Logo

Description automatically generated

Day 1

Object Oriented Development with .Net

.NET Development Platform

Console Application

Basic Code Structures

Coding Conventions

Syntax

Debugging

Data Types

Exception Handling

Contents

[NET Development Platform 2](#_Toc207020440)

[The C# Language 2](#_Toc207020441)

[Microsoft .NET 2](#_Toc207020442)

[Visual Studio 2](#_Toc207020443)

[Create Your First C# Console Application 3](#_Toc207020444)

[Interactive Development Environment (IDE) 5](#_Toc207020445)

[Console Read and Write 6](#_Toc207020446)

[Basic Code Structures 7](#_Toc207020447)

[Coding Conventions 10](#_Toc207020448)

[Naming 10](#_Toc207020449)

[Case 10](#_Toc207020450)

[Indentation 10](#_Toc207020451)

[Syntax 11](#_Toc207020452)

[Compilation 11](#_Toc207020453)

[Errors 13](#_Toc207020454)

[The Code Editor 14](#_Toc207020455)

[Debugging 15](#_Toc207020456)

[The Visual Studio Debugger 16](#_Toc207020457)

[Breakpoints 16](#_Toc207020458)

[Inspect Variables 17](#_Toc207020459)

[Data Types 19](#_Toc207020460)

[Common Data Types: 19](#_Toc207020461)

[Declaring Variables 19](#_Toc207020462)

[Initializing Variables 20](#_Toc207020463)

[Constants 21](#_Toc207020464)

[More on the ReadLine Method 22](#_Toc207020465)

[Exception Handling 23](#_Toc207020466)

[Try Catch 24](#_Toc207020467)

[Flow Charts 24](#_Toc207020468)

[Log Data 26](#_Toc207020469)

# NET Development Platform

## A purple hexagon with a letter c and a hashtag AI-generated content may be incorrect.The C# Language

Also known as "See-Sharp," it is an object-oriented programming language developed by Microsoft. Its a multi-purpose tool for building a wide range of applications, including Windows applications, web and mobile apps, and even video games.

## Microsoft .NET

In 2002, Microsoft changed application development by introducing the .NET Framework. This release, paired with the C# programming language and the Visual Studio IDE, provided developers with a powerful environment for creating a wide range of applications within the Windows ecosystem.

.NET encompasses two essential components: the **Common Language Runtime** (CLR) and the .NET Framework **Class Library**.

**CLR:** The **Common Language Runtime** acts as an intermediary layer between applications and the operating system. It assumes responsibility for code execution, ensuring efficient memory management while enforcing code accuracy (syntax).

**Class Library:** Is a comprehensive and object-oriented repository of reusable code that developers leverage in their application development process.

A purple text on a white background

Description automatically generated with low confidence

## Visual Studio

Microsoft Visual Studio, an integrated development environment (IDE), has stood the test of time for over two decades. This feature-rich application allows developers to write programs in a wide range of languages, including C#, F#, and Visual Basic.

Visual Studio’s capabilities extend far beyond basic coding tasks. It serves as a comprehensive debugger, publisher, and testing environment, and it offers a vast selection of extensions that enhance the overall development experience.

In the **Software Systems Developer (SSD) program**, you will use Visual Studio and the .NET Framework to create several types of applications, including:

* **Console applications** (command-line programs)
* **Web applications** (ASP.NET MVC, Razor Pages)
* **Cloud-based applications** (integrated with Azure)
* **APIs and services** (Web APIs)

|  |  |
| --- | --- |
| A red triangle with a white exclamation mark  AI-generated content may be incorrect. | To continue with this course, you must have Visual Studio installed on the Windows operating system. If you haven’t installed Visual Studio yet, please do so now and refer to the Installation notes on the Learning Hub for assistance if needed. |

Quiz Questions 1, 2, 3 and 4

# Create Your First C# Console Application

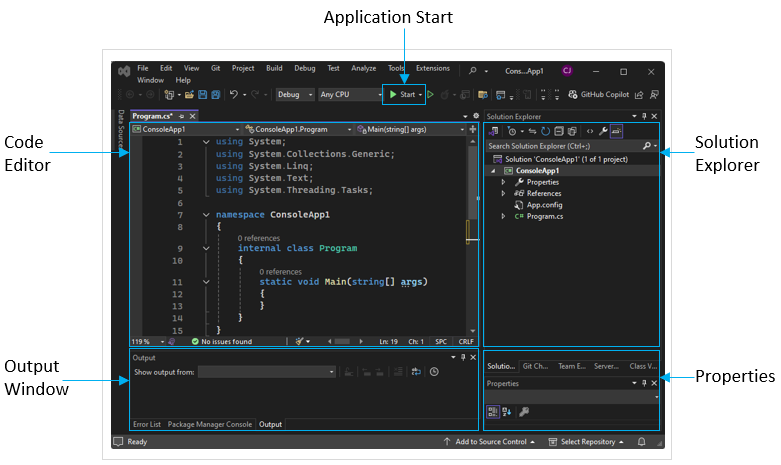
We’ll start with a C# console app to keep things simple and focus on core syntax and logic. Running in the command line, it accepts input and prints output for fast feedback and easy debugging. The fundamentals you practice (variables, control flow, methods, classes, error handling) carry into web, API, and cloud work. Follow the steps below to create your first console app.

|  |
| --- |
| Hands-on Practice:   1. Launch Visual Studio and click on **Create a new project** under the **Get Started** section. |
| 1. In the **Create a new project** dialog, make sure to select the following options:   **C# 🡪 Windows 🡪 Console**, then select **Console App (.Net Framework)**. Then, click **Next.**  A screenshot of a computer  Description automatically generated |

|  |
| --- |
| Hands-on Practice - Continued: |
| 1. In the **Configure your new project** dialog, leave the default project name and location settings as they are, and click **Create**.      1. If everything goes smoothly, Visual Studio, along with the .NET Framework, will generate your console application. |

# Interactive Development Environment (IDE)

Visual Studio is a comprehensive program that facilitates your software development. The diagram below showcases a few of its essential components.



Application Start: Allows you to execute your code and observe its functionality.

Code Editor Window: Displays the contents of files and serves as your primary workspace for writing code.

Output Window: Presents error messages, debugging details, search results, and other relevant information.

Solution Explorer: Enables you to view, navigate, and manage your code files. It assists in organizing your code by grouping files into folders, solutions, and projects.

Properties Window: Provides additional information about selected components within the current project.

|  |
| --- |
| Hands-on Practice:  To execute your code, click the green application start button .  When you do, you will notice the console window appearing briefly and then disappearing. This behavior is expected since you haven't provided any instructions for the application to perform. |

# Console Read and Write

In order to prevent the console window from automatically closing, it is necessary to incorporate a Console.ReadLine()**;** statement within the Main method of your application.

The ReadLine() method is used to read a line of text from the console. It waits for the user to enter a line of text and press the Enter key.

|  |  |  |  |
| --- | --- | --- | --- |
| Hands-on Practice:  Insert the yellow highlighted statement into your application as shown below.  Once added, click the green application start button again .   |  |  |  | | --- | --- | --- | | |  | | --- | | using System;  namespace ConsoleApp1  {  internal class Program  {  static void Main(string[] args)  {  Console.ReadLine();  }  }  } | | This time the Windows Command Line remains visible instead of immediately closing, as it awaits your input. Pressing the **Enter** key will close the console and terminate the application. | |

To display text on the command line, use the WriteLine() method. The WriteLine() method is responsible for writing a line of data to the output window. You need to enclose the desired text within **double quotes** and place it inside the method’s **parentheses**.

|  |  |  |  |
| --- | --- | --- | --- |
| Hands-on Practice:  Insert Console.WriteLine("Hello World")**;** directly above the ReadLine(); statement within the Main method, as it is below. After adding the code, click the **Start** button once again.   |  |  |  | | --- | --- | --- | | using System;  namespace ConsoleApp1  {  internal class Program  {  static void Main()  {  Console.WriteLine("Hello World");  Console.ReadLine();  }  }  } | At this point, you should see the "**Hello World**" text being displayed in the console. To terminate the application, press **Enter** once more.   |  | | --- | | Hello World | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Now you try:   |  |  |  |  | | --- | --- | --- | --- | | Run the following code.   |  | | --- | | using System;  namespace ConsoleApp1  {  internal class Program  {  static void Main()  {  Console.Write("Hello ");  Console.WriteLine("World");  Console.ReadLine();  }  }  } | | Then execute this code.   |  | | --- | | using System;  namespace ConsoleApp1  {  internal class Program  {  static void Main()  {  Console.WriteLine("Hello ");  Console.Write("World");  Console.ReadLine();  }  }  } | |   Now consider the difference between these two statements:  Console.Write() and Console.WriteLine()  Quiz Question 5 |

# Basic Code Structures

|  |  |  |  |
| --- | --- | --- | --- |
| Hands-on Practice:  Run the following code.   |  |  |  | | --- | --- | --- | | using System;  // double slashes are used to  // comment out single lines  /\* slash asterisk followed by  asterisk slash will allow the  comment to span multiple lines \*/  namespace ConsoleApp1  {  internal class Program  {  static void Main()  {  string greeting = "Hello World";  Console.WriteLine(greeting);  Console.ReadLine();  }  }  } | Although there is more code, the result is still the same:   |  | | --- | | Hello World | | |

This example highlights several commonly used C# code structures.

A screen shot of a computer program

Description automatically generated

|  |  |
| --- | --- |
| A pair of white balls with black holes  AI-generated content may be incorrect. | Pay attention to the code being presented in different colours as it conveys important information. |

Using directive:Imports a **namespace** so you can reference its classes and methods.

Comments:Comments are notes placed throughout your code, typically to explain its functionality. They are invaluable when maintaining or enhancing your code in your absence. Comments are not executable and are disregarded by the compiler. You can surround the code with comment tags, such as // for single-line comments or /\* \*/ for multi-line comments.

|  |  |
| --- | --- |
| Visual Studio provides these buttons which c*omment-out*  and *un-comment* sections of your code. | Try it out! |

Namespace:Namespaces help organize classes into logical groups and are often referred to as libraries. By utilizing a using statement to reference a namespace, you can access its classes within your file. A namespace definition commences with the namespace keyword, followed by the name. In the code example above the System namespace is imported into this file, granting access to all classes within that namespace.

Access Modifier:The Access Modifier governs how a class and its components can be utilized by other code in your application. Further details on access modifiers will be covered later in this course.

Class:A class is a collection of related methods and properties. Instances of a class are referred to as objects. A class definition begins with the Access Modifier, followed by the class keyword, class name, and a body enclosed in curly braces. In the code above, the Program class is declared, and the Main() method utilizes the Console class to read from and write to the console.

Variable:Variables are user-defined named references to data. Each variable in C# possesses a specific **type** that determines the kind of values it can store. For instance, a string type can store **"Hello, World!"** while an int type can store the number 9. C# is a **type-safe** language, ensuring that values stored in variables are always of an appropriate type. In the code above, the greeting variable is declared and assigned the value **"Hello, World!"**. The WriteLine() method is then used to write the contents of the greeting variable to the console.

Method:Methods are blocks of code designed to perform a specific task. They allow you to break down complex problems into smaller, manageable code segments, enhancing the code’s reusability and comprehension. A method commences with the Access Modifier, followed by a return type, name, parentheses, and a body enclosed in curly braces. In the code above, the Main() method is declared, serving as the entry point of this console application. The void keyword denotes that this method does not return a value. Within the parentheses, the args parameter represents the input variable for this method. The ReadLine() and WriteLine() methods are members of the Console class and the Console class is in the System namespace.

|  |  |
| --- | --- |
|  | Every C# console program has a Main() method located inside a Program.cs file. When the .NET CLR runs the application, it executes the first statement in Main(), executes each statement in order, and then the process ends. In this example, greeting is declared and printed. Console.ReadLine() waits for input before exit. |

|  |  |
| --- | --- |
| using System;  namespace ConsoleApp1  {  internal class Program  {  static void Main()  {  string greeting = "Hello World";  Console.WriteLine(greeting);  Console.ReadLine();  }  }  } | * The .NET CLR begins execution. * The .NET CLR sequentially executes each statement inside the Main() method. * Then the program terminates. |

Quiz Questions 6, 7, 8 and 9

# Coding Conventions

Maintaining consistency and readability within your code is crucial. By adhering to conventions, you can ensure a higher standard of code quality and understandability. Consider the following conventions to enhance your coding practices.

## Naming

Choose meaningful and descriptive names for your new members to enhance code readability. Well-named items serve as a narrative, making it easier to understand their purpose when revisiting the code later. In general, namespace, class, and variable names are often nouns, while method names typically start with a verb, such as Get, Save, or Write.

## Case

In C#, members can be named using multiple words without spaces. There are three commonly accepted casings in C# standards:

PascalCase: The first character of each word is uppercase, while the rest are lowercase.

Example: **MyMethodName**.

camelCase: The first character is lowercase, and the first letter of each subsequent concatenated word is uppercase. Example: **myVariableName**.

UPPER\_CASE: All letters are uppercase, and words are separated by underscores.

Example: **MY\_CONSTANT\_VALUE**.

This table presents various case usages for different C# identifiers.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | **Identifier** | **Case** | **Example** | | Namespace | PascalCase | System | | Class | PascalCase | CustomerAddress | | Property | PascalCase | MaxTemperature | | Method | PascalCase | GetMaxTemperature | | Local Variable/Parameter | camelCase | accountNumber | | Constant | UPPER\_CASE | FREEZING\_POINT | | Enumeration Type | PascalCase | BrowserType | | Enumeration Value | PascalCase | InternetExplorer | |  |  |  | | | | Following these case conventions, helps maintain consistency and readability throughout your code. |
| Indentation In C#, indentation means consistently aligning opening and closing braces for namespaces, classes, and methods so code blocks are clear.  Good indentation improves readability and shows structure. Inside each brace pair, left-align the code, as shown to the right. | |  | | |

Quiz Questions 10 and 11

# Syntax

The rules that define how code must be written so the compiler understands it.

* Key elements:
  + Semicolons: “;” end statements.
  + Braces: “{ }” define code blocks (namespaces, classes, methods, control flow).
  + Parentheses: “()” group expressions and list parameters.
  + Quotation marks: "" delimit strings.
  + Keywords: (e.g., class, return) must be spelled exactly.
* Case sensitivity: C# is case sensitive, Console ≠ console.
* Why it matters: Breaking these rules causes **build errors**, preventing your

program from running.

## Compilation

Translates C# source into **Intermediate Language** using the .NET compiler and outputs a Windows **console executable (.exe)**.

* Why it matters: Compilation catches syntax/type errors **before execution** and

produces an artifact you can run, test, and deploy.

* Press : Visual Studio **builds** then launches your app.
  + **F5** = Start **with** debugging (build + run under debugger)
  + **Ctrl+F5** = Start **without** debugging
  + **Ctrl+Shift+B** = builds the solution and reports compile-time errors.

|  |
| --- |
| Hands-on Practice:  Use the keyboard shortcut **Ctrl+Shift +B** to compile your code. Check the Visual Studio Output window to verify the success of the compilation process. If it resembles the screenshot below, it is a successful compilation.  Graphical user interface, text  Description automatically generated |

Q. Where do build outputs go?

A. In your project folder. After a build, Visual Studio writes files directly to <project>\bin\Debug\

|  |  |  |  |
| --- | --- | --- | --- |
| Hands-on Practice:  Right click on the Program.cs file and select *Open Containing Folder*.   |  |  | | --- | --- | |  |  |   Here you’ll see the compiled executable which you can run directly from the application folder:  Double click the .exe file to run your application.   |  | | --- | |  | |

|  |  |
| --- | --- |
| A yellow bell with a white background  AI-generated content may be incorrect. | Build with **Ctrl+Shift+B** and confirm the exact output path in the **Output** window.  Graphical user interface, text  Description automatically generated |

Quiz Questions 12, 13 and 14

## Errors

When working with C# code, precision is crucial. Even a single incorrect character can result in an error. For instance, using single quotation marks to surround the phrase 'Hello World' instead of double quotation marks will cause a syntax error.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Hands-on Practice:  Copy the code below into your VS code editor.   |  |  |  | | --- | --- | --- | | |  | | --- | | using Systems;  namespace ConsoleApp1  {  internal class Program  {  static void Main()  {  string greeting = 'Hello ';  string region = "World";  Console.Write(greeting);  Console.WriteLine("World")  Console.ReadLine();  }  }  } | | Notice in Visual Studio's output window, there is a list of syntax errors.  Double-click on the error message, and the IDE will automatically navigate to the corresponding line of code. |   Unfortunately, the .Net error message generated may not always be helpful, in this case indicating "*too many characters in character literal*" instead of providing specific guidance.     |  |  | | --- | --- | |  |  | |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Now you try:  Go ahead and correct any errors in your application to ensure a successful compilation.  Then run your program to check for any potential runtime errors.   |  |  |  |  | | --- | --- | --- | --- | | |  | | --- | | Hello World | | The **"Hello World"** message displayed in the console indicates there are no syntax errors or runtime errors in your application. | | |  | | | Quiz Question 15 | | |

# The Code Editor

The VS code editor offers features to enhance your coding experience. These features include code completion, syntax highlighting, quick info hints, error notifications, and code fix suggestions.

|  |  |  |  |
| --- | --- | --- | --- |
| Hands-on Practice:  To explore these features further, paste the code below into the VS Code Editor.   |  |  |  | | --- | --- | --- | | |  | | --- | | using System;  using System.Linq;  namespace ConsoleApp1  {  internal class Program  {  static void Main()  {  Console.WriteLine("Hello World");  }  }  } | | After the Console.WriteLine(); statement, type in "Console.Re" just like in the image below:    A pair of white balls with black holes  AI-generated content may be incorrect. Notice the helpful information available to you. | |

Take a moment to look over the information presented by the code editor.

A screenshot of a computer

Description automatically generated

|  |  |
| --- | --- |
| A yellow bell with a white background  AI-generated content may be incorrect. | Pay attention to these helpful tips provided by the editor. By using them, you can catch potential errors and gain a deeper understanding of your code. |

Coloured Text: Different colours are used to highlight various elements in your code, such as C# keywords, libraries, and comments. This colour coding helps you visually distinguish different parts of your code.

Helpful Icons:The editor includes icons displayed in the margin area, which provide suggestions and code fixes for common errors or warnings in your code files. These icons help you quickly identify and address potential issues.

Error: A red dot will appear in the margin of the code editor's slider or a red squiggly lune under your code which indicates there is an error within your code.

Coloured Wavy Lines:Red and green wavy lines are shown in the editor to indicate errors and warnings, respectively. These visual indicators draw your attention to problematic areas in your code, allowing you to take appropriate actions to resolve them.

Quick Info:Hovering over elements in your code triggers a quick info dialog that provides useful information about that element. This feature helps you gain insights into the purpose, usage, and documentation of specific code elements.

IntelliSense: IntelliSense, powered by AI, offers intelligent code completion suggestions as you type. Pressing Tab twice accepts the suggested completion, automatically inserting the corresponding statement for you.

Quiz Questions 16 and 17

# Debugging

Computer bugs are defects in your code, and debugging is the process of identifying and fixing those defects. Visual Studio is an excellent tool for troubleshooting errors in your code. In the previous section you saw its effectiveness in detecting syntax errors. It is also great at detecting runtime errors.

A runtime error occurs during the execution of your code. It can result from logical errors or improper data handling. These errors can cause your program to crash or produce incorrect results.

|  |  |  |  |
| --- | --- | --- | --- |
| Hands-on Practice:  Paste the following code into the Code Editor and execute it:   |  |  |  | | --- | --- | --- | | |  |  | | --- | --- | | using System;  namespace ConsoleApp1  {  internal class Program  {  static void Main()  {  int x = 10;  int y = 0;  int result = x / y;  Console.WriteLine("Result: " + result);  }  }  } | This code snippet intentionally introduces a runtime error by attempting to divide an integer by zero. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Hands-on Practice - Continued:   |  |  |  | | --- | --- | --- | | |  | | --- | |  | | Executing the code will produce this error. | |

Quiz Questions 18, 19 and 20

## The Visual Studio Debugger

When executing code in Visual Studio, the debugger is automatically attached and actively monitors the program's execution. This allows you to pause and examine variables, and step through the code line by line. The debugger helps you track the code's execution, set breakpoints to pause the program, and analyze variable values. It enables you to identify and resolve runtime errors effectively, ensuring the code behaves as expected.

## Breakpoints

Breakpoints are used to pause the debugger during the execution of your application.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Hands-on Practice:   |  |  |  | | --- | --- | --- | | Copy this code into Visual Studio’s code editor. | |  | | --- | | using System;  namespace ConsoleApp1  {  internal class Program  {  static void Main()  {  Console.Write("Hello ");  Console.WriteLine("World");  Console.ReadLine();  }  }  } | | | |  |  |  | | --- | --- | --- | | Now, set a breakpoint in the code editor by clicking in the left margin next to the desired line of code.  A red dot will appear.  To remove the breakpoint, click on the red dot again. | |  | | --- | |  | | | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| When your code is executed and encounters a breakpoint, it enters break mode. In break mode, the execution is paused, allowing you to examine the state of your program.  Go ahead and execute your application and execution will stop at line 9, as indicated by the yellow arrow and yellow text.   |  |  |  | | --- | --- | --- | | |  | | --- | |  | | Notice the console output only displays **Hello**. This is because line 8 has been executed, and line 9 has not. | |

|  |  |
| --- | --- |
| A yellow bell with a white background  AI-generated content may be incorrect. | You can step through your code line by line by pressing **F10**. Pressing **F5** will continue running the code until the next breakpoint or until the program completes. |

## Inspect Variables

When a breakpoint is reached you can inspect variables by hovering over the variable to see a tooltip displaying its type and value. You can also inspect variables in the Locals window, where the watch list shows the variable’s type and current value.

A computer screen shot of a program

Description automatically generated

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Hands-on Practice:   |  |  |  | | --- | --- | --- | | Copy this code into Visual Studio’s code editor. | |  | | --- | | using System;  namespace ConsoleApp1  {  internal class Program  {  static void Main()  {  string greeting = "Hello ";  Console.Write(greeting);  Console.WriteLine("World");  Console.ReadLine();  }  }  } | | | |  |  |  |  |  | | --- | --- | --- | --- | --- | | Set a breakpoint on line 10 by clicking in the left margin next to Console.Write(greeting);  Execute your code.  In the Locals window modify the value of greeting to "Brave New ". Don’t forget to hit enter to save the change.  Continue processing to the end.  The output will reflect greeting‘s value change. | |  | | --- | |  |  |  | | --- | |  |  |  | | --- | | Brave New World | | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Now you try:  Copy the following code into the code editor and set a breakpoint on line 9. Then execute it.   |  |  |  |  | | --- | --- | --- | --- | | |  | | --- | | using System;  namespace ConsoleApp1  {  internal class Program  {  static void Main()  {  string greeting = "Hello ";  Console.Write(greeting);  Console.WriteLine("World");  Console.ReadLine();  }  }  } | | |  | | --- | |  | |   Consider why the variable greeting has a value of null. Quiz Questions 21, 22 and 23 |

# Data Types

In C#, data types define the size and type of values that can be stored in variables. **C# is a strongly typed** language. The C# compiler ensures type safety by checking the compatibility of operations in your code.

## Common Data Types:

In .NET, you'll work with various data types and below is a list of the commonly used ones:

**String:**

string: Represents a sequence of characters enclosed in double quotes.

**Character:**

char: Represents a single text character enclosed in single quotes.

**Boolean:**

bool: Represents true or false values.

**Numeric** (without decimal) **:**

byte: Range from 0 to 255.

int: Range from -2,147,483,648 to 2,147,483,647.

long: Range from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.

**Floating Point** (with decimal, also capable of storing non-decimal values) **:**

float: Precision up to 7 decimal places.

double: Precision up to 16 decimal places.

decimal: Precision up to 29 decimal places.

**DateTime:**

DateTime: Stores date and time.

## Declaring Variables

When declaring a variable, you have two options: either specify its type explicitly or use the var keyword to let the compiler infer the type **implicitly**.

|  |  |  |  |
| --- | --- | --- | --- |
| You can declare a variable without assigning a value to it as long as you specify its **type** **explicitly**. | |  | | --- | |  | | string greeting; | |

However, if you declare a variable using the var keyword, you must assign a value to it during initialization; otherwise, the compiler will generate an error. 🡪 **C# is strongly typed!**

|  |  |
| --- | --- |
| A yellow bell with a white background  AI-generated content may be incorrect. | Use meaningful names and an appropriate data type. This helps avoid errors, saves memory, and will make your code more readable. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Invalid syntax   |  | | --- | | var greeting; |   Correct syntax   |  | | --- | | var greeting = 6; | | |  | | --- | | A screenshot of a computer  Description automatically generated with medium confidence | |

**Note:** If you declare a variable with a specific type, such as int, and later try to assign a string value to it, the compiler will raise an error. 🡪 **C# is strongly typed!**

## Initializing Variables

Initializing variables refers to assigning an initial value to a variable after it is declared. This is done using the assignment operator "=".

|  |  |  |
| --- | --- | --- |
| Hands-on Practice:  Copy the code below into Visual Studio’s code editor. There are four variables being declared. The first two variables, establishment and minimumAge, have specific data types (string and int) assigned to them explicitly. The other two variables, hours and duration, use the var keyword, allowing the compiler to automatically determine their types based on the assigned values. In this case, the compiler infers that minimumAge is of type int because it is assigned the value `21`, and duration is of type string based on its assigned value.   |  | | --- | | using System;  namespace ConsoleApp1 {  internal class Program  {  static void Main()  {  string establishment;  int minimumAge = 21;  var hours = 24;  var duration = "day";  establishment = "Caesars Palace Casino";  Console.WriteLine($"{establishment} is open {hours} hours " +  $"a {duration} and has a minimum age requirement of " +  $"{minimumAge} years.");  Console.ReadLine();  }  }  } |   The code above demonstrates the use of string interpolation in C#. It combines variables and text to create a formatted string output using the $ symbol. More about string interpolation later.  Executing the code will produce the following result.   |  | | --- | | Caesars Palace Casino is open 24 hours a day and has a minimum age requirement of 21 years. | |

Quiz Questions 24, 25 and 26

## Constants

Constants are used to represent values that remain unchanged throughout the execution of a program. They are declared using the const modifier and assigned a value that cannot be modified later. Constants are typically spelled using uppercase letters and, if multiple words are used, they are separated by underscores to enhance readability.

|  |  |  |  |
| --- | --- | --- | --- |
| For example, here is a declaration of a constant: | | |  | | --- | | const int MINIMUM\_AGE = 21; | |
| Hands-on Practice:  Replace the contents of the Program.cs file with the code below.   |  | | --- | | using System;  namespace ConsoleApp1 {  internal class Program {  static void Main() {  // Constant int declaration (the value must never change)  const int MINIMUM\_AGE = 21;  // Declare numeric types (has no decimal point)  byte numOfAttemps = 255;  int minimumAge = 21;  long distinctions = 384400000;  // Declare floating point types (has decimal point)  float price = 20.99f;  double interestRate = 2.625;  decimal temperature = 21.5m;  // Declare boolean types  bool enjoyProgramming = true;  // Declare character types  char answer1 = 'C';  // Declare string types  string establishment = "Caesars Palace Casino";  // Declare date/time types  DateTime rightNow = DateTime.Now;  // Declare object types  object car = new {  make = "Mustang",  Colour = "Red"  };  // Write the declared variables to the console  Console.WriteLine($"Attempts: {numOfAttemps}, Minimum age: {minimumAge}," +  $" Distance to the moon: {distanceToMoon} meters");  Console.WriteLine($"Price: ${price}, Interest rate: {interestRate}%," +  $" Temperature: {temperature}c");  Console.WriteLine($"You answered: {answer1})");  Console.WriteLine($"You enjoy programming: {enjoyProgramming}");  Console.WriteLine($"The establishment is : {establishment}");  Console.WriteLine($"The date-time now is : { rightNow }");  Console.WriteLine($"You have a nice car: {car}");  Console.ReadLine();  }  }  } |   The code example above demonstrates the usage of the different data types mentioned earlier. When you execute the code, the output will be displayed in Visual Studio’s Output window.   |  | | --- | | Attempts: 255, Minimum age: 21, Distance to the moon: 384400000 meters  Price: $20.99, Interest rate: 2.625%, Temperature: 21.5c  You answered: C)  You enjoy programming: True  The establishment is : Caesars Palace Casino  The date-time now is : 2024-09-03 12:09:41 PM  You have a nice car: { make = Mustang, Colour = Red } | | | |

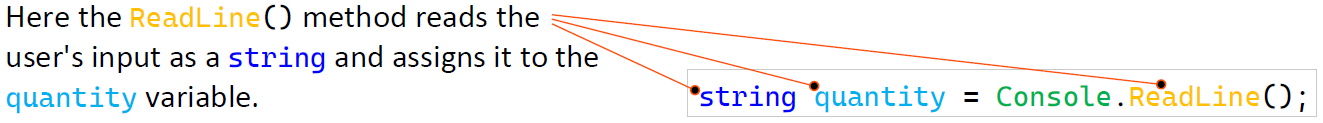
When using the var keyword, a suffix of 'f' or 'm' is required to differentiate float and decimal values from double values. This distinction is necessary for the compiler to determine the appropriate data type. If the suffix is not provided when assigning a decimal value to var, the compiler defaults the type to double. Both examples below result in the same data type assignments:

|  |  |  |  |
| --- | --- | --- | --- |
| |  | | --- | | float price = 20.99f;  double interestRate = 2.625;  decimal temperature = 21.5m; | | |  | | --- | | var price = 20.99f;  var interestRate = 2.625;  var temperature = 21.5m; | |

Quiz Question 27

# More on the ReadLine Method

The Console.ReadLine(); allows you to create interactive console applications. The method is used to read input from the console and store it in a variable. This allows you to use the value entered by the user later in the program.



However, when using Console.ReadLine() the input is always treated as a string. If you want to store the input as an int (integer), you need to convert the string input into an int using the appropriate conversion method. In the example below, the int.Parse() method is used.

A white background with black text

Description automatically generated

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Hands-on Practice:  Execute the following code in your code editor.   |  |  |  | | --- | --- | --- | | |  |  | | --- | --- | | |  | | --- | | using System;  namespace ConsoleApp1  {  class Program  {  static void Main()  {  string input;  Console.Write("How many hotdogs would you like: ");  input = Console.ReadLine();  int quantity = int.Parse(input);  Console.WriteLine($"Wow {quantity} sounds like a lot?");  Console.ReadLine();  }  }  } | | |   Doing so will produce the following output.   |  | | --- | | How many hotdogs would you like: 6  Wow 6 sounds like a lot? | |

|  |  |
| --- | --- |
| Now you try:  Create a new console application that asks for and reads a favorite colour entered in the console, and then writes the colour back to the console.  Have your output look exactly like this.   |  | | --- | | Please enter your favorite colour: Red  I agree, Red is a nice colour!  Now press 'Enter' to terminate this application. |   **Tip:** Use Console.WriteLine(); to print a blank line.  Quiz Question 28 |

# Exception Handling

Exceptions are a type of error that can occur during the execution of an application. They allow for graceful handling and recovery from unexpected situations in your code. When an exception occurs, an exception object is created to store information about the error, such as the type of exception and the location where it occurred.

## Try Catch

A try-catch block is used to handle exceptions that may occur during the execution of code. The code within the try block is monitored for any exceptions, and if an exception occurs, it is caught and processed in the corresponding catch block. This allows for graceful error handling and the ability to take appropriate actions, such as displaying an error message or performing alternative operations, when an exception occurs.

## Flow Charts

Flowcharts are commonly used to visually represent the logical flow and decision-making process in an application. As a software developer, you will frequently encounter flowcharts as a means of understanding and communicating program structure.

The flowchart below illustrates the decision-making and logical flow of a try-catch block:

A picture containing text, screenshot, diagram, font

Description automatically generated

Here's an example demonstrating a try-catch block that prevents a program crash caused by a divide-by-zero exception.

A screenshot of a computer program

Description automatically generated with low confidence

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Hands-on Practice:  Copy the code below into your code editor and place a breakpoint on line 14. Then step through your code using the debugger. Pay attention to the logical flow.   |  | | --- | | using System;  namespace ConsoleApp1 {  class Program {  static void Main() {  int numerator = 10;  int denominator = 0;  int result;  try  {  result = numerator / denominator;  Console.Write($"Result: {result}");  }  catch (Exception ex)  {  Console.Write($"Error: Result: {ex.Message}");  }  Console.ReadLine();  }  }  } |  |  |  |  | | --- | --- | --- | | Your program will return this: | |  | | --- | | Error: Result: Attempted to divide by zero. | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Now you try:  Copy the code below into your code editor and add a try-catch to gracefully handle a non numeric entry from the console. Have the catch block write an error message to the console.   |  | | --- | | using System;  namespace ConsoleApp {  class Program {  static void Main() {  int age;  Console.Write("Enter the legal drinking age: ");  string ageInput = Console.ReadLine();  age = int.Parse(ageInput);  Console.Write("The legal drinking age is: " + age);  Console.ReadLine();  }  }  } |  |  |  |  | | --- | --- | --- | | Your output should look exactly like this when the non-numeric "#" value is entered in the console: | |  | | --- | | Enter the legal drinking age: #  Error: The age you entered was not numeric! | |   Quiz Question 29 |

## Log Data

To record exceptions for later analysis, you can use the StreamWriter class from the System.IO namespace. It allows you to easily output text to a file. The StreamWriter class requires two parameters:

String: The file path indicating the location and name of the file

Boolean: A boolean (true or false) parameter to specify whether the data should be appended to the file, or not.

The code snippet below demonstrates how to create a log file called "log.txt" in the same directory as your project's code files. The StreamWriter class is used to write log data to the file. By specifying true for the APPEND parameter, the data will be appended to the file if it already exists. Otherwise, a new file will be created.

"../" moves up one directory and Two parameters are required,

"log.txt"is the name of the file. the path and true or false.

|  |
| --- |
| string path = "../../log.txt";  StreamWriter streamWriter = new StreamWriter(path, true); |

|  |  |
| --- | --- |
| Hands-on Practice:  Copy the code below into your code editor and execute it.   |  | | --- | | using System.IO;  namespace ConsoleApp1  {  class Program  {  static void Main()  {  const string FILE\_NAME = "log.txt";  const bool APPEND = true;  const string MESSAGE = "Here is some log data.";  const string FILE\_PATH = "../../" + FILE\_NAME;  // Create the writer object  StreamWriter sw = new StreamWriter(FILE\_PATH, APPEND);  // Append to file  sw.WriteLine(MESSAGE);  // Close the writer  sw.Close();  }  }  } | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| To verify the log data, navigate to the directory where your application solution is stored and open the "log.txt" file.  You will see that the WriteLine statement has written the specified message to the log file.  Directory showing the new log file. New file’s contents.   |  |  |  |  | | --- | --- | --- | --- | | |  | | --- | |  | | |  | | --- | |  | | |

|  |  |
| --- | --- |
| A yellow bell with a white background  AI-generated content may be incorrect. | If you want a quicker way to navigate to the folder, you can right-click on your application name in the **Solution Explorer** and select **"Open Folder in File Explorer"**. |

|  |
| --- |
| Now you try:  Create a new console application that writes three dog breeds to a file named DogBreeds.txt. Have the file emptied before you write to it.  Quiz Question 30 |