution Explorer Ctrl+Alt+L
Team Explorer Ctrl+\, Ctrl+M
Server Explorer Ctrl+Alt+S
Data Lake Analytics Explorer
Cloud Explorer Ctrl+\, Ctrl+X
SQL Server Object Explorer Ctrl+\, Ctrl+S
Test Explorer Ctrl+E, T
Cookiecutter Explorer
Bookmark Window Ctrl+K, Ctrl+W
Call Hierarchy Ctrl+Alt+K
Class View Ctrl+Shift+C
Code Definition Window Ctrl+\, D
Object Browser Ctrl+Alt+J
Error List Ctrl+\, E
Output Ctrl+Alt+O
Task List Ctrl+\, T
Toolbox Ctrl+Alt+X
Notifications Ctrl+\, Ctrl+N
2
Other Windows
Toolbars
Full Screen Shift+Alt+Enter
All Windows Shift+Alt+M
Navigate Backward Ctrl+-
Data Tools Operations
Microsoft Azure Activity Log
Containers
HDInsight Task List Window
HiveServer2 Output Window
Application Insights Search
Live Share
Web Publish Activity
Task Runner Explorer Ctrl+Alt+Bkspce
JSON Outline
Diagnostic Events
Package Manager Console
Browser Link Dashboard
Document Outline Ctrl+Alt+T
Property Manager
Resource View Ctrl+Shift+E
Python Environments Ctrl+K, Ctrl+`
Python Interactive Window Alt+I
Node.js Interactive Window Ctrl+K, N
F# Interactive Ctrl+Alt+F
C# Interactive
3
Code Metrics Results
Python Performance Explorer
Server Explorer
Program.cs
ConsoleApp

*Launching C# Interactive Windows from Visual Studio*

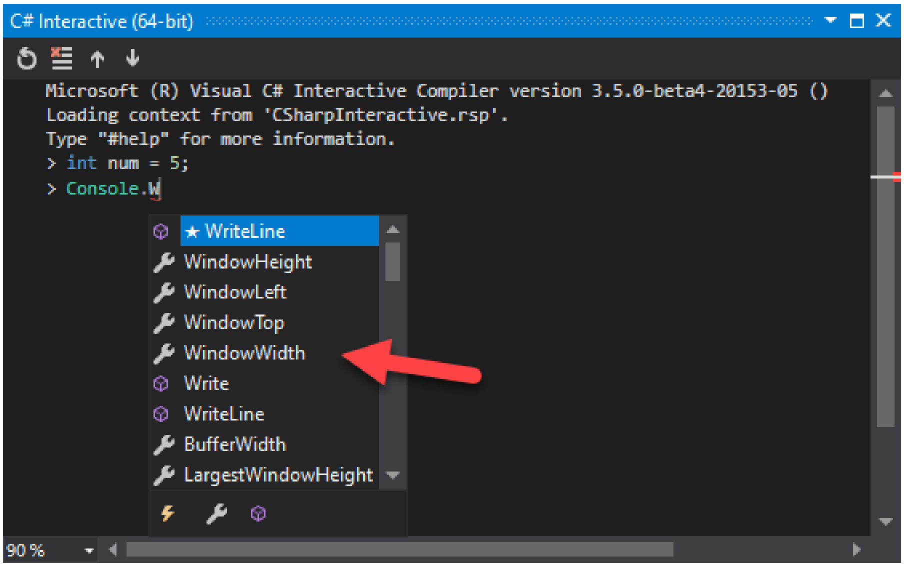
This will bring up the Interactive Window as demonstrated in the following Figure:



```
Microsoft (R) Visual C# Interactive Compiler version 3.5.0-beta4-20153-05 ()
Loading context from 'CSharpInteractive.rsp'.
Type "#help" for more information.
>
```

*C# Interactive Window*

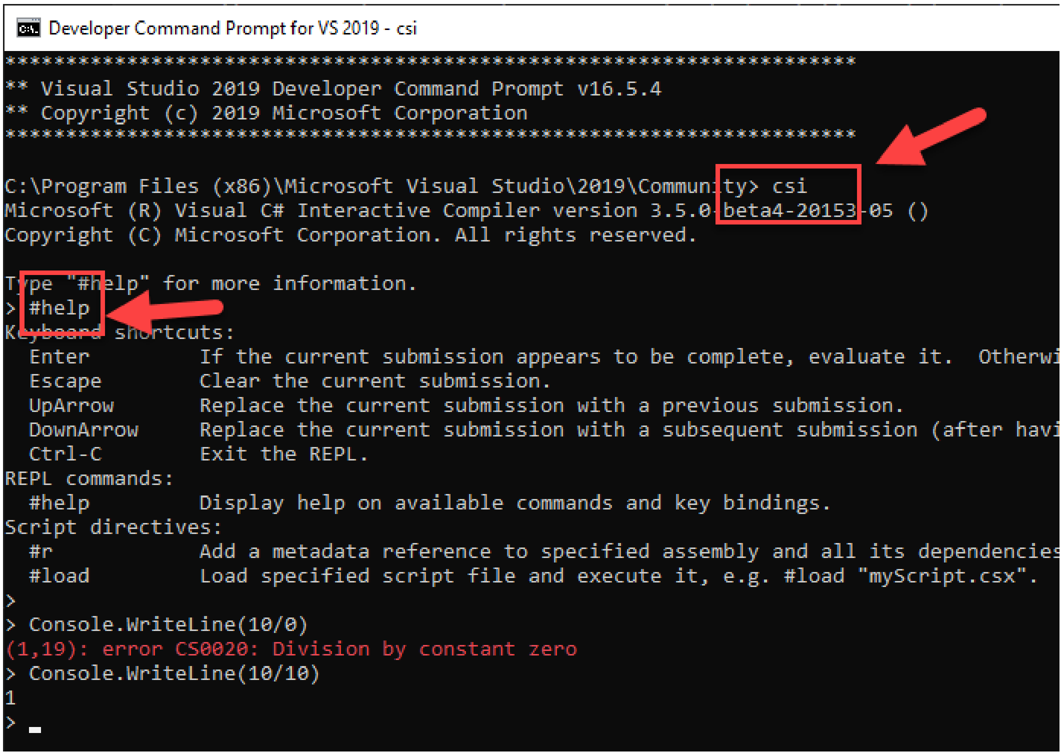
You can very simply execute code samples one line at a time, with rich text editor like features: syntax coloring and code completion from the comfort of interactive console as demonstrated in the following image.



```
Microsoft (R) Visual C# Interactive Compiler version 3.5.0-beta4-20153-05 ()
Loading context from 'CSharpInteractive.rsp'.
Type "#help" for more information.
> int num = 5;
> Console.W
```

*Running Commands in C# Interactive Window*

This command-line script execution engine (CSI) is also available outside Visual Studio and can be access from Developer Command Prompt or PowerShell for Visual Studio 2019, just by running the command: `csi` , here `csi` stands for C Sharp Interactive.


Developer Command Prompt for VS 2019 - csi

```
**********************************************************************
** Visual Studio 2019 Developer Command Prompt v16.5.4
** Copyright (c) 2019 Microsoft Corporation
**********************************************************************

C:\Program Files (x86)\Microsoft Visual Studio\2019\Community> csi
Microsoft (R) Visual C# Interactive Compiler version 3.5.0-beta4-20153-05 ()
Copyright (C) Microsoft Corporation. All rights reserved.

Type "#help" for more information.
> #help
Keyboard shortcuts:
  Enter         If the current submission appears to be complete, evaluate it.  Otherwi
  Escape        Clear the current submission.
  UpArrow       Replace the current submission with a previous submission.
  DownArrow     Replace the current submission with a subsequent submission (after havi
  Ctrl-C        Exit the REPL.
REPL commands:
  #help         Display help on available commands and key bindings.
Script directives:
  #r            Add a metadata reference to specified assembly and all its dependencies
  #load         Load specified script file and execute it, e.g. #load "myScript.csx".
>
> Console.WriteLine(10/0)
(1,19): error CS0020: Division by constant zero
> Console.WriteLine(10/10)
1
> _
```

*C# Interactive from Developer Command Prompt*

## `dotnet-script` extension for .Net Core Command Line

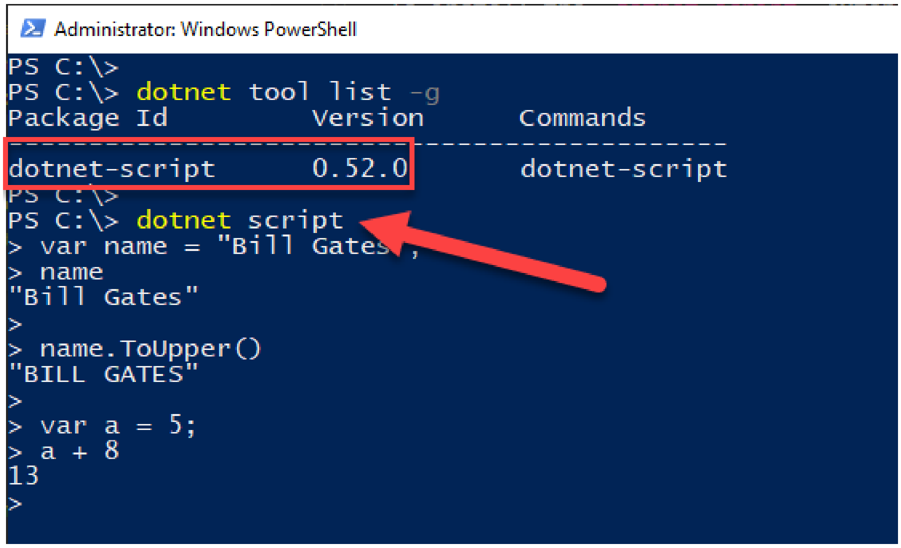
A list of tool extensions for .NET Core Command Line, also known as '.NET Core global tools' can be downloaded using the dotnet CLI, but we are specifically looking for a tool called `dotnet-scripts`. This extension allows you to run C# scripts (`.csx` files) from the .NET CLI, define NuGet packages inline, edit and debug them in VS Code. More than that you get a interactive C# console to run your snippets directly from Visual Studio Code Terminal or any console like PowerShell or even Command prompt (CMD).

To install the `dotnet script` extension run the following command, simply using nothing but the .NET CLI:

```
dotnet tool install -g dotnet-script
```

Once the installation is complete you can also list the and verify all the tool extension using the .NET CLI again as demonstrated in the following example and enter in an interactive REPL console by running `dotnet script`:

```
dotnet tool list -g
dotnet script
```

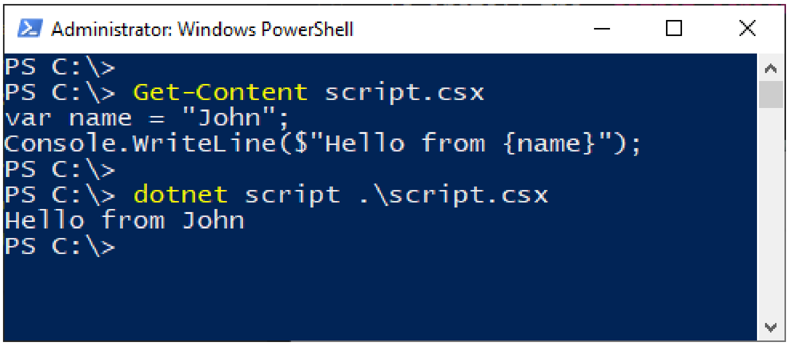


```
Administrator: Windows PowerShell
PS C:\>
PS C:\> dotnet tool list -g
Package Id         Version      Commands
---------------------------------------------
dotnet-script      0.52.0       dotnet-script
PS C:\>
PS C:\> dotnet script
> var name = "Bill Gates";
> name
"Bill Gates"
>
> name.ToUpper()
"BILL GATES"
>
> var a = 5;
> a + 8
13
>
```

*Installing global tools extension with .NET CLI*

Above image illustrates, that we can define variables interactively and access methods on them one line at a time. Moreover, you can combine such commands into a file and save it as `.csx` extension, which is a CSharp script file and execute the script using the syntax:

```
dotnet script <path to .csx script file>
```



```
PS C:\>
PS C:\> Get-Content script.csx
var name = "John";
Console.WriteLine($"Hello from {name}");
PS C:\>
PS C:\> dotnet script .\script.csx
Hello from John
PS C:\>
```

*Running C# script files using .NET CLI*

## PowerShell and C# Jupyter Notebooks

This book comes with code samples in a Jupyter Notebook that you can download and run locally. It will make very easy for the readers to test their code and there would be no need to copy-paste code and execute it, all you have to do is to follow the below mentioned steps to make the Jupyter notebooks working.

1. Install Python3 & Jupyter.
   Ensure to select the option to add Python to the PATH while installing on Windows.

   with .NET interactive kernels.

   > If the above command doesn't resolve Python binary in a PowerShell console. Then discover the Python executable on your system and configure the environment variables accordingly.

   * On Linux/MacOS

   ```
   whereis python3
   ```

   Install Jupyter by running below commands (should work for Windows/Linux/MacOS):

   ```
   pip3 install --upgrade pip
   pip3 install jupyter
   ```

2. Download .NET Core 3.1 SDK and install it.
3. Install Microsoft .NET Interactive global tool using the `dotnet` CLI as shown in the following example:

```
dotnet tool install -g --verbosity normal --add-source "https://pkgs.dev.azure\
.com/dnceng/public/_packaging/dotnet-tools/nuget/v3/index.json" Microsoft.dotne\
t-interactive
```

4. Install the .NET Kernel.

```
dotnet interactive jupyter install
```

5. Verify the Kernels are installed.

```
jupyter kernelspec list
```

6. Run Jupyter Lab, it will provide a weblink to access the notebooks.

```
cd ~/notebooks # Set the current directory to the path of downloaded notebooks
jupyter lab # run jupyter lab
```

7. Use the NoteBook in the web portal.

# Chapter 1: C# Primer

## Purpose of the Chapter

When you have read this chapter you should be able to:

1. Create your first program in C# and organize it into Namespaces and Classes.
2. Create a new .Net project, build and run 'Hello World' in C# from `dotnet` CLI.
3. Utilize .Net Namespaces in your C# programs.
4. Define variables of various Data types and initialize them.
5. Use arithmetic, logical, relational, assignment and ternary operators in your program.
6. Define conditional statements and loops.
7. Create simple C# classes and instantiate them into objects.
8. Handle simple errors and exceptions in your code.

## Introduction

In this chapter we are going to learn the key characteristics of C# language and brief history of its evolution over the years. Then we will look into the very basics of C# language, starting from creating your 'Hello World' program and understanding a general structure and organization of C# program. Extending to learning the underlying framework and concepts that compile and executes your C# program. Finally we will learn the elements of C# programming language such as Data Types, Variables, Class, Objects, Loops, Conditional statement and Exception Handling.

Let's begin.

## C# Language

C# is a powerful, flexible and very popular modern programing language, which is simple and easy to learn and at the same time elegant as a programming language of few words.

C# is used for a wide range of reasons but the popularity lies in its use in building cross platform applications, websites, games and cloud services.

## Key Characteristics

C# have some modern features and wide variety of applications and use cases, but these are possible because of the following key characteristics it offers:

- **Modern** - C# was designed keeping in mind that it supports all modern day functionalities and provides features to cater modern software development needs. Some of the modern programming language features are automatic garbage collection, lambda expressions, advanced debugging, exception handling and most importantly security.
- **Easy to learn** - It is a high-level language and fairly simple to digest, because it is somewhat similar to English language. More than that, C# is a C-styled language and if you have experience with C, C++ or Java, then you can very easily get familiar with the syntax and learn the language fast.
- **Open source** - .NET Core and the C# compilers are both open source under .Net Foundation and available on GitHub. The community can participate in the C# language development along with Microsoft who owns and governs the changes.
- **Cross platform** - Developers can build .NET applications that can be deployed on Windows, Linux, and MacOS and more than that deployed in containers or cloud.
- **Object Oriented** - C# is object oriented programming language and concepts of object-oriented programming language like encapsulation, inheritance, and polymorphism are all supported, which makes development and maintenance of code easier as the project grows compared to a Procedure-oriented programming language.
- **Type safety** - C# enforces type safety by limiting ways of interaction of objects by the object type. Only operations are permitted by type definition and are applied to the object, that means type casting objects to incompatible data type is restricted.
- **Robust and Versatile** - You can use C# to create cross platform client applications, Web services, distributed components, cloud applications, database applications, Internet of Things (IoT) and now artificial intelligence and Machine learning.
- **Modular** - C# supports modular software development, that means applications can be written in chunks or pieces of code as in functions, classes etc that are reusable, easy to modify and extensible.
- **Secure** - .Net provides many useful classes and services to enable developers to secure their code with prebuilt algorithms like AES, that have stood the test of time. More than data you also get a Data protection API (DAPI) that provides classes for cryptography that can be leveraged by developers to encrypt data and secure their applications.

- **Evolving** - C# is the fastest evolving programming language since its announcement and initially it was only designed to develop Windows client applications, but today it can do pretty much anything. Every year new features are added to the language, and this is all possible because of Microsoft and the strong open-source community behind it.

## A Brief History of CSharp

C# was developed as part of the .NET framework initiative at Microsoft in 1999, by a team lead by Anders Hejlsberg, who had designed several popular and successful programming languages, like Turbo Pascal, Delphi etc. The C# was developed with a design goal to develop a programming language that is simple, general-purpose, and object-oriented. But with time C# evolved into a much mature and versatile programming language and the following table represents the features added in each version since it was announced.

| Version | Year | Features |
|---|---|---|
| 1.0 | 2002-03 | Object Oriented, Classes, Flexible, Typesafe, Managed, Garbage Collection, Cross-platform |
| 2.0 | 2005 | Generics, Anonymous Methods, Iterators, Partial types, Nullable value types |
| 3.0 | 2007 | Expression trees, LINQ, Lambda Expression, Extension Method, Anonymous types |
| 4.0 | 2010 | Dynamic Binding, Named and Optional Parameters |
| 5.0 | 2012 | Async Programming |
| 6.0 | 2015 | Exception filters, Auto property initializers, Null propagator, String interpolation and `nameof` operator, Await in catch/finally block |
| 7.0 | 2017 | Tuples, Local functions, Out variables, Pattern matching, Throw expressions |
| 7.1 | 2017 | Async main, Default literal expressions, Inferred tuple element names |
| 7.2 | 2017 | `private` and `protected` access modifier, Non-trailing named arguments, Leading underscores in numeric literals |
| 7.3 | 2018 | Accessing fixed fields without pinning, Reassigning ref local variables, Using initializers on stackalloc arrays, Using fixed statements with any type that supports a pattern, Using additional generic constraints |
| 8.0 | 2019 | `Readonly` member, ranges and indices, switch expressions, static local functions, Null-coalescing assignment |

## Compilers, Runtime and .NET Framework

C# is a programming language and .NET is a blanket term to cover both the .NET Framework, which is an application framework library and the Common Language Runtime (CLR) which is the runtime in which .NET assemblies are run.

Modern day software developers usually work on programming languages that are high level abstractions and the majority of us don't understand how our code actually runs down to the level of processor. I mean this is the purpose of modern programming languages to make it simple for developers to focus on problem solving, rather than understand details of computers and underlying architecture.

Let's take a step back and understand: how a C# program executes under the hoods? it my be good exercise for us to learn how C# and .Net work together to provide us the intended results.

1. First we write the source code in C# and let's suppose to save it in a file named: `FirstProgram.cs` and then we compile it with all the required resources and references.
2. A compiler which is a program that converts the source code into an intermediate language and saves that into a file `FirstProgram.exe` or `FirstProgram.dll` with `.dll` or `.exe` extension. This intermediate language is also known as Microsoft Intermediate Language (MSIL).
3. The computer processor still doesn't understand the intermediate language and can only work on native\machine codes. That is why we have another program called 'Common Language Runtime' (CLR) that uses a 'Just-In-Time' (JIT) compiler to transform this intermediate language into machine code in runtime.

   > A Just-In-Time compiler uses the .Net Framework's extensive library of more than 4000 classes to provide a wide variety of useful functionalities like Windows Forms, Cryptography, File handling etc.

4. The machine code native code that a computer can understand is nothing but a set of instructions for the computer to perform, which are generally very low-level, like memory allocation etc.

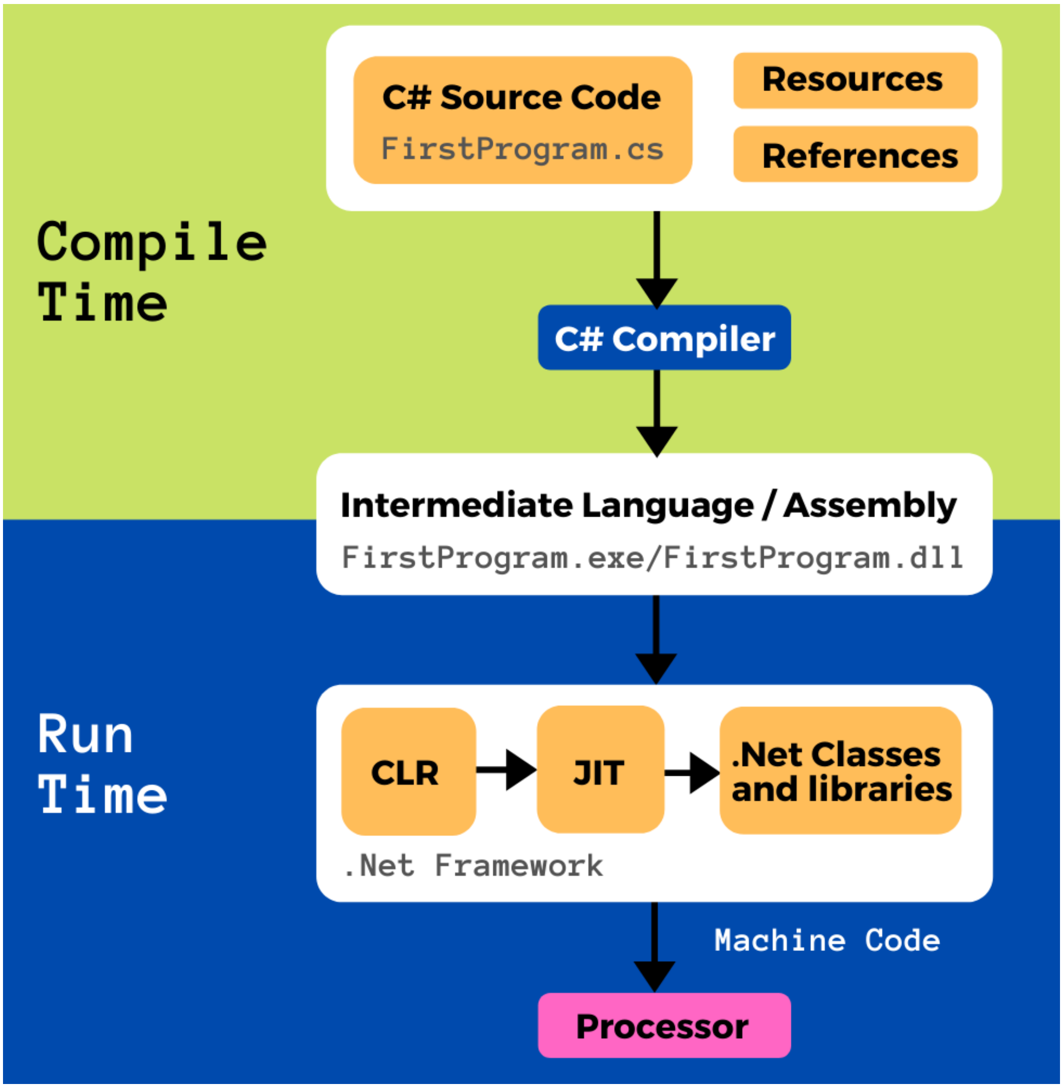


*Figure 1-1. C# Code compilation and execution*

On a high-level these steps can also be categorized into two parts:

1. Compile Time - Transformation of source code to intermediate language.
2. Run time - Conversion of intermediate language to machine code and executing machine code instructions.

## Program Structure and Organization

In this subsection we will learn, some key organizational concepts that enable developers to structure their C# code in:

- Program
- Namespace
- Class
- Members
- Assemblies

### Program, Assemblies and Hello World

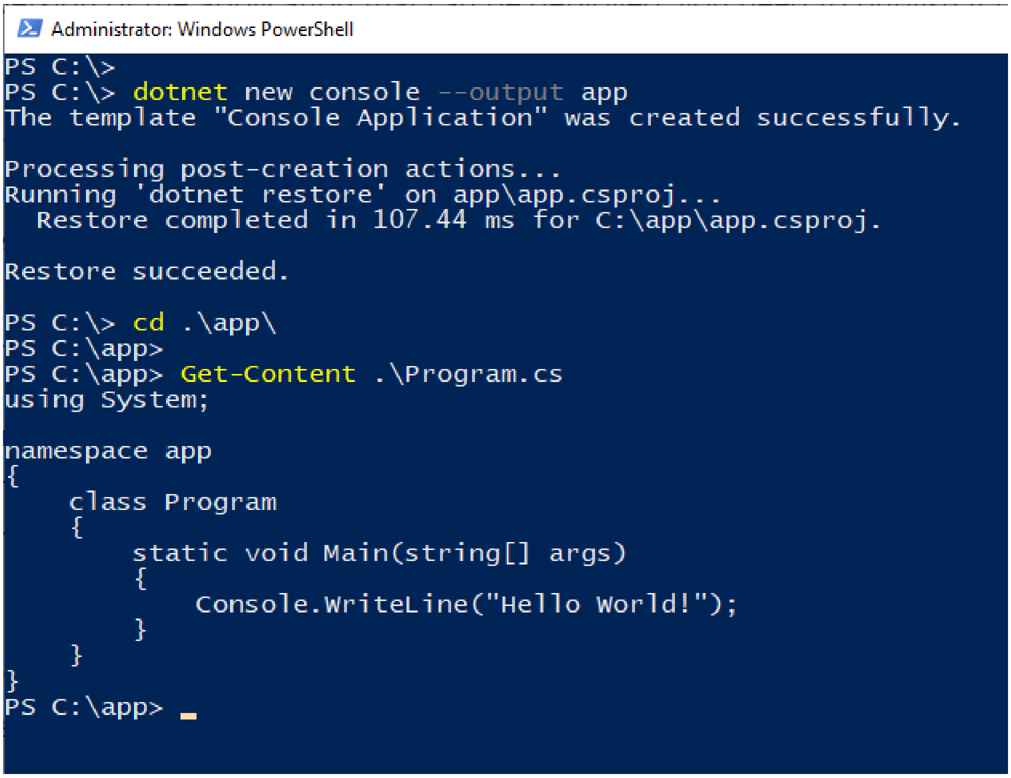
A C# program consists of one or more source code files, that contains Classes, methods, properties separated in namespaces and when the C# program is compiled, then these are packaged into assemblies with extension `.exe` for applications and `.dll` for libraries.

Since, it's like an old tradition to introduce new programming language to the readers using a 'Hello World!' program, so keeping that in mind here are the steps to create your first Hello World program in C#:

To create a basic console application using .Net Core simply run the following command from PowerShell or Windows Command prompt:

```
dotnet new console --output app
```

This will create a new console application project and scaffold a folder named: `app` with the bare minimum files required. One important file created in this folder is `program.cs` where `.cs` extension refers to the C# Hello Program or source code file.



Administrator: Windows PowerShell

```
PS C:\>
PS C:\> dotnet new console --output app
The template "Console Application" was created successfully.

Processing post-creation actions...
Running 'dotnet restore' on app\app.csproj...
  Restore completed in 107.44 ms for C:\app\app.csproj.

Restore succeeded.

PS C:\> cd .\app\
PS C:\app>
PS C:\app> Get-Content .\Program.cs
using System;

namespace app
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
PS C:\app> _
```

*Figure 1-2. Creating your first program in C#*

The purpose of this Hello world program is to help you understand the basic syntax and requirements of a C# program, Lets try to understand everything in this program, by breaking it line by line:

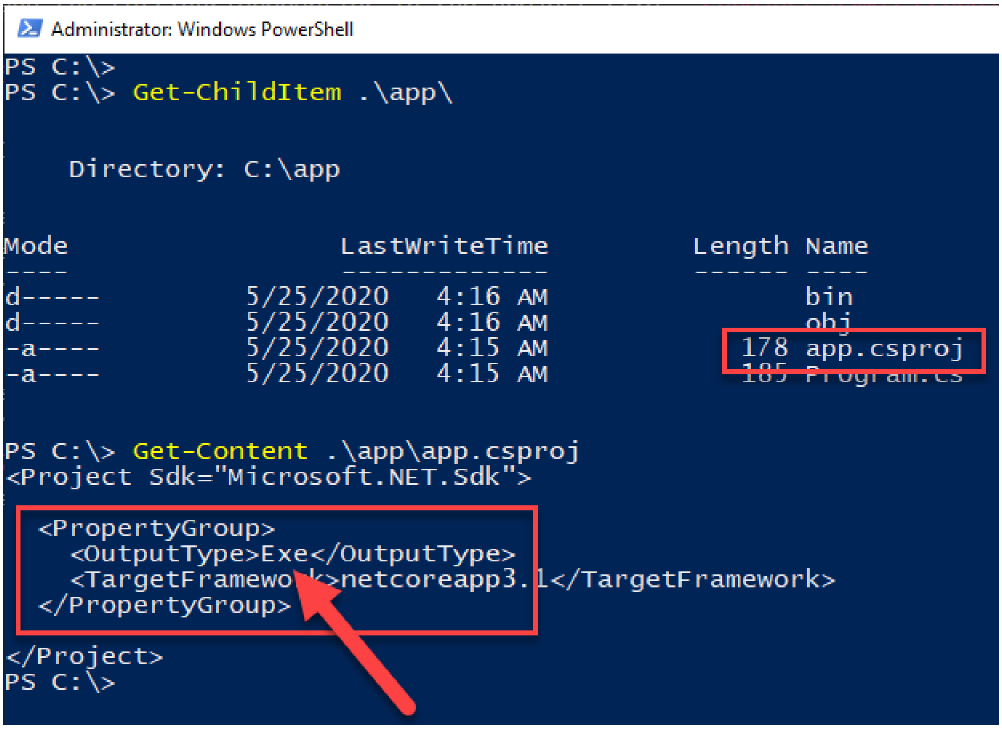
- `using System;` - The `using` keyword is used to call `class` libraries which are required to execute the project. Here `System` is the class library, that is required to call the `Console.WriteLine()` method later in the Hello world program.
- `namespace app` - Just like we have `System` class library we can also create custom class libraries like ‘`app`’ using the `namespace` keyword, which helps to organize our classes.
- `class Program` - Class is a blueprint or template that help define a type. A class has a name like ‘`Program`’ and can contain variables, methods and other types. Every C# program must have a class definition.
- `static` - This is a special keyword, and when it is used with a method then the method can be called or accessed directly without creating instance of class in which it is defined.

- `void` - It is a return type of the method which means that the method doesn't returns any value.
- `Main()` - This is the entry point of the program that means in a console application, the first statement in the `Main()` method is the starting point of the program.
- `string[] args` - This is used to supply command-line arguments, where `string[]` is an array of `String` data type with array name '`args`'
- `Console.WriteLine("Hello World!");` - `Console` is a class in `System` namespace, that has a method called `WriteLine()` that can write a string value like "Hello World!" supplied to the standard output.
- `;` - Semicolons are statement terminators, which mean any expression, assignment or declaration statements are supposed to terminate with a semicolon, that represents line termination character.
- `{ }` - Curly braces represent start and end of code block, like start or end of a class or method body.

Now that we understand the Hello world program, we can go ahead and run the program using the `dotnet run` command, which is a convenient way to run your application source code from the command line, make sure you are in the application directory when you are running this command. The `dotnet run` can automatically build the project if necessary, but you can also explicitly build the project using the command: `dotnet build` that builds but doesn't run the program.

```
dotnet run
dotnet build
```

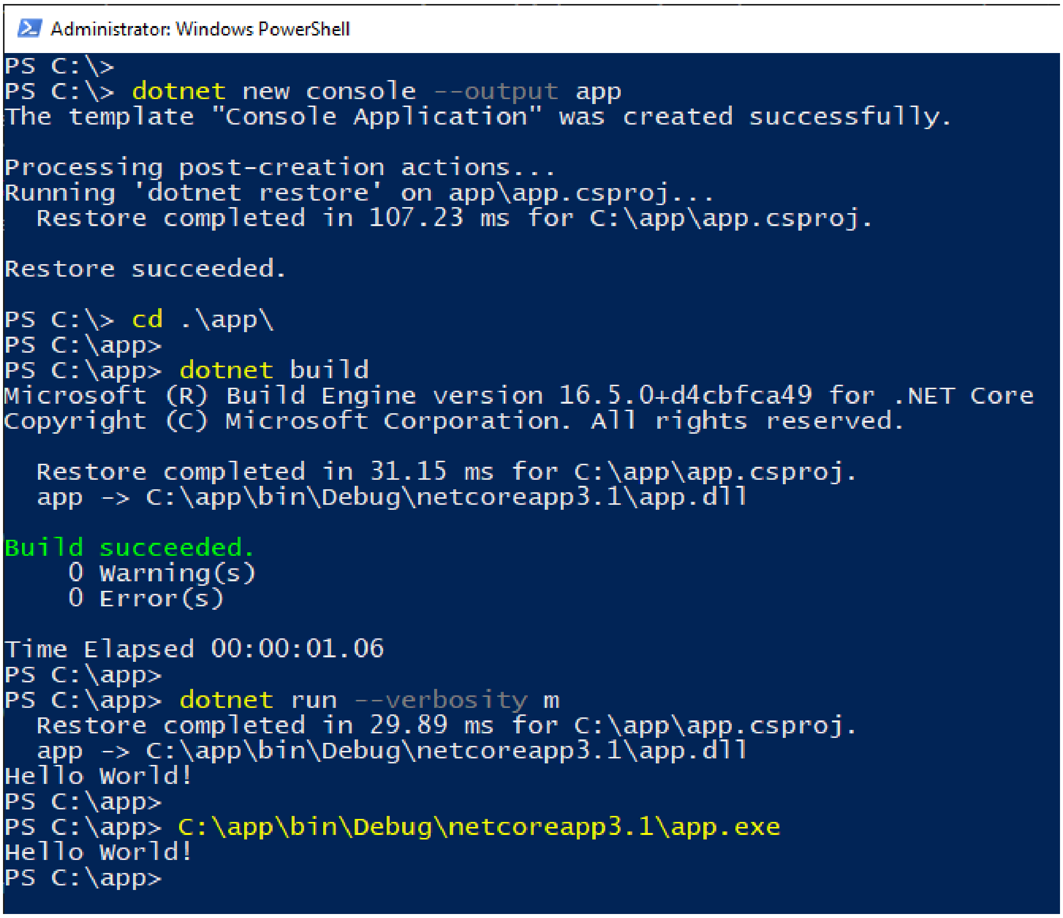
The `dotnet build` command converts all the project source code, references and dependencies into intermediate language (IL) with extension `.dll` and depending upon the type of project and the settings\properties in the project file: `<project name>.csproj` other files like executable `.exe` will be included in the project build.



```
PS C:\>
PS C:\> Get-ChildItem .\app\


    Directory: C:\app


Mode                LastWriteTime         Length Name
----                -------------         ------ ----
d-----        5/25/2020   4:16 AM                bin
d-----        5/25/2020   4:16 AM                obj
-a----        5/25/2020   4:15 AM            178 app.csproj
-a----        5/25/2020   4:15 AM            185 Program.cs


PS C:\> Get-Content .\app\app.csproj
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>netcoreapp3.1</TargetFramework>
  </PropertyGroup>

</Project>
PS C:\>
```

*Figure 1-3. C# project file .csproj*

All the output file on build will be written by default to the location: `<project name>\bin\<configuratio`



```
PS C:\>
PS C:\> dotnet new console --output app
The template "Console Application" was created successfully.

Processing post-creation actions...
Running 'dotnet restore' on app\app.csproj...
  Restore completed in 107.23 ms for C:\app\app.csproj.

Restore succeeded.

PS C:\> cd .\app\
PS C:\app>
PS C:\app> dotnet build
Microsoft (R) Build Engine version 16.5.0+d4cbfca49 for .NET Core
Copyright (C) Microsoft Corporation. All rights reserved.

  Restore completed in 31.15 ms for C:\app\app.csproj.
  app -> C:\app\bin\Debug\netcoreapp3.1\app.dll

Build succeeded.
    0 Warning(s)
    0 Error(s)

Time Elapsed 00:00:01.06
PS C:\app>
PS C:\app> dotnet run --verbosity m
  Restore completed in 29.89 ms for C:\app\app.csproj.
  app -> C:\app\bin\Debug\netcoreapp3.1\app.dll
Hello World!
PS C:\app>
PS C:\app> C:\app\bin\Debug\netcoreapp3.1\app.exe
Hello World!
PS C:\app>
```

*Figure 1-4. Build a C# project into assemblies and executable*

## Namespaces

C# Namespaces are used to neatly organize classes as the project continues to grow, more than that Namespaces also provide a way to separate your code and avoid any `class` naming conflicts. In other words Namespaces are containers which contain other namespaces and classes that are logically related to each other.

To define a namespace we simply use keyword: `namespace` followed by the name of your namespace, like `demo` as demonstrated in the following example, and then all the classes: `Class1`, `Class2` and so on.. are defined in this container within the body of the namespace enclosed by the brackets `{ }`.

In the above example, `System` is a namespace defined in .Net Framework that contains the fundamental and base classes like `Console`, which has a method called `WriteLine()` that can write the specified data to the standard output stream. At the top of the program, in our example, we used `using` directive followed by the name of namespace , which allows the use of types in the namespace, without fully qualified name like `Console.WriteLine()` instead of `System.Console.WriteLine()`.

## Class

In simple terms a class is a blueprint or prototype that is used to define an object, which is a programmatic representation of a real world object that has characteristics or properties such as color, height, width and can perform functionalities such as start, stop, move, jump etc.

Let's take an example, that we want to define a Car in C#, first thing we have do is create a class declaration for that using the following code snippet:

Here, the first thing you notice is `public` which is an access modifier, followed by the keyword `class` and the name of the class. then the body of class is enclosed in open and close brackets `{ }`.

> **NOTE:** This chapter is just C# primer to get readers to speed and for now we are only covering the very basics to build a foundation for the readers. Later in this book we will deep dive into classes and object oriented programming in C# cover all the aspects in detail.

## Members

All the constants, properties and methods defined inside the body of a `class` are known as members of that class, as shown in the Example 1-3 of the above subsection. Generally speaking members can be:

1. Property - Properties are attributes or characteristics of the class, which by default are `private` but, if they are `public` they can be accessed using class objects to change the characteristics of the Object. Like for `Car` Class, `color`, `maxSpeed` are properties that can have some default value like `color = "red"`, but these can be accessed and changed on each instance of this class called object.

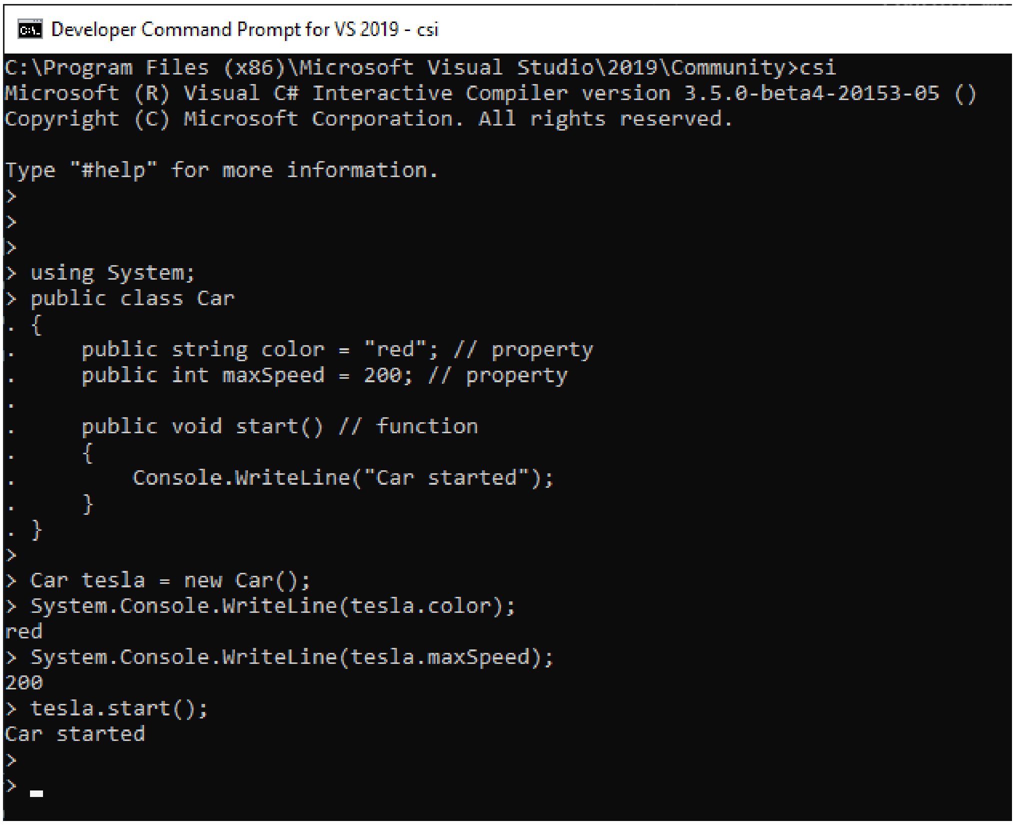
2. Method - Methods are functions defined in a class, which have access to all the members of a class. The purpose is to perform a functionality for the object of the class, for example `Car` Class has methods like: `start()` and `stop()`.

C# language doesn't support any global variables or methods, that means all the entry points of the program, which the `Main()` method is also defined inside a class. More than that class is just a blueprint and we have to instantiate the class or in other words create objects of the class to access the members.

To create an object from class we use the following syntax in C#:

```
<Name Of Class> Object = new <Name Of Class>();
```

So, to create a `tesla` object from `Car` class, we will use the `new` keyword as demonstrated in the Example 1-4, and then access the members of this object using the `(.)` Dot operator in C#.


Developer Command Prompt for VS 2019 - csi

```
C:\Program Files (x86)\Microsoft Visual Studio\2019\Community>csi
Microsoft (R) Visual C# Interactive Compiler version 3.5.0-beta4-20153-05 ()
Copyright (C) Microsoft Corporation. All rights reserved.

Type "#help" for more information.
>
>
>
> using System;
> public class Car
. {
.     public string color = "red"; // property
.     public int maxSpeed = 200; // property
.
.     public void start() // function
.     {
.         Console.WriteLine("Car started");
.     }
. }
>
> Car tesla = new Car();
> System.Console.WriteLine(tesla.color);
red
> System.Console.WriteLine(tesla.maxSpeed);
200
> tesla.start();
Car started
>
> _
```

*Figure 1-5. Creating Object from C# Class and accessing members*

## C# Programming Elements

This subsection provides an overview and basic information about all elements of C# programming language. Later in this book most of these topics have their dedicated chapter for readers to delve deeper into the topics. For now we are covering these topics to in simple easy to digest small sub-sections, that will introduce the C# language to the reader and build some momentum for the deep dives later in this book.

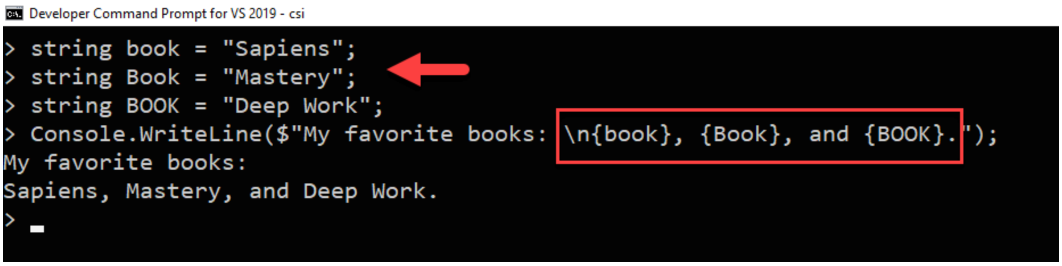
### Comments

Comments in any programming language are a handy way to document the code to make it more readable, and makes it easy for other developers to understand it. Any commented part in code is not executed and can also be used when testing alternative code.

In C#, to create a single-line comment we use double forward-slash ( `//` ), whereas a multi-line comment starts with `/*` and ends with `*/`.

## Case Sensitivity

C# is a case sensitive programming language that means `book`, `Book` and `BOOK` are 3 different identifiers, irrespective of the fact all three of them are the same word, but have different cases.



Developer Command Prompt for VS 2019 - csi

```
> string book = "Sapiens";
> string Book = "Mastery";
> string BOOK = "Deep Work";
> Console.WriteLine($"My favorite books: \n{book}, {Book}, and {BOOK}.");
My favorite books:
Sapiens, Mastery, and Deep Work.
>
```

*Figure 1-6. C# case sensitive identifiers*

## Using Directive and Statement

In C# language, the `using` directive has three main high-level purposes, first is to provide a way to utilize the types in a namespace, so that you don't have to fully qualify the types and functions in the namespace.

Second, purpose is to give developers ability to create alias for namespaces, as demonstrated in the following example:

Finally, the third purpose is to provide an option to release resources automatically. Any object that implements an `IDisposable` interface, which has only one method: `Dispose()` where you can write the code to dispose of any resources. This is where `using` statement comes into picture, so that we can instantiate an object in the statement and at the end of the `using` statement block, `Dispose()` method is automatically called. Following example demonstrates this and will help you understand.

Output:

```
Howdy!
Dispose method called
```

## Types and Variables

All C# Data types or simple 'types' can be classified into two broad categories:

1. Value types
2. Reference types

### Values Types

Value type data types hold a data value within its own designated memory space and they can not contain `null` values. Following is a list of value data types you can use in your C# programs.

| .Net Type | Type Alias | Represents | Range | Default Value |
|---|---|---|---|---|
| System.Byte | byte | 8-bit unsigned integer | 0 to 255 | 0 |
| System.Boolean | bool | Boolean value | True or False | False |
| System.Char | char | 16-bit Unicode character | U+0000 to U+ffff | '\0' |
| System.Decimal | decimal | 128-bit precise decimal values with 28-29 significant digits | (+/-)1.0 x $10^{-28 \text{ to } (+/-)7.9 \text{ x } 10}28$ | 0.0M |
| System.Double | double | 64-bit double-precision floating point type | (+/-)5.0 x $10^{-324 \text{ to } (+/-)1.7 \text{ x } 10}308$ | 0.0D |
| System.Single | float | 32-bit single-precision floating point type | (+/-)1.5 x 10^-45 to (+/-)3.4 x 1038 | 0.0F |

| .Net Type | Type Alias | Represents | Range | Default Value |
|---|---|---|---|---|
| System.Int32 | int | 32-bit signed integer type | -2,147,483,648 to 2,147,483,647 | 0 |
| System.Int64 | long | 64-bit signed integer type | -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 | 0L |
| System.SByte | sbyte | 8-bit signed integer type | -128 to 127 | 0 |
| System.Int16 | short | 16-bit signed integer type | -32,768 to 32,767 | 0 |
| System.UInt32 | uint | 32-bit unsigned integer type | 0 to 4,294,967,295 | 0 |
| System.UInt64 | ulong | 64-bit unsigned integer type | 0 to 18,446,744,073,709,551,615 | 0 |
| System.UInt16 | ushort | 16-bit unsigned integer type | 0 to 65,535 | 0 |

## Reference Types

Reference types contain references to other objects and don't store any actual data in a variable. In simpler words they store reference to a memory location. Reference types bring in the possibility of one or more variables to reference a single object, and similarly any action performed by any one variable changes the referenced object. C# provides some built-in reference types such as: `dynamic`, `object`, `string`. In order to declare your own reference types in C#, you can take advantage of the keywords: `class`, `interface` and `delegate`.

Following table will help you understand basic differences between value and reference types:

| | Value type | Reference type |
|---|---|---|
| Stores | Actual value | Memory location |
| Allocation | Stack, member | Heap |
| Nullable | Always has value | Maybe null |
| Default | 0 | null |
| Assignment | Copying actual data | Copying reference |

### Variables

A variable is the name of the storage location that is used to store values of data types supported by the programming language. In C# depending upon the data type of the variable, a memory location with a specific size and layout is assigned to the variable.

To define and initialize a variable we follow the below mentioned syntax:

```
// variable definition
<data-type> <variable-name>;

// multiple variable definition
<data-type> <variable-name1>, <variable-name2>, <variable-name3>;

// variable definition and initialization
<data-type> <variable-name> = value;
```

Here, `<data-type>` can be one of data types that we discussed in the previous subsection such as value types: `char`,`int`,`float` and reference types or in other words a user defined data types: `Employee`, `Car` etc. Following are some examples to demonstrate this:

When declaring a variable in your C# program, you must explicitly specify the data type, otherwise you can also use the '`var`' keyword to let the compiler implicitly decide the type of variable at compilation time.

## Operators, Operands and Expressions

Expressions are combinations or sequences of operands and operators and once an expression is evaluated, it returns a value. The operators of an expression represent operations to apply on the operands. For example: `(x + y) * z` is an expression in which `x`, `y`, `z` are operands and `+`, `*` are operators.

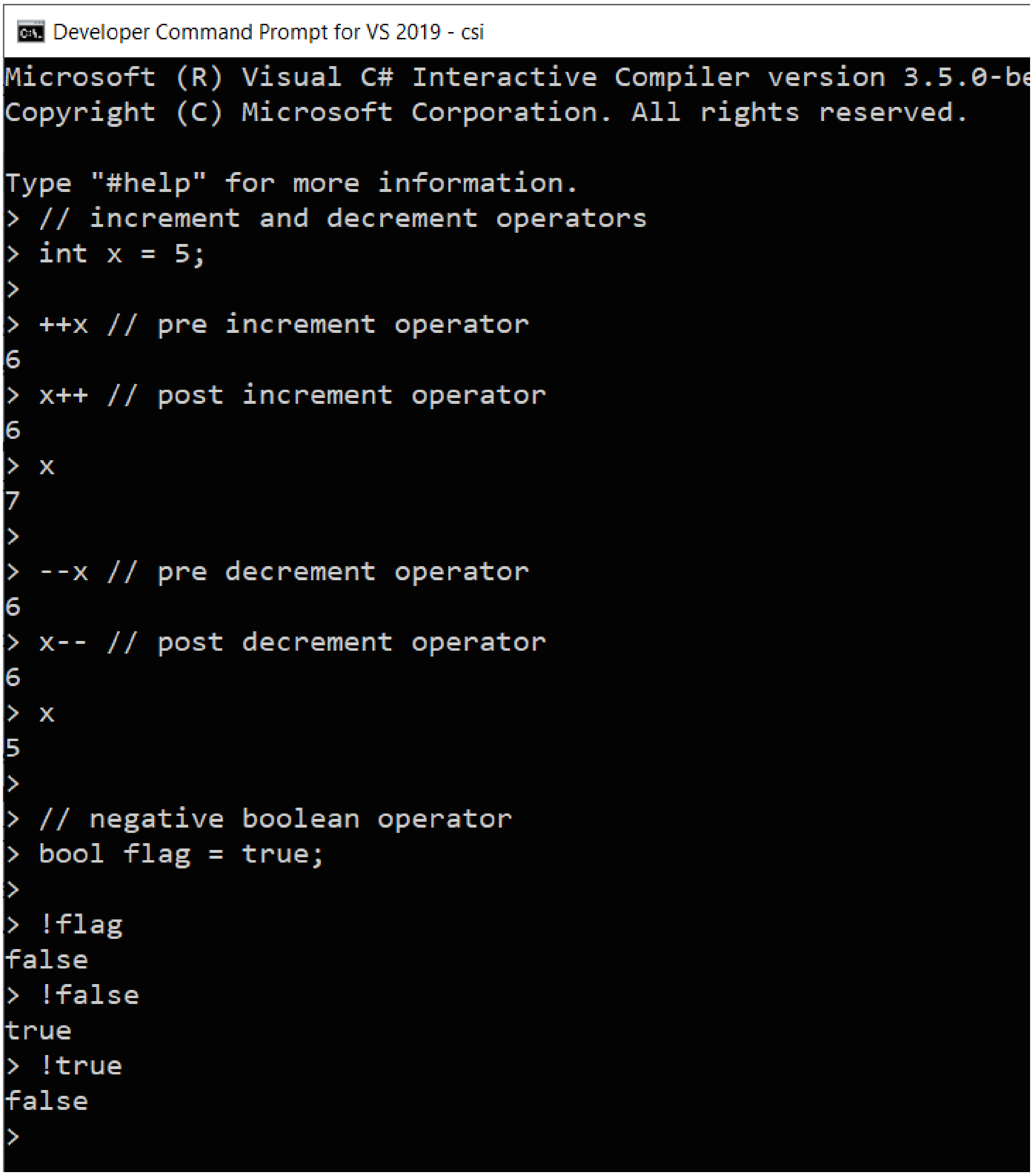
### Operators

The operators in the C# Language can be categorized into, following 3 broad categories:

1. Unary operator - Unary operators take one operand to perform the operation and either prefixed or postfixed to the operand. Some common use cases can be increment (++), decrement (`--`) and negative boolean (`!`) operators, below are some examples that will help you understand.

   Example 1-11. C# increment and decrement operators

   Figure 1-7. C# Unary operators



Developer Command Prompt for VS 2019 - csi

```
Microsoft (R) Visual C# Interactive Compiler version 3.5.0-be
Copyright (C) Microsoft Corporation. All rights reserved.

Type "#help" for more information.
> // increment and decrement operators
> int x = 5;
>
> ++x // pre increment operator
6
> x++ // post increment operator
6
> x
7
>
> --x // pre decrement operator
6
> x-- // post decrement operator
6
> x
5
>
> // negative boolean operator
> bool flag = true;
>
> !flag
false
> !false
true
> !true
false
>
```

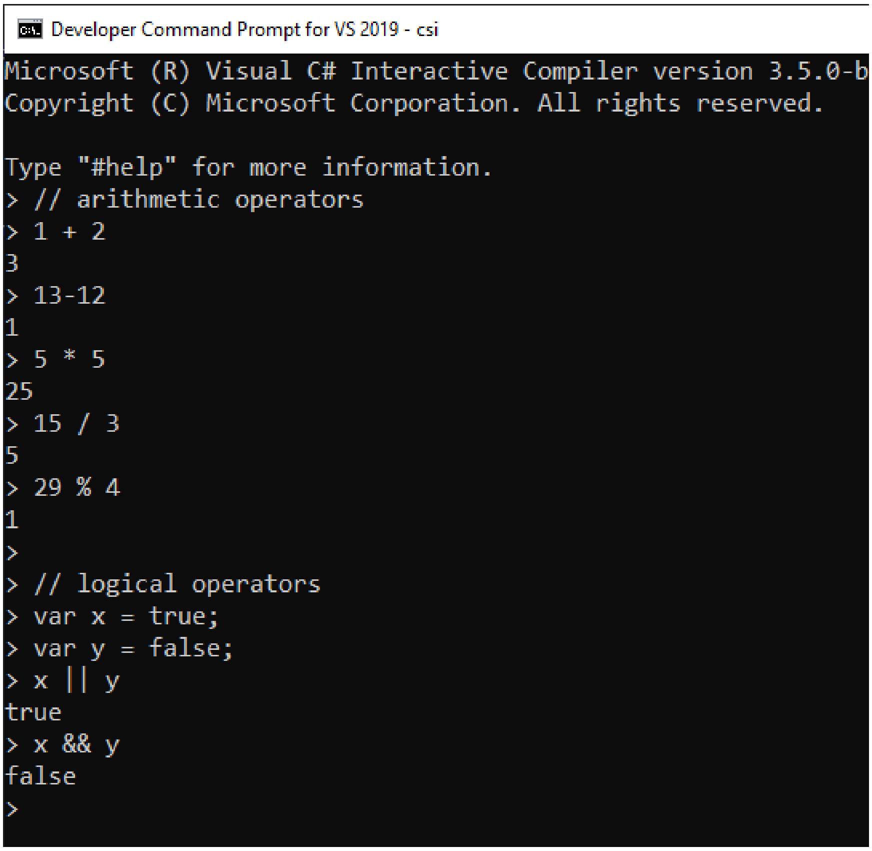
*Figure 1-7. C# Unary operators*

2. Binary operator - Binary operators take two operands to perform the operation, and operator is in between the two operands, as shown in the following examples binary

operators can be used as arithmetic (+,-,*,/,%) operators or logical OR (||) and logical AND (&&) operators.

Example 1-12. C# arithmetic and logical operators

Figure 1-8. C# Arithmetic and Logical operators



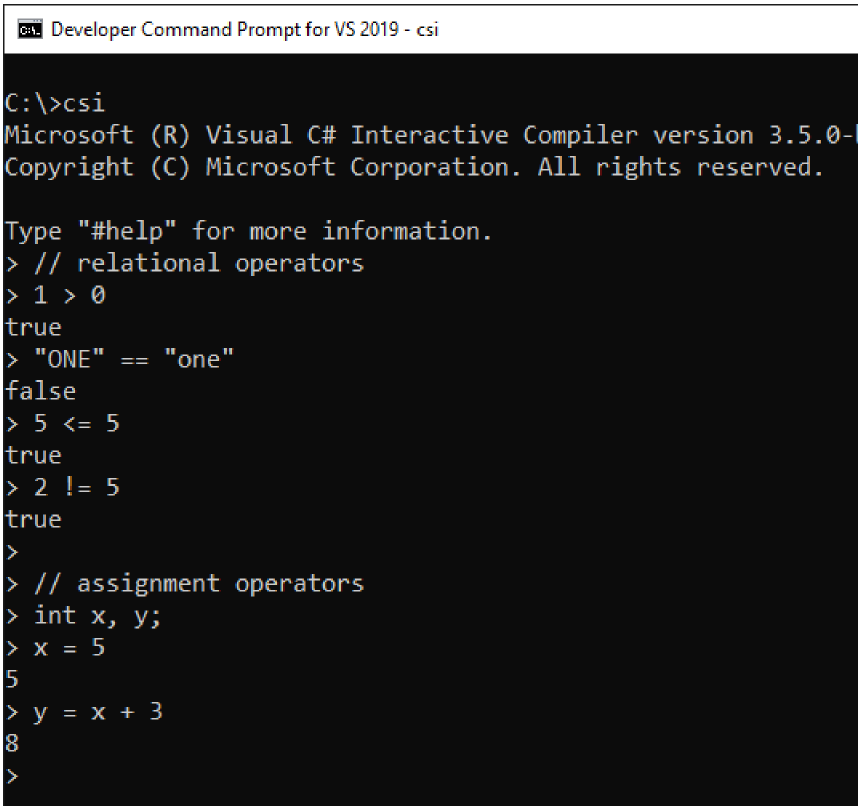
```
Developer Command Prompt for VS 2019 - csi
Microsoft (R) Visual C# Interactive Compiler version 3.5.0-b
Copyright (C) Microsoft Corporation. All rights reserved.

Type "#help" for more information.
> // arithmetic operators
> 1 + 2
3
> 13-12
1
> 5 * 5
25
> 15 / 3
5
> 29 % 4
1
>
> // logical operators
> var x = true;
> var y = false;
> x || y
true
> x && y
false
>
```

*Figure 1-8. C# Arithmetic and Logical operators*

Relation (>,>=,<,<=,==,!=) and assignment (=) operators also fall under the category of the binary operators, following are some examples to demonstrate this:

Example 1-13. C# relational and assignment operators

Figure 1-9. C# Relational and Assignment operators


Developer Command Prompt for VS 2019 - csi

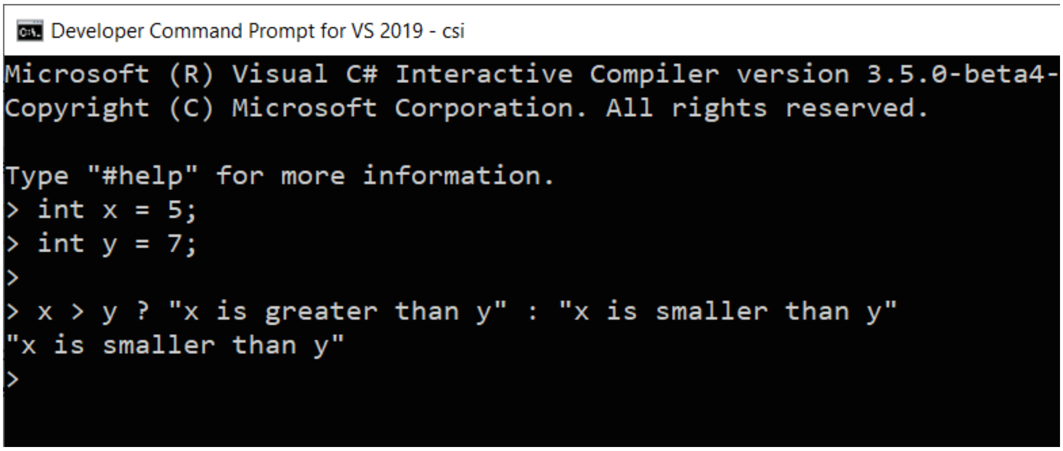
```
C:\>csi
Microsoft (R) Visual C# Interactive Compiler version 3.5.0-
Copyright (C) Microsoft Corporation. All rights reserved.

Type "#help" for more information.
> // relational operators
> 1 > 0
true
> "ONE" == "one"
false
> 5 <= 5
true
> 2 != 5
true
>
> // assignment operators
> int x, y;
> x = 5
5
> y = x + 3
8
>
```

*Figure 1-9. C# Relational and Assignment operators*

3. Ternary operator - Ternary operator is a special operator that is used in decision making or to check conditions, this operator takes three operands in syntax mentioned below:

```
<condition> ? <if true> : <if false>
```

Following is an example that will help you understand this better.



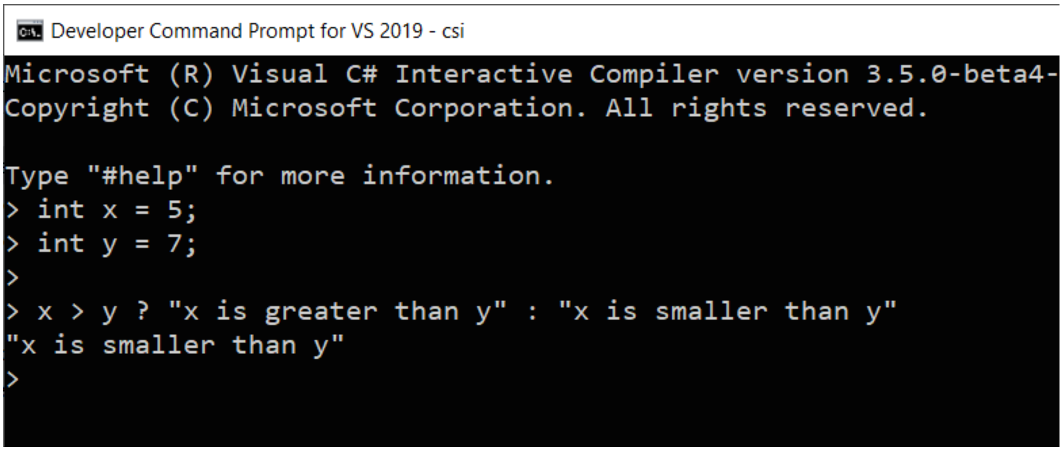
```
Microsoft (R) Visual C# Interactive Compiler version 3.5.0-beta4-
Copyright (C) Microsoft Corporation. All rights reserved.

Type "#help" for more information.
> int x = 5;
> int y = 7;
>
> x > y ? "x is greater than y" : "x is smaller than y"
"x is smaller than y"
>
```

*Figure 1-10. C# Ternary operator*

## Basic Input and Output

In order to output something in your C# program, we can use the `WriteLine()` or `Write()` method of namespace `System` and class `Console` as demonstrated in the following example. The only difference between `Write()` and `WriteLine()` method is that the former prints the string passed as an argument and stays one the same line, while the latter prints the string and then moves to the start of the next line.

To take user input in a C# program, we can simply use the `ReadLine()` method, also included in class `Console`, part of namespace `System`.



```
Microsoft (R) Visual C# Interactive Compiler version 3.5.0-beta4-
Copyright (C) Microsoft Corporation. All rights reserved.

Type "#help" for more information.
> int x = 5;
> int y = 7;
>
> x > y ? "x is greater than y" : "x is smaller than y"
"x is smaller than y"
>
```

*Figure 1-10. C# Ternary operator*

## Conditional statements

C# like any other high level programming language needs a mechanism to execute statements on basis on conditions, also known as Conditional Statements. Which can be further classified into two categories:

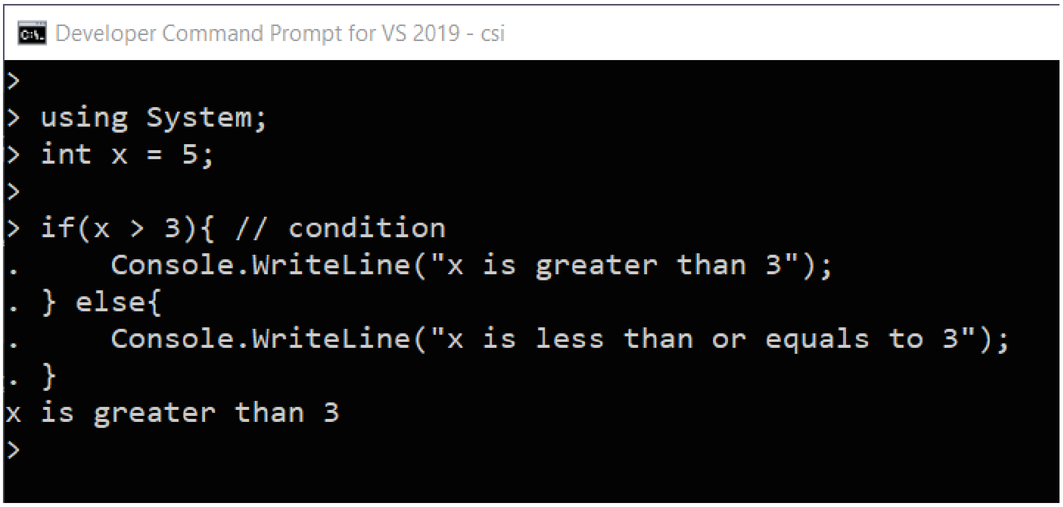
1. Conditional Branching - `if`, `if..else`, `switch`
2. Conditional Looping - `for`, `while`, `do..while`, `foreach`

Let's look into some simple examples and understand conditional branching and we will cover conditional loop in the next subsection: "Loops and Iterations".

### If and If..Else statement

An `if` statement allows you to test whether a condition is met or not, if the condition is met, then the statements in the body of `if` will be executed.

Similarly you can also use `else` to execute statements when condition is not met.



Developer Command Prompt for VS 2019 - csi

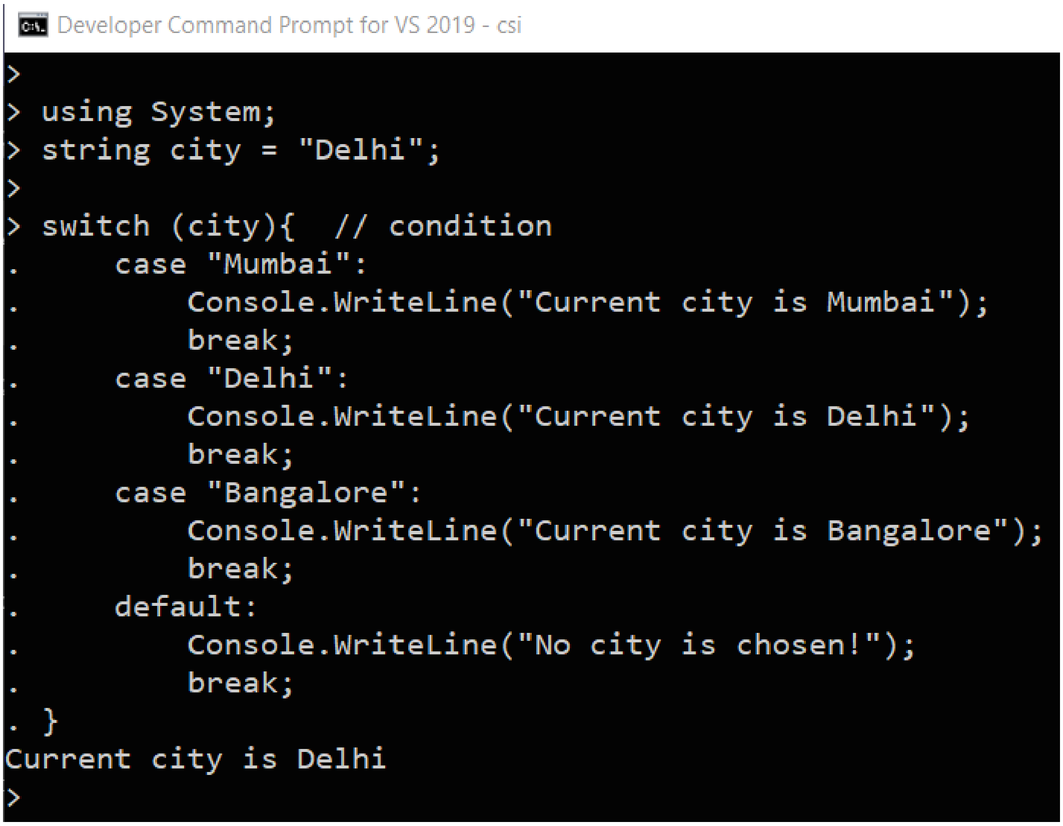
```
>
> using System;
> int x = 5;
>
> if(x > 3){ // condition
.     Console.WriteLine("x is greater than 3");
. } else{
.     Console.WriteLine("x is less than or equals to 3");
. }
x is greater than 3
>
```

*Figure 1-11. C# If..Else statement*

Or use multiple `if..else if..else if..else` to test multiple conditions and execute the statement based on if the conditions are met or not.

## Switch statement

Switch statements are basically an enhanced version of `if..elseif..else` statements, and are very useful when you want to test multiple conditions without writing too much code. In simpler terms it is easy to read and less verbose.



Developer Command Prompt for VS 2019 - csi

```
>
> using System;
> string city = "Delhi";
>
> switch (city){  // condition
.     case "Mumbai":
.         Console.WriteLine("Current city is Mumbai");
.         break;
.     case "Delhi":
.         Console.WriteLine("Current city is Delhi");
.         break;
.     case "Bangalore":
.         Console.WriteLine("Current city is Bangalore");
.         break;
.     default:
.         Console.WriteLine("No city is chosen!");
.         break;
. }
Current city is Delhi
>
```

*Figure 1-12. C# switch statement*

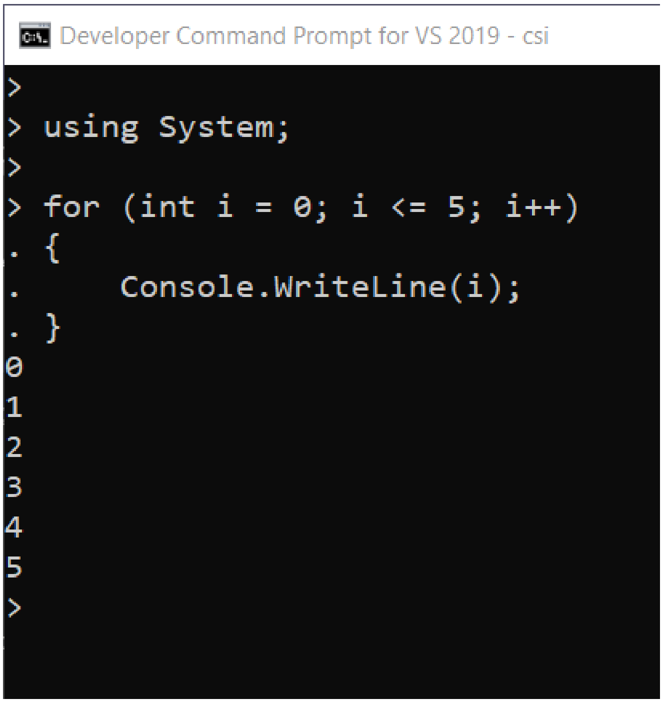
## Loops and Iterations

Loops and Iterations in any programming language are used to repeat one or more statements over and over based on a condition. Here we are only going to look into `for` loop and remaining loops will be covered in detail in later chapters of this book.

Syntax:

```
For (initialize; condition; increment)
{
    < statement >
    < statement >
}
```

Let's take a very simple example and create a `for` loop to iterate numbers from `0` to `5`.



```
Developer Command Prompt for VS 2019 - csi
>
> using System;
>
> for (int i = 0; i <= 5; i++)
. {
.     Console.WriteLine(i);
. }
0
1
2
3
4
5
>
```

*Figure 1-13. C# For loop*

Iteration statements are used to repeatedly execute an embedded statement. This group contains the while, do, for, and foreach statements.

## Exceptions `Try..Catch..Finally`

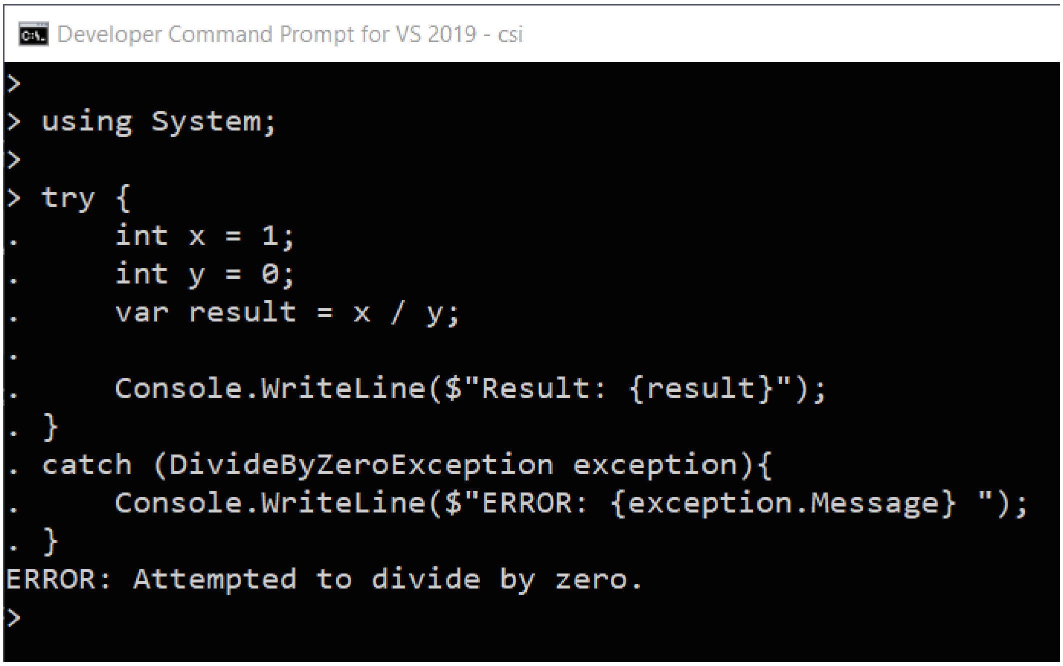
When in a C# program an error occurs, C# will normally stop the further execution program and generate an error message, which is also known as an Exception or an error thrown by the program. To handle such exception C# like any other programming language has a mechanism called error handling, which basically consists of 3 blocks mentioned below:

1. Try - Block of code which is tested or tried for errors
2. Catch - Block of code to handle the errors. There can be more than one `catch` blocks.
3. Finally - Block of code that allow developers to execute code regardless of result (exception or no exception)

So, code in the `try` block will be tested and if an exception is thrown it automatically gets captured by the `catch` block, where you can display custom messages for the exception or handle the exception by running some other statements. At last we have the `finally` block which will execute every time regardless of the fact an exception was thrown or not, this block is optional and can be omitted if not required. The purpose of `finally` block is resource cleanup, garbage collection or controlling the

Following is a simple example to demonstrate `try..catch` by intentionally raising a `DivideByZeroExcepti`

In the above example any error thrown in the `try` block will be caught in the `catch` block. irrespective of the fact that it is a `DivideByZeroException` or some other exception, which is often not a very good idea and we should only catch exceptions which we understand and want to handle. So, to specifically handle or filter a certain type of exception we can define a catch block with arguments like `catch (DivideByZeroException)`, and even create a variable of the exception type to use it inside the catch block, which has built-in `Message` property.



```
>
> using System;
>
> try {
.     int x = 1;
.     int y = 0;
.     var result = x / y;
.
.     Console.WriteLine($"Result: {result}");
. }
. catch (DivideByZeroException exception){
.     Console.WriteLine($"ERROR: {exception.Message} ");
. }
ERROR: Attempted to divide by zero.
>
```

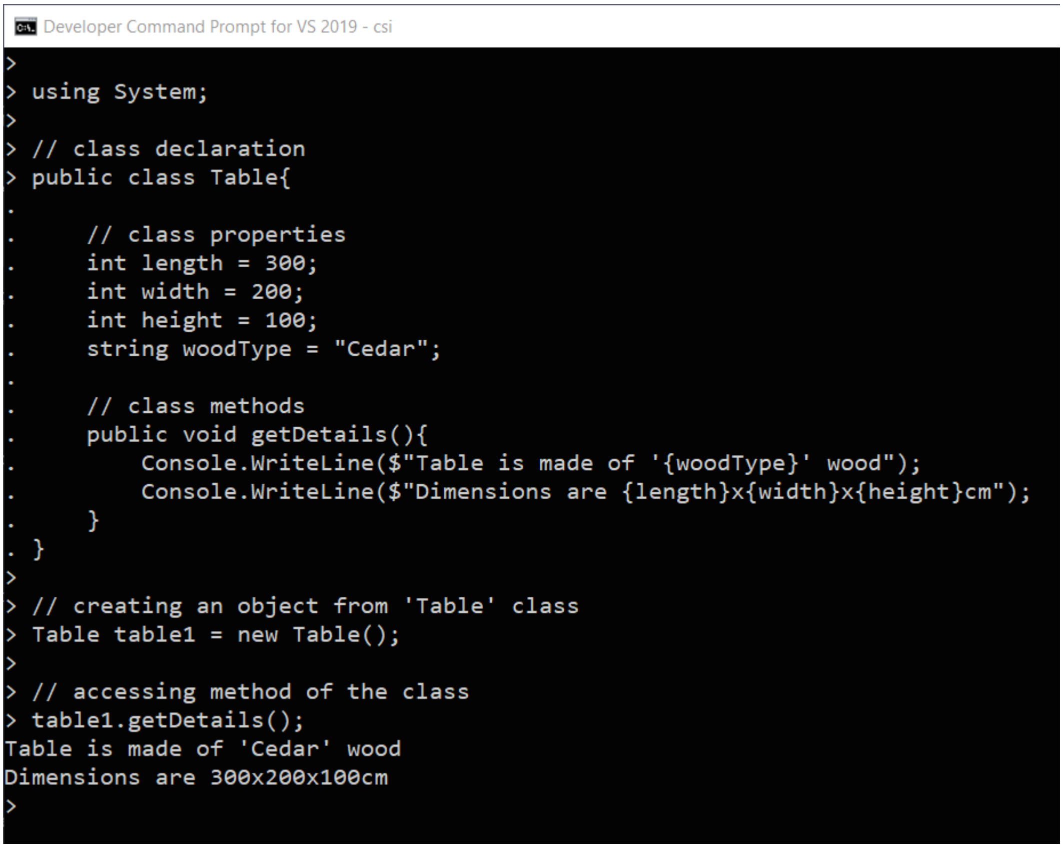
*Figure 1-14. C# For loop*

At last if a `finally` block exists, it will be executed after the execution of the `try..catch` block.

## Classes and objects

Classes are also C# types, basically a blueprint of data structure that combine `properties` and `methods` in a single unit. A C# Class is declared using the `class` keyword, prefixed with an access modifier: like `public` and followed by the body of the class and its members as shown in the following example:

When a class is instantiated it is called an Object and it can be used to access the properties and methods of the class from which it is instantiated using the Dot (`.`) operator followed by the member of the class as shown in the following example:

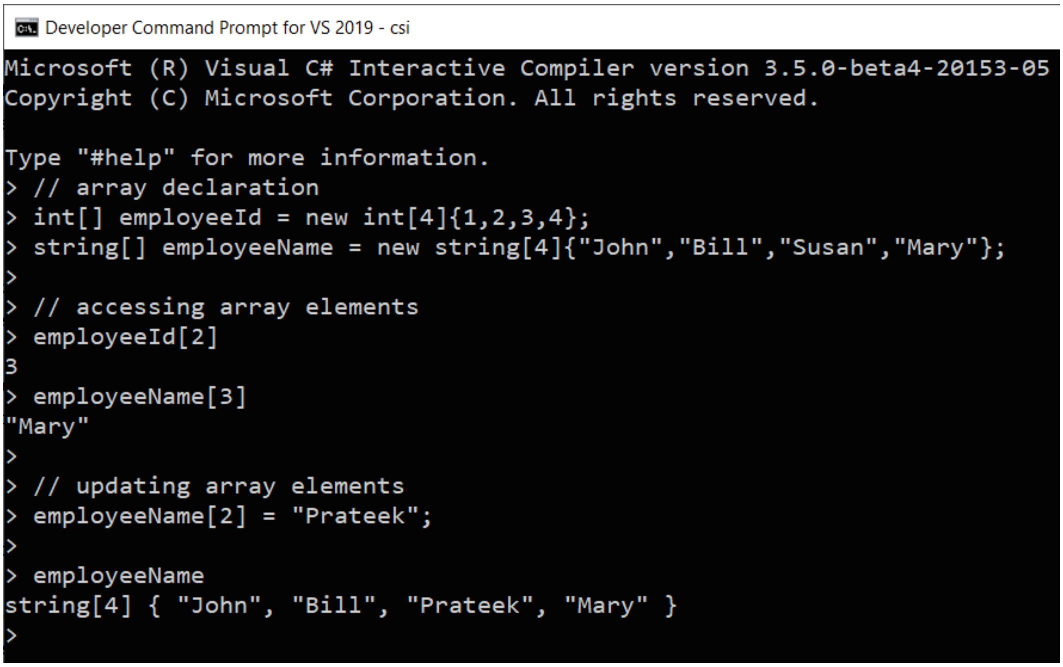


```
>
> using System;
>
> // class declaration
> public class Table{
.
.     // class properties
.     int length = 300;
.     int width = 200;
.     int height = 100;
.     string woodType = "Cedar";
.
.     // class methods
.     public void getDetails(){
.         Console.WriteLine($"Table is made of '{woodType}' wood");
.         Console.WriteLine($"Dimensions are {length}x{width}x{height}cm");
.     }
. }
>
> // creating an object from 'Table' class
> Table table1 = new Table();
>
> // accessing method of the class
> table1.getDetails();
Table is made of 'Cedar' wood
Dimensions are 300x200x100cm
>
```

*Figure 1-15. C# class and object*

## Arrays

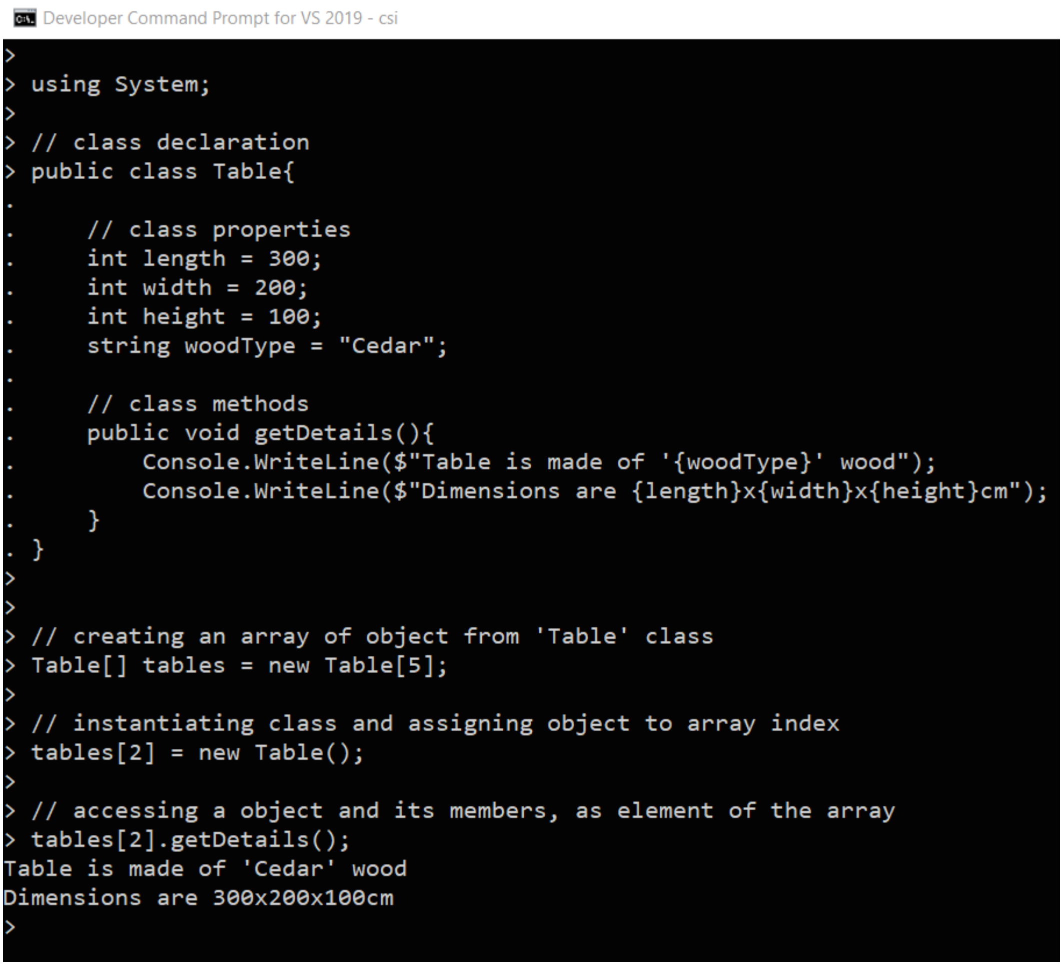
C# arrays are data structures that store a fixed number of homogeneous data types, grouped as a single unit. These individual data types are called elements of the array and can be accessed by its numerical index. Array indexes start at `0` so that second element is at index `1`, following examples will help you understand arrays better:


Developer Command Prompt for VS 2019 - csi

```
Microsoft (R) Visual C# Interactive Compiler version 3.5.0-beta4-20153-05
Copyright (C) Microsoft Corporation. All rights reserved.

Type "#help" for more information.
> // array declaration
> int[] employeeId = new int[4]{1,2,3,4};
> string[] employeeName = new string[4]{"John","Bill","Susan","Mary"};
>
> // accessing array elements
> employeeId[2]
3
> employeeName[3]
"Mary"
>
> // updating array elements
> employeeName[2] = "Prateek";
>
> employeeName
string[4] { "John", "Bill", "Prateek", "Mary" }
>
```

*Figure 1-16. C# Arrays*

You can also create an array of objects, such that each element of the array is an object of a class. Let's take the same example from the previous subsection where we created a `class Table`, but this time we will create an array of objects as demonstrated in the following example:



Developer Command Prompt for VS 2019 - csi

```
>
> using System;
>
> // class declaration
> public class Table{
.
.     // class properties
.     int length = 300;
.     int width = 200;
.     int height = 100;
.     string woodType = "Cedar";
.
.     // class methods
.     public void getDetails(){
.         Console.WriteLine($"Table is made of '{woodType}' wood");
.         Console.WriteLine($"Dimensions are {length}x{width}x{height}cm");
.     }
. }
>
>
> // creating an array of object from 'Table' class
> Table[] tables = new Table[5];
>
> // instantiating class and assigning object to array index
> tables[2] = new Table();
>
> // accessing a object and its members, as element of the array
> tables[2].getDetails();
Table is made of 'Cedar' wood
Dimensions are 300x200x100cm
>
```

*Figure 1-16. C# Array of objects*

## Summary

In this chapter we learned some key characteristics of C# programming language that differentiates it from other languages and its evolution from version `1.0` to the current version `8.0`. Then we looked into underlying concepts and learned how a C# source code is converted into Intermediate language by the compiler in compile time and then to the machine code using .Net framework in run time. We also build and ran our first .Net project, a C# 'Hello World!' using `dotnet` CLI. Later in this chapter we covered the elements of C#

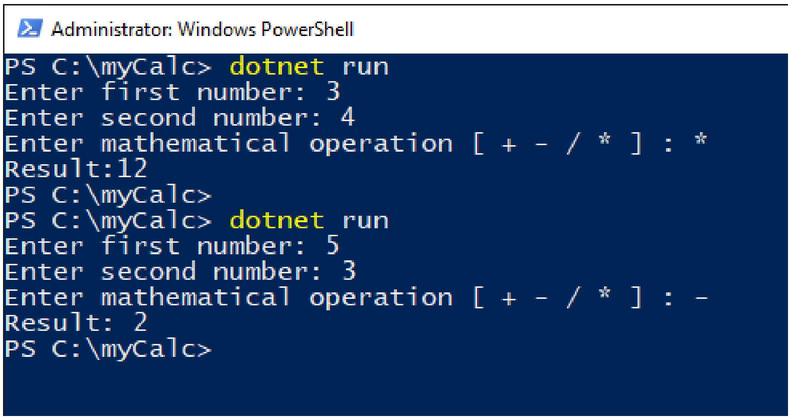
programming languages, such as data types, variables, arrays, conditional branching and iterations, instantiating class into objects and handling errors.

## Key Pointers

- C# is a case sensitive language.
- The `dotnet run` can automatically build the project if necessary, but you can also explicitly build the project using the command: `dotnet build`
- Every C# program must have a class definition.
- `Main()` method is the entry point of a C# program.
- In C#, to create a single-line comment we use double forward-slash ( `//` ), whereas a multi-line comment starts with `/*` and ends with `*/`.
- Alias of namespace can be created like, `using Con = System.Console`
- '`var`' keyword to let the compiler implicitly decide the type of variable at compilation time.
- Capture user input using the `System.Console.ReadLine()` method and print to standard output using the `Console.WriteLine()` method.
- `Static` methods can be accessed or called directly without instantiating the class in which method has been defined.

## Exercises and Assignments

**Assignment 1-1 :** Create a simple C# command line calculator (executable like `myCalc.exe`), that can take two operands as input, like 5 and 3 and one of arithmetic operation as an input like: `+`, `-`, `/`, `*`, `%`, and provide results on your console by performing the arithmetic operation on the operands as shown in the following image. Solution to this assignment is at the end of the book under "Solutions to Exercises" section.



```
PS C:\myCalc> dotnet run
Enter first number: 3
Enter second number: 4
Enter mathematical operation [ + - / * ] : *
Result:12
PS C:\myCalc>
PS C:\myCalc> dotnet run
Enter first number: 5
Enter second number: 3
Enter mathematical operation [ + - / * ] : -
Result: 2
PS C:\myCalc>
```

*Assignment 1-1*

***Hint:*** *Use `Console.ReadLine()` read user input and `Convert.ToInt32()` to convert user input `string` to `integer` values to perform mathematical operations.*

## Reading Recommendations

- Official Microsoft Documentation for C# Language
- Introduction to C# and .NET Framework
- .NET Core CLI Overview
- `using` directive in C#
- `static` keyword in C#
- C# Types, variables and values

# Chapter 2: Classes and Objects

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

## Introduction

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

## Procedural vs Object-oriented programming

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

## .Net Class Library

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

## Classes and Objects

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

### PowerShell Class

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

### C# Class

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

## Object Introspection

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

### Inspecting PowerShell objects for properties and methods

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

### Inspecting C# objects for properties and methods

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

### Inspecting Constructors in C# objects

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

### typeof() vs GetType() vs ‘is’

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

## Class Members

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

## Fields and Properties

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

### PowerShell Class Properties

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

#### Default values to PowerShell class properties

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

#### PowerShell Cast initialization

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

#### Static Properties in PowerShell

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

### C# Class Fields, Properties, and Accessors

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

#### Fields

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

#### Properties and Accessors

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

#### Initial values to C# class properties

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

#### Static Properties in C#

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

#### C# Object Initializer

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

## Methods

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

### PowerShell Class Methods

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

#### Static methods in PowerShell Class

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

#### Instance methods in PowerShell Classes

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

### C# Class Methods

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

#### Static methods in C# Class

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

#### Instance methods in C# Class

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

## Constructors

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

### Constructors in PowerShell Classes

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

### Constructors in C# Classes

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

## Summary

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

## Key Pointers

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

## Exercises and Assignments

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

## Reading Recommendations

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

# Chapter 3: Variables

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

## Introduction

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

### Value Types

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

### Reference Types

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

## Naming conventions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

### Naming PowerShell variables

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

### Naming C# variables

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

## Declaring variables

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

## Constant variables

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

## Clearing and un-setting values of variables

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

## Variable scopes

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

### Variable scopes in PowerShell scripts

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

## Variable scopes with C# Classes

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

### Class level scope

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

### Method level scope

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

### Block level Scope

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

## Summary

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

## Key Pointers

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

## Exercises and Assignments

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

## Reading Recommendations

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

# Chapter 4: Console Input and Output

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

## Introduction

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

## What is a Console?

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

## Standard Input and Output

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

## Writing to Console

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

## Writing without a New-Line

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

## Reading from Console

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

## Reading numeric values

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

## Reading keys from Console

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

## Reading from Console Securely

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

## Clearing the Console

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

## Summary

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

## Key Pointers

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

## Exercises and Assignments

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

## Reading Recommendations

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

# Chapter 5: Passing Command-line Arguments

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

## Introduction

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

## Parameters and arguments

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

## `$args` Automatic Variable in PowerShell

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

## Main() and `args` in C# command-line applications

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

## Params() in PowerShell

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

## Environment.GetCommandLineArgs() method in .Net

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

## Summary

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

## Key Pointers

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

## Exercises and Assignments

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

## Reading Recommendations

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

# Chapter 6 - Collections

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

## Introduction

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

### Generic collections

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

### Non-Generic collections

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

## Array

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

## ArrayList

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

## List

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

## HashTable

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

## Dictionary

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

## Stack

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

## Queue

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

## HashSet

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

## Exercises and Assignments

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

## Reading Recommendations

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/powershell-to-csharp.

# Solutions to Exercises