Language Map for C#

Variable Declaration *Is this language strongly typed or dynamically typed? Provide at least three examples (with different data types or keywords) of how variables are declared in this language.*	C# is strongly typed, meaning variable types are known at compile time. Examples: Explicitly: int age = 31; string name = “Bo”; Implicitly: var age = 31; var name = “Bo”; Constant: const int maxAge = 100; const string exit = “Goodbye.”;
Data Types *List all of the data types (and ranges) supported by this language.*	**Integral Types** 1. **sbyte**: Signed 8-bit integer ○ Range: -128 to 127 2. **byte**: Unsigned 8-bit integer ○ Range: 0 to 255 3. **short**: Signed 16-bit integer ○ Range: -32,768 to 32,767 4. **ushort**: Unsigned 16-bit integer ○ Range: 0 to 65,535 5. **int**: Signed 32-bit integer ○ Range: -2,147,483,648 to 2,147,483,647 6. **uint**: Unsigned 32-bit integer ○ Range: 0 to 4,294,967,295 7. **long**: Signed 64-bit integer ○ Range: -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 8. **ulong**: Unsigned 64-bit integer ○ Range: 0 to 18,446,744,073,709,551,615 9. **nint**: Signed native-sized integer (32-bit or 64-bit) 10. **nuint**: Unsigned native-sized integer (32-bit or 64-bit) **Floating-Point Types** 1. **float**: Single-precision floating-point ○ Range: ±1.5 x 10^−45 to ±3.4 x 10^38 2. **double**: Double-precision floating-point ○ Range: ±5.0 x 10^−324 to ±1.7 x 10^308 3. **decimal**: High-precision decimal ○ Range: ±1.0 x 10^−28 to ±7.9 x 10^28 **Other Types** 1. **char**: Single 16-bit Unicode character ○ Range: U+0000 to U+FFFF 2. **bool**: Boolean value ○ Values: true or false

3. **object**: Base type of all other types
4. **string**: Sequence of characters
5. **DateTime**: Represents date and time

Selection Structures

Provide examples of all selection structures supported by this language (if, if else, etc.) ***Don’t just list them, show code samples of how each would look in a real program.***

if

```
If (number < 7)
{
 Console.WriteLine(" number greater than 5.");
}
```

if else

```
if (number > 5)
{
    Console.WriteLine("number greater than 5.");
}
else
{
    Console.WriteLine("The number is 5 or less.");
}
```

If – else if - else

```
if (number > 10)
{
    Console.WriteLine("number greater than 10.");
}
else if (number > 5)
{
    Console.WriteLine("number greater than 5 but 10 or less.");
}
else
{
    Console.WriteLine("The number is 5 or less.");
}
```

switch

```
int day = 3;

switch (day)
{
    case 1:
        Console.WriteLine("Monday");
        break;
    case 2:
```

		Console.WriteLine("Tuesday");
		break;
		case 3:
		Console.WriteLine("Wednesday");
		break;
		case 4:
		Console.WriteLine("Thursday");
		break;
		case 5:
		Console.WriteLine("Friday");
		break;
		case 6:
		Console.WriteLine("Saturday");
		break;
		case 7:
		Console.WriteLine("Sunday");
		break;
		default:
		Console.WriteLine("Invalid day");
		break;
		}

	Ternary Operator	int number = 8;
		string result = (number > 5) ? "Greater than 5" : "5 or less";
		Console.WriteLine(result);

Repetition Structures
Provide examples of all repetition structures supported by this language (loops, etc.) ***Don't just list them, show code samples of how each would look in a real program.***
	For loop	for (int i = 0; i < 5; i++)
		{
		Console.WriteLine("Iteration: " + i);
		}

	Foreach loop	string[] fruits = { "Apple", "Banana", "Cherry" };
		
		foreach (string fruit in fruits)
		{
		Console.WriteLine(fruit);
		}

	While loop	while (count < 5)
		{
		    Console.WriteLine("Count is: " + count);
		    count++;
		}

	Do-while loop	do
		{
		    Console.WriteLine("Count is: " + count);
		    count++;
		} while (count < 5);

	Break statement	for (int i = 0; i < 10; i++)
		{
		    if (i == 5)
		    {
		        break; // Exit the loop when i is 5
		    }
		    Console.WriteLine("Iteration: " + i);
		}

	Continue statement	for (int i = 0; i < 10; i++)
		{
		    if (i % 2 == 0)
		    {
		        continue; // Skip the rest of the loop iteration if i is even
		    }
		    Console.WriteLine("Odd number: " + i);
		}

Arrays
*If this language supports arrays, provide **at least two examples** of creating an array with a primitive or String data types (e.g. float, int, String, etc.) If the language supports declaring arrays in multiple ways, provide an example of way.*
	Integer Array	//method 1: Array initializer
		int[] numbers = { 1, 2, 3, 4, 5 };
		
		// Method 2: Using the new keyword
		int[] moreNumbers = new int[5];
		moreNumbers[0] = 1;
		moreNumbers[1] = 2;
		moreNumbers[2] = 3;
		moreNumbers[3] = 4;
		moreNumbers[4] = 5;

	String Array	// Method 1: array initializer
		string[] fruits = { "Apple", "Banana", "Cherry" };
		
		// Method 2: Using the new keyword
		string[] moreFruits = new string[3];
		moreFruits[0] = "Apple";
		moreFruits[1] = "Banana";
		moreFruits[2] = "Cherry";

	Float Array	// Method 1: Using array initializer
		float[] temperatures = { 98.6f, 99.5f, 100.1f };
		
		// Method 2: Using the new keyword
		float[] moreTemperatures = new float[3];
		moreTemperatures[0] = 98.6f;
		moreTemperatures[1] = 99.5f;
		moreTemperatures[2] = 100.1f;

Data Structures *If this language provides a standard set of data structures, provide a list of the data structures and their Big-Oh complexity (identify what the complexity represents).*	Array	• **Access**: O(1) • **Search**: O(n) • **Insertion**: O(n) • **Deletion**: O(n)
	List<T>	• **Access**: O(1) • **Search**: O(n) • **Insertion**: O(n) (amortized O(1) for adding at the end) • **Deletion**: O(n)
	LinkedList<T>	• **Access**: O(n) • **Search**: O(n) • **Insertion**: O(1) • **Deletion**: O(1)
	Stack<T>	• **Access**: O(n) • **Search**: O(n) • **Insertion**: O(1) • **Deletion**: O(1)

Queue<T>	• **Access**: O(n) • **Search**: O(n) • **Insertion**: O(1) • **Deletion**: O(1)
Dictionary<TKey, TValue> (Hash Table)	• **Access**: O(1) • **Search**: O(1) • **Insertion**: O(1) • **Deletion**: O(1)
SortedList<TKey, TValue>	• **Access**: O(log n) • **Search**: O(log n) • **Insertion**: O(n) • **Deletion**: O(n)
SortedDictionary<TKey, TValue> (Binary Search Tree)	• **Access**: O(log n) • **Search**: O(log n) • **Insertion**: O(log n) • **Deletion**: O(log n)
HashSet<T>	• **Access**: O(1) • **Search**: O(1) • **Insertion**: O(1) • **Deletion**: O(1)
SortedSet<T> (Binary Search Tree)	• **Access**: O(log n) • **Search**: O(log n) • **Insertion**: O(log n) • **Deletion**: O(log n)

Objects
If this language support object-orientation, provide an example of how you would write a simple object with a default constructor and then how you would instantiate it.
	Yes, C# supports object-orientation.
	<u>Defining Simple class:</u>
	`public class Person`
	`{`
	`    // Fields`
	`    public string Name;`

```
    public int Age;

    // Default constructor
    public Person()
    {
        Name = "Unknown";
        Age = 0;
    }

    // Method to display person details
    public void DisplayInfo()
    {
        Console.WriteLine($"Name: {Name}, Age: {Age}");
    }
}
```

Instantiating the class

```
class Program
{
    static void Main(string[] args)
    {
        // Creating an instance of the Person class using the default constructor
        Person person = new Person();

        // Displaying the default values
        person.DisplayInfo();
    }
}
```

Runtime Environment *What runtime environment does this language compile to? For example, Java compiles to the Java Virtual Machine.* *Do other languages also compile to this runtime? If so, what these other languages?*	C# compiles to the Common Language Runtime (CLR), which is part of the .NET framework. The CLR provides a managed execution environment for .NET applications, handling tasks such as memory management, security, and exception handling. Other lagnuges that use Common Language Runtime(CLR): 1. **Visual Basic .NET (VB.NET)** 2. **F#** 3. **C++/CLI** 4. **IronPython** (a .NET implementation of Python) 5. **IronRuby** (a .NET implementation of Ruby)

	6. **PowerShell** 7. **JScript .NET** 8. **Eiffel** 9. **COBOL** (via third-party compilers) 10. **Perl** (via third-party compilers)	
Libraries/Frameworks *What are the popular libraries or frameworks used by programmers for this language? List at least three (3) and describe what they are used for.*	ASP.NET Core	ASP.NET Core is a cross-platform, high-performance framework for building modern, cloud-based, and internet-connected applications. It allows developers to create web applications, APIs, and microservices. ASP.NET Core is known for its speed, modularity, and flexibility, making it a popular choice for web development.
	Entity Framework Core	Entity Framework Core (EF Core) is an object-relational mapper (ORM) that simplifies data access by allowing developers to work with a database using .NET objects. It eliminates the need for most of the data-access code that developers usually need to write. EF Core supports LINQ queries, change tracking, updates, and schema migrations.
	Xamarin	Xamarin is a framework for building cross-platform mobile applications using C#. It allows developers to write shared code that runs on iOS, Android, and Windows devices. Xamarin provides a single language (C#), a class library, and a runtime that works across all three mobile platforms, enabling code sharing and reducing development time.
Domains *What industries or domains use this programming language? Provide at least three specific examples of companies that use this language and what they use it for**. E.g. Company X uses C# for its line of business applications.***	Microsoft	Technology Usage: Microsoft, the creator of C#, extensively uses it for a wide range of applications, including web development, desktop applications, and game development. For instance, C# is used in developing applications for the Azure cloud platform, as well as in game development for Xbox.
	Stack Overflow	Technology/Internet Usage: Stack Overflow uses C# for backend development and web services. The platform relies on C# to handle millions of queries and interactions from developers around the world, ensuring robust and scalable performance.
	Accenture	Consulting and Professional Services Usage: Accenture utilizes C# to develop agile and flexible applications for their clients. This includes building enterprise-level applications and cloud-based solutions that require high performance and reliability.