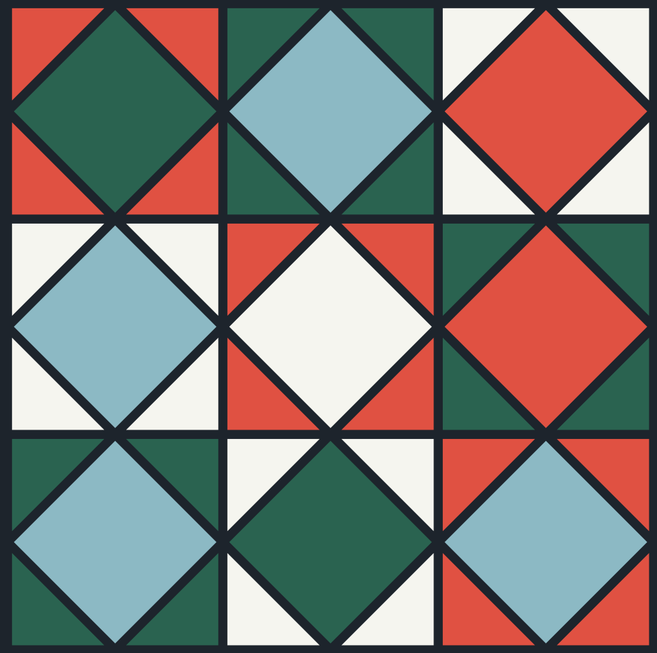
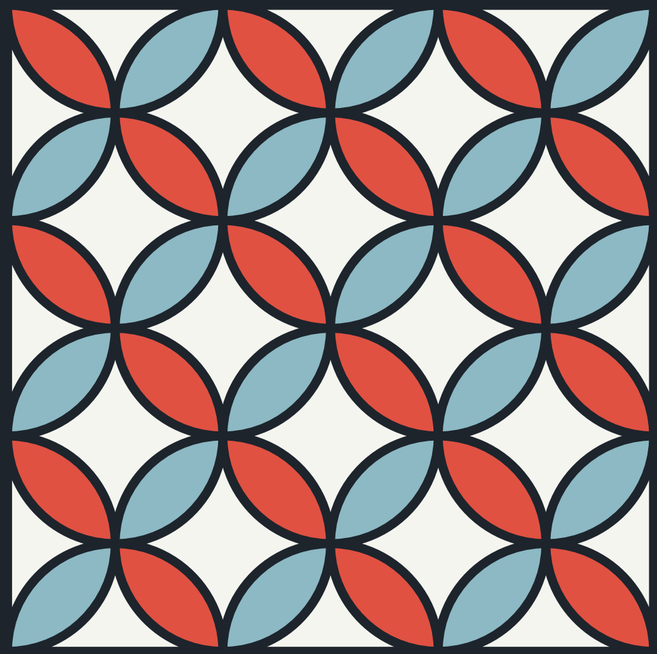
MODULE 1





TODAY'S AGENDA

- WHAT IS C#?
- .NET ARCHITECTURE
- C# IDES

WHAT IS C#?

C# is a strongly typed, object-oriented programming language developed by Microsoft in 2001. It's characterized by its simplicity, modern features, and versatility.

KEY CHARACTERISTICS

- Modern and easy
- Fast and open source
- Cross-platform
- Safe
- Versatile
- Evolving

WHERE IS C# OFTEN USED IN?

C# finds applications in various domains, making it a popular choice among developers.

APPLICATIONS

- Windows client applications
- Web applications and services
- Mobile app development (iOS and Android)
- Cloud applications
- Gaming development
- Artificial Intelligence and Machine Learning
- Internet of Things (IoT) devices

WHY USE C# ?

There are several reasons why C# is preferred by developers and businesses alike.

Reasons to Use C#:

- Simplicity and modern features
- Open-source nature
- Cross-platform compatibility
- Safety and efficiency
- Versatility in application development
- Continuous evolution and updates

C# CHARACTERISTICS

MODERN AND EASY

C# is a simple, modern, and an object-oriented programming language. The purpose of C# was to develop a programming language that is not only easy to learn but also supports modern day functionality for all kind of software development.

FAST AND OPEN SOURCE

C# is open source under the .NET Foundation, which is governed and run independently of Microsoft. C# language specifications, compilers, and related tools are open source projects on Github. While C# language feature design is lead by Microsoft, the open source community is very active in the language development and improvements.

CROSS PLATFORM

You can build .NET applications that can be deployed on Windows, Linux, and Mac platforms. C# apps can also be deployed in cloud and containers.

C# CHARACTERISTICS

SAFE AND EFFICIENT

C# is a type safe language. C# does not allow type conversions that may lead to data loss or other problems. C# allows developers to write safe code. C# also focuses on writing efficient code.

KEY CONCEPTS IN C#

- Unsafe type casting is not allowed.
- Nullable and non-nullable types are supported in C#.
- Declare a readonly struct to express that a type is immutable and enables the compiler to save copies when using in parameters.
- Use a ref readonly return when the return value is a struct larger than IntPtr.Size and the storage lifetime is greater than the method returning the value.
- When the size of a readonly struct is bigger than IntPtr.Size, you should pass it as an in parameter for performance reasons.
- Never pass a struct as an in parameter unless it's declared with the readonly modifier because it may negatively affect performance and could lead to an obscure behavior.
- Use a ref struct, or a readonly ref struct such as Span<T> or ReadOnlySpan<T> to work with memory as a sequence of bytes

C# CHARACTERISTICS

VERSATILE

C# is a Swiss army knife. While most programming languages were designed for a specific purpose, C# was designed to do C#. We can use C# to build today's modern software applications. C# can be used to develop all kind of applications including Windows client apps, components and libraries, services and APIs, Web applications, Mobile apps, cloud applications, and video games.

TYPES OF APPLICATIONS C#

- Windows client applications
- Windows libraries and components
- Windows services • Web applications
- Web services and Web API
- Native iOS and Android mobile apps
- Backend services
- Azure cloud applications and services
- Backend database using ML/Data tools
- Interoperability software such as Office, SharePoint, SQL Server and so on.
- Artificial Intelligence and Machine learning 11
- Blockchains and distributed ledger technology including cryptocurrency
- Internet of Things (IoT) devices
- Gaming consoles and gaming systems
- Video games

C# CHARACTERISTICS

EVOLVING

C# 8.0 is the latest version of C#. If you look at C# language history, C# is evolving faster than any other languages. Thanks to Microsoft and a strong community support. C# was initially designed to write Windows client applications but today, C# can do pretty much anything from console apps, cloud app, and modern machine learning software.

.NET ARCHITECTURE AND C#

COMMON LANGUAGE RUNTIME AND COMMON LANGUAGE INFRASTRUCTURE

CLR is a virtual execution system provided by Microsoft for running C# programs. CLI is an international standard that defines how languages and libraries can work together seamlessly.

INTERMEDIATE LANGUAGE

Intermediate Language (IL) is a low-level programming language that C# code is compiled into. It's not directly executable by the computer but is understood by the CLR.

CONVERSION OF IL TO NATIVE MACHINE INSTRUCTIONS

Just-In-Time (JIT) compilation is a process where IL code is converted into native machine code, which can be executed by the computer's processor.

.NET ARCHITECTURE AND C#

LANGUAGE INTEROPERABILITY

Ability of different programming languages to work together seamlessly within the .NET framework.

.NET EXTENSIVE LIBRARIES

Collections of pre-written code provided by .NET framework, organized into namespaces, offering various functionalities.

C# Versions

1.0 1999- 2002

laid the groundwork for object-oriented development within the .NET Framework

2.0 2005

introduced significant advancements that empowered developers

3.0 2008

streamlined object-oriented development practices

4.0 2010

focused on improving how C# code interacts with other components and the overall developer experience

5.0 2012

embraced asynchronous programming

6.0 2015

focused on developer productivity and clarity.

7.0 2017

Biggest features are tuples for multiple results and pattern matching for simplifying code based on the shape and size of data. Improved data consumption and code simplification .

7.1 2017

Minor version or first point release of C#. Additional features include Async Main for waiting a returned task to complete, Inferred tuple and Default Literals.

C# 1.0 (2002)

The Foundation

```
class Student
{

    0 references
    static void Main(string[] args)
    {

        int roll_number = 100;
        string class_name = "Grade 2";
        string section = "A";
        string subject_1 = "English";
        string subject_2 = "Math";
        int subject_marks_1 = 95;
        int subject_marks_2 = 87;

        int total_marks = subject_marks_1 + subject_marks_2;

    }
}
```

- Established core syntax and features
- Classes, objects, inheritance
- Basic data types, control flow statements, operators

C# 2.0 (2005)

Boosting Code Reusability and Flexibility

```
Unity Script | 1 reference
public class GenericClass<T> : MonoBehaviour
{
    public T someGenericVariable;

    //input parameter and return type are of type T
    0 references
    public T SomeGenericFunction(T genericInput)
    {
        return genericInput;
    }
}

Unity Script | 0 references
public class LessGenericClass : GenericClass<float>
{

}
```

Content:

- Introduced **Generics**:
  - Type-safe, reusable code templates for various data types.
- Enhanced Code Organization:
  - Partial classes for splitting class definitions across files.
- Improved Code Readability and Flexibility:
  - Anonymous methods for defining functions on-the-fly.
- Iterators for creating custom looping constructs.
- (Optional) Nullable Types (introduced but finalized later):
  - Allowed representing the absence of a value with a null type (partially addressed potential null reference exceptions).

C# 3.0 (2007)

Simplifying Object-Oriented Development

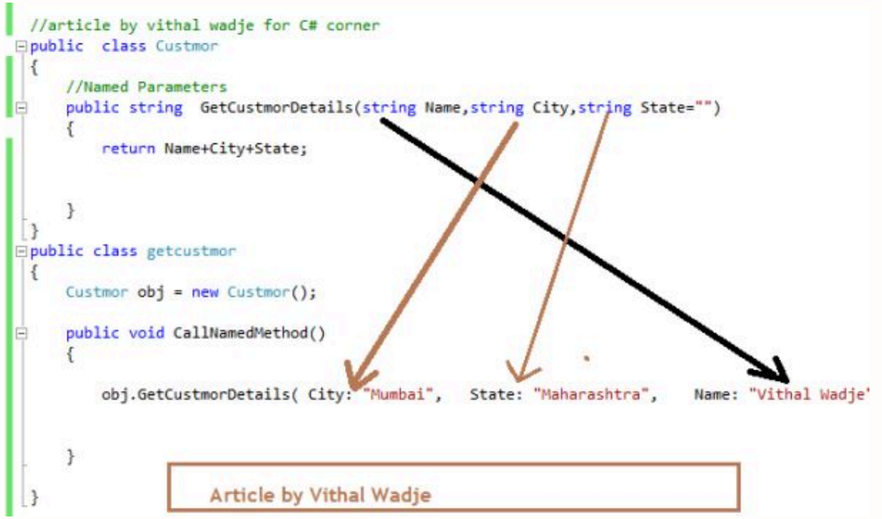
```
public class Main : MonoBehaviour
{
    public Func<int> onGetLength;

    Unity Message | 0 references
    private void Start()
    {
        onGetLength = () => gameObject.name.Length;
    }

    /*int GetName()
    {
        return gameObject.name.Length;
    }*/

}
```

Content:

- Introduced **Lambda Expressions**:
  - Concise syntax for defining anonymous functions, improving code readability.
- Enabled Extension Methods:
  - Added functionalities to existing types without modifying the original definition.
- Reduced Boilerplate Code:
  - Automatic properties for simplified property declaration and backing field creation.
  - Object and collection initializers for a more readable way to initialize objects and collections.
- Implicitly Typed Local Variables
  - Compiler-Inferred types for local variables, reducing code verbosity.

C# 4.0 (2010)

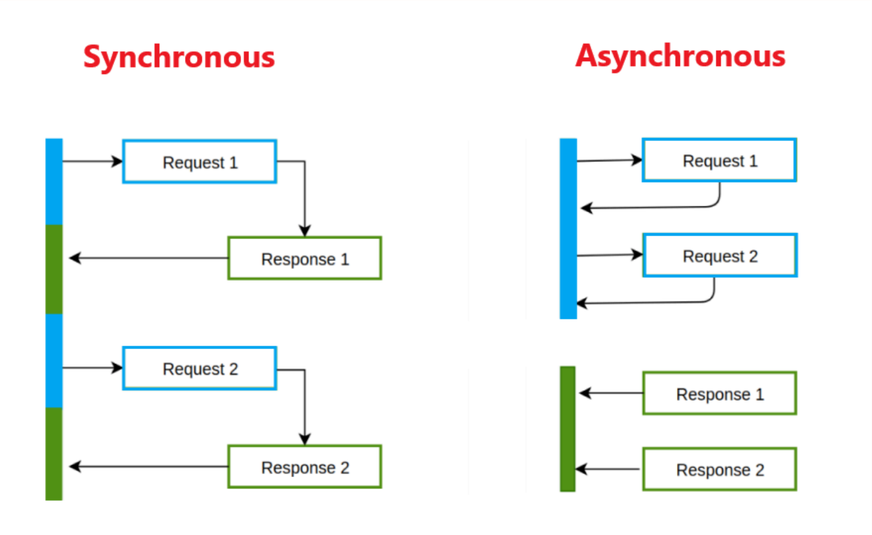
Enhanced Interoperability



- Improved Interoperability:
  - Dynamic binding for flexible method calls at runtime.
  - Enhanced COM interoperability for smoother interaction with COM components.
- Easier Development Experience:
  - **Named arguments** for code readability by specifying parameter names when calling methods.

C# 5.0 (2012)

Embracing Asynchronous Programming



Content:

- Introduced **async** and **await** keywords:
  - Enabled asynchronous programming for handling long-running operations without blocking the UI thread.
  - Improved responsiveness and user experience in applications.

C# 6.0 (2015)

Enhanced Developer Workflow

Content:

- Focused on developer productivity and code clarity:
  - Expression-bodied members: Concise way to define methods and properties with a single expression.
  - Null-Conditional operators: Safe navigation through object properties to avoid NullReferenceException
  - String interpolation: Embedding expressions within strings for easier and more readable string formatting.
  - Read-only auto-properties: Created properties with read-only access by default.
  - Await in Catch and Finally Blocks: Enabled handling exceptions asynchronously within catch and finally blocks.

C# Versions

1.0 1999- 2002

laid the groundwork for object-oriented development within the .NET Framework

2.0 2005

introduced significant advancements that empowered developers

3.0 2008

streamlined object-oriented development practices

4.0 2010

focused on improving how C# code interacts with other components and the overall developer experience

5.0 2012

embraced asynchronous programming

6.0 2015

focused on developer productivity and clarity.

7.0 2017

Biggest features are tuples for multiple results and pattern matching for simplifying code based on the shape and size of data. Improved data consumption and code simplification .

7.1 2017

Minor version or first point release of C#. Additional features include Async Main for waiting a returned task to complete, Inferred tuple and Default Literals.

C# 7.0 2017

Out Variables

Before

```
1  public void PrintCoordinates(Point p)
2  {
3      int x, y; // have to "predeclare"
4      p.GetCoordinates(out x, out y);
5      WriteLine($"({x}, {y})");
6  }
```

In older versions, before you can call a method, you first have to declare variables to pass to it. You cannot also use var to declare them and need to specify the full data type.

After

```
1  public void PrintCoordinates(Point p)
2  {
3      p.GetCoordinates(out int x, out int y);
4      WriteLine($"({x}, {y})");
5  }
```

In 7.0, out variables was added where you can declare a variable right at the point of the argument

C# 7.0 2017

Pattern Matching

In C# 7.0, the notion of “pattern was introduced”. It is like a set of rules, guidelines or syntax that you will follow in order to test that a certain value follows a “shape”. If that value follows that “shape” it will extract information from it.

```
using System;

public class Program {

    // Main function
    public static void Main()
    {
        // variable to be checked
        int sample = 110;

        // if statement that checks if value of sample is of type int and greater than 100
        if (sample is int count && count > 100)
        {
            //output if condition is true
            Console.WriteLine("The value of sample, " + sample + ", is of type int and grater than 100");
        }
        else
        {
            //output if condition is false
            Console.WriteLine("The value of the variable sample is not of our desired type and/or value");
        }

    }
}
```

C# 7.0 2017

Tuples

Used for storing fixed and different but related values in a single data structure

```
(string country, string capital, double gdpPerCapita) =
  ("Malawi", "Lilongwe", 226.50);
```

Let’s say we want to assign this tuple to individually declared variables:

```
(string country, string capital, double gdpPerCapita) =
("Malawi", "Lilongwe", 226.50);
System.Console.WriteLine(
$@"The poorest country in the world in 2017 was {
country}, {capital}: {gdpPerCapita}");
```

C# 7.0 2017

Local Functions

Before, you can only define a helper function in a separate method. But with the added feature of local functions, you can now declare functions inside another function.

```
// C# program to illustrate local function
using System;

public class Program {

    // Main method
    public static void Main()
    {
        // Local Function
        void AddValue(int a, int b)
        {
            Console.WriteLine("Value of a is: " + a);
            Console.WriteLine("Value of b is: " + b);
            Console.WriteLine("Sum of a and b is: {0}", a + b);
            Console.WriteLine();
        }

        // Calling Local function
        AddValue(20, 40);
        AddValue(40, 60);
    }
}
```

C# Versions

1.0 1999- 2002

laid the groundwork for object-oriented development within the .NET Framework

2.0 2005

introduced significant advancements that empowered developers

3.0 2008

streamlined object-oriented development practices

4.0 2010

focused on improving how C# code interacts with other components and the overall developer experience

5.0 2012

embraced asynchronous programming

6.0 2015

focused on developer productivity and clarity.

7.0 2017

Biggest features are tuples for multiple results and pattern matching for simplifying code based on the shape and size of data. Improved data consumption and code simplification .

7.1 2017

Minor version or first point release of C#. Additional features include Async Main for waiting a returned task to complete, Inferred tuple and Default Literals.

C# 7.1 2017

Async Main

Instead of solely relying on the traditional Main method as the starting point of program execution, we now have the option to use async Main for handling asynchronous tasks. This allows us to perform operations that don't block the main thread

```
using System;
using System.Threading.Tasks;

namespace AsyncMainDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Before C# 7.1, To use async method");
            Console.WriteLine($"Main Method execution started at
{System.DateTime.Now}");

            //Calling Async Method
            //We cannot use await as the Main method is not async
            //Hence using Wait Method to wait for the completion of the
SomeAsyncMethod()
            SomeAsyncMethod().Wait();

            Console.WriteLine($"Main Method execution ended at {System.DateTime.Now}");

            Console.WriteLine("Press any key to exist.");
            Console.ReadKey();
        }

        //Async Method
        static async Task SomeAsyncMethod()
        {
            await Task.Delay(2000);
        }
    }
}
```

C# 7.1 2017

Default Literals

In this version, it introduced an easier way to obtain a default value of any type.

Before

```
int intValue = default(int);
double doubleValue = default(double);
bool boolValue = default(bool);
string str = default(string);
int? nullableInt = default(int?);

Action<int, bool> action = default(Action<int, bool>);
Predicate<string> predicate = default(Predicate<string>);
List<string> list = default(List<string>);
Dictionary<int, string> dictionary = default(Dictionary<int, string>);
```

After

```
double doubleValue = default;
bool boolValue = default;
string str = default;
int? nullableInt = default;

Action<int, bool> action = default;
Predicate<string> predicate = default;
List<string> list = default;
Dictionary<int, string> dictionary = default;
```

C# 7.1 2017

Inferred Tuple Element Names

Instead of specifying the types of each element in the tuple, inferred tuples allows omission of tuple elements when declared

Before

```
public static void Demo()
{
    var count = 5;
    var type = "Orange";

    var tuple = (Count: count, Type: type);
}
```

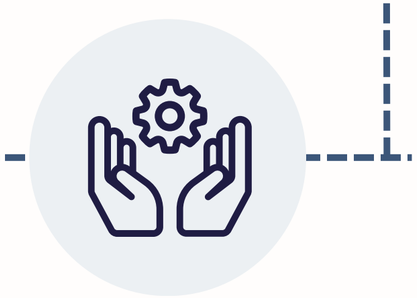
After

```
public static void Demo()
{
    var count = 5;
    var type = "Orange";

    var tuple = (count, type);
}
```

C# Versions

7.2 2017

Centered on structs by reference. Focused on high performance scenario to avoid wasted collection of allocation and excessive copying. Features include - Reference semantics with value types, Non-trailing named arguments, Leading underscores in numeric literals, private protected access modifier
,



8.0 2019

The first major C# release. me features rely on new Common Language Runtime (CLR) capabilities, others on library types added only in .NET Core. Features include nullable reference types, ranges and indices, recursive patterns, switch expressions and many more.

7.3 2018

Improved C# 7.2 on features such as ref variables, pointers and stackalloc. Removed long time restrictions on constraints.

C# 7.2 2017

Reference Semantics with Value Types

- `in` Specifies that you want to pass a struct by reference, but the caller cannot modify it
- `ref readonly` returns a struct by reference as an object that may not be modified
- `readonly struct` creates a struct that can never be modified once created
- `ref struct` deserves an article of its own, but basically means that it can never be allocated on the managed heap. The main motivation is for the awesome new `Span<T>` class which also deserves its own article.

C# 7.2 2017

Non-trailing Named Arguments

Before C# 7.2, named arguments can only be used after positional arguments. Here, you can now specify named arguments in any position within the argument,

```
public void Log(bool verbose, string message, object arg)
{
    // Method implementation goes here
}

// Call with positional arguments
Log(true, "Foo", 42);

// Call with named arguments
Log(verbose: true, message: "Foo", arg: 42);
```

C# 7.2 2017

Private Protected Access Modifier

Allows a member of a class to be accessible within the containing class and to its derived classes

```
internal sealed class Hidden
{

}

public abstract class Visible
{
    protected Hidden HiddenObj { get; private set; }
}
```

C# Versions

7.2 2017

Centered on structs by reference. Focused on high performance scenario to avoid wasted collection of allocation and excessive copying. Features include - Reference semantics with value types, Non-trailing named arguments, Leading underscores in numeric literals, private protected access modifier
,

7.3 2018

Improved C# 7.2on features such as ref variables, pointers and stackalloc. Removed long time restrictions on constraints.

8.0 2019

The first major C# release. me features rely on new Common Language Runtime (CLR) capabilities, others on library types added only in .NET Core. Features include nullable reference types, ranges and indices, recursive patterns, switch expressions and many more.

C# 7.3 2018

Ref local Variables

Allows you to create a reference to an exisitng variable . It is useful when you want a to pass a reference to a method or store it for future use

```
void DoStuff(ref int parameter)
{
    // Now otherRef is also a reference, modifications will
    // propagate back
    var otherRef = ref parameter;

    // This is just its value, modifying it has no effect on
    // the original
    var otherVal = parameter;
}
```

C# 7.3 2018

Stackalloc Initializers

Adds the ability to initialize a stack allocated array. In this version, you don't have to use heap allocation and pointers.

Before

```
Span<int> x = stackalloc int[3] { 1, 2, 3 };
```

Initializing a stack-allocated array required specifying the number of elements

After

```
Span<int> x = stackalloc[] { 1, 2, 3 };
```

You can omit the size specification

C# Versions

7.2 2017

Centered on structs by reference. Focused on high performance scenario to avoid wasted collection of allocation and excessive copying. Features include - Reference semantics with value types, Non-trailing named arguments, Leading underscores in numeric literals, private protected access modifier
,

7.3 2018

Improved C# 7.2on features such as ref variables, pointers and stackalloc. Removed long time restrictions on constraints.

8.0 2019

The first major C# release. Features rely on new Common Language Runtime (CLR) capabilities, others on library types added only in .NET Core. Features include nullable reference types, ranges and indices, recursive patterns, switch expressions and many more.

C# 8.0 2019

Nullable Reference Types

Displays a compiler warning when a variable ‘s value is set to null but must not be assigned to nulll .

```
string? nullableString = null;
Console.WriteLine(nullableString.Length); // WARNING: may be null!
Take care!
```

C# 8.0 2019

Ranges and Indices

It is added to provide a more concise syntax in accessing sequences of data (arrays, list, strings, etc.).

```
Index i1 = 3; // number 3 from beginning
Index i2 = ^4; // number 4 from end

int[] a = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
Console.WriteLine($"{a[i1]}, {a[i2]}"); // "3, 6"
```

C# 8.0 2019

Default Interface Methods

Already applied and is cloned in Java. It is used for adding new methods to existing interfaces.

```
interface IWriteLine
{
    public void WriteLine()
    {
      Console.WriteLine("Wow C# 8!");
    }
}
```

C# IDES

MONODEVELOP

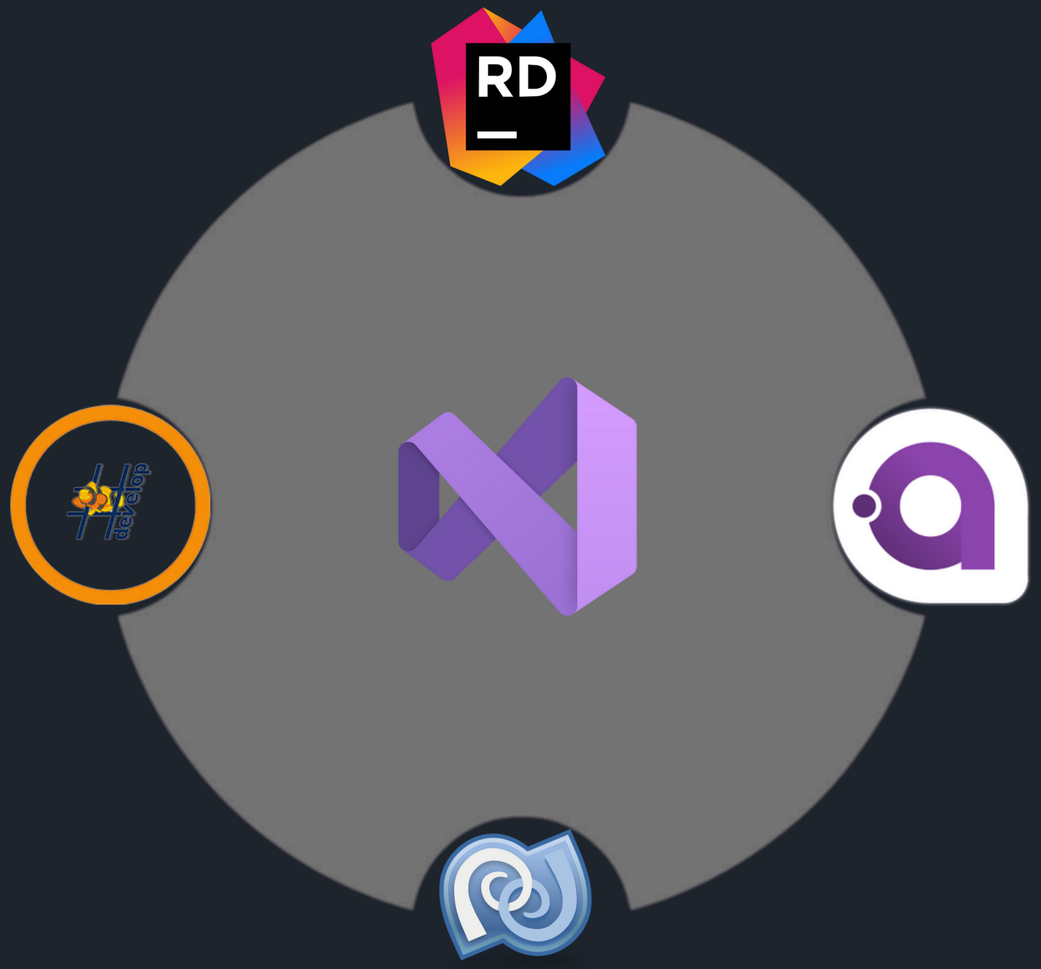
JETBRAINS RIDER

SHARPDEVELOP

VISUAL STUDIO

AVALONIA STUDIO

C# IDES



Visual Studio: A comprehensive IDE developed by Microsoft.

SharpDevelop: An open-source IDE for C# development.

MonoDevelop: A cross-platform IDE supporting C# development.

Avalonia Studio: An IDE tailored for developing desktop applications using the Avalonia UI framework.

JetBrains Rider: A powerful cross-platform IDE developed by JetBrains for .NET and C# development.

C# IDES

VISUAL STUDIO

Virtual Studio is the easiest way to use as an IDE, This software allows you to write, edit, and compile your code in one place.

C# IDES

SHARPDEVELOP

SharpDevelop is an open-source IDE for C# development, offering features like code editing and debugging.

C# IDES

MONODEVELOP

MonoDevelop, is a cross-platform IDE for C# development. It supports code editing, debugging, and project management.

C# IDES

AVALONIA STUDIO

Avalonia Studio is an IDE tailored for developing desktop applications using the Avalonia UI framework.

C# IDES

JETBRAINS RIDER

JetBrains Rider is a robust cross-platform IDE tailored for .NET and C# development.

C# IDES

ADD INSTRUCTIONS OR GUIDELINES HERE.
YOU CAN ALSO PUT IN THE AMOUNT OF TIME
ALLOTTED FOR THIS.

QUESTION

Write the question you want to ask your students and allot space for the answers.

QUESTION

Write the question you want to ask your students and allot space for the answers.

THANK YOU!