A tour of the C# language

C# (pronounced "See Sharp") is a modern, object-oriented, and type-safe programming language. C# enables developers to build many types of secure and robust applications that run in .NET. C# has its roots in the C family of languages and will be immediately familiar to C, C++, Java, and JavaScript programmers. This tour provides an overview of the major components of the language in C# 11 and earlier. If you want to explore the language through interactive examples, try the introduction to C# tutorials.

Several C# features help create robust and durable applications. Garbage collection automatically reclaims memory occupied by unreachable unused objects. Nullable types guard against variables that don't refer to allocated objects. Exception handling provides a structured and extensible approach to error detection and recovery. Lambda expressions support functional programming techniques. Language Integrated Query (LINQ) syntax creates a common pattern for working with data from any source. Language support for asynchronous operations provides syntax for building distributed systems. C# has a unified type system. All C# types, including primitive types such as `int` and `double`, inherit from a single root `object` type. All types share a set of common operations. Values of any type can be stored, transported, and operated upon in a consistent manner. Furthermore, C# supports both user-defined reference types and value types. C# allows dynamic allocation of objects and in-line storage of lightweight structures. C# supports generic methods and types, which provide increased type safety and performance. C# provides iterators, which enable implementers of collection classes to define custom behaviors for client code.

Hello world

The "Hello, World" program is traditionally used to introduce a programming language. Here it is in C#:

```
using System;
class Hello
{
   static void Main()
   {
      Console.WriteLine("Hello, World");
   }
}
```

The "Hello, World" program starts with a `using` directive that references the `System` namespace. Namespaces provide a hierarchical means of organizing C# programs and libraries.

A `using` directive that references a given namespace enables unqualified use of the types that are members of that namespace. Because of the `using` directive, the program can use `Console.WriteLine` as shorthand for `System.Console.WriteLine`.

The `Hello` class declared by the "Hello, World" program has a single member, the method named `Main`. The `Main` method is declared with the static modifier. By convention, a static method named `Main` serves as the entry point of a C# program.

The output of the program is produced by the `WriteLine` method of the `Console` class in the `System` namespace. This class is provided by the standard class libraries, which, by default, are automatically referenced by the compiler.

Types and variables

A *type* defines the structure and behavior of any data in C#. The declaration of a type may include its members, base type, interfaces it implements, and operations permitted for that type. A *variable* is a label that refers to an instance of a specific type.
There are two kinds of types in C#:

1. Value types
2. Reference types.

Value types

C#'s value types are further divided into *simple types*, *enum types*, *struct types*, *nullable value types*, and *tuple value types*.

Value types	**Details**
Simple types	Signed integral, unsigned integral, unicode characters, IEEE binary floating-point, High-precision decimal floating-point, and boolean.
Enum types	User-defined types of the form enum E {...}. An enum type is a distinct type with named constants. Every enum type has an underlying type, which must be one of the eight integral types. The set of values of an enum type is the same as the set of values of the underlying type.
Struct types	User-defined types of the form struct S {...}
Nullable value types	Extensions of all other value types with a null value
Tuple value types	User-defined types of the form (T1, T2, ...)

Reference types

Reference types	**Details**
Class types	Ultimate base class of all other types: object. Unicode strings: string, which represents a sequence of UTF-16 code units. User-defined types of the form class C {...}
Interface types	User-defined types of the form interface I {...}
Array types	Single-dimensional, multi-dimensional, and jagged. For example: int[], int[,], and int[][]
Delegate types	User-defined types of the form delegate int D(...)