- Strongly Typed

**C# is the strongly typed language. It means you can assign a value of the specified data type. You cannot assign an integer value to string type or vice-versa.**

Example: Cannot assign string to int type variable

**int** num = "Steve";

*-*

*(It means that 1 and "1" (or any other number and string for that matter) are different values that cannot be casually interchanged in expressions. // This means, if there are two variables of different types, you have to cast them, to make an operation executable. Else it will throw an exception.)*

**The CLR provides additional services including**[**memory management**](https://www.wikiwand.com/en/Memory_management)**,**[**type safety**](https://www.wikiwand.com/en/Type_safety)**,**[**exception handling**](https://www.wikiwand.com/en/Exception_handling)**,**[**garbage collection**](https://www.wikiwand.com/en/Garbage_collection_(computer_science))**, security and**[**thread management**](https://www.wikiwand.com/en/Thread_management)**.**

-

-

-

Bunun NET bir cevabı yokmuş!

çok önemli devamı aşağıda: Dynamic typing in C# 4 allows **weakly** typed **but still type-safe** code, as shown above.

-

-

-

https://stackoverflow.com/questions/969088/do-type-safe-and-strongly-typed-mean-the-same-thing

Do "type-safe" and "strongly typed" mean the same thing?

No, not necessarily - although it depends on your definition of the terms, and **there are no very clear and widely accepted definitions.**

**For instance, dynamic programming languages are often type safe, but not strongly typed. In other words, there's no compile-time type information determining what you can and can't do with a type, but at execution time the runtime makes sure you don't use one type as if it were another.**

For example, in C# 4.0, you can do:

dynamic foo = "hello";

dynamic length = foo.Length; // Uses String.Length at execution time

foo = new int[] { 10, 20, 30 };

length = foo.Length; // Uses Array.Length at execution time

dynamic bar = (FileStream) foo; // Fails!

The last line is the key to it being type-safe: there's no safe conversion from an int array to a FileStream, so the operation fails - instead of treating the bytes of the array object *as if* they were a FileStream.

EDIT: C# is normally both "strongly typed" (as a language) and type safe: the compiler won't let you attempt to make arbitrary calls on an object, and the runtime won't let you perform inappropriate conversions.

I'm not entirely sure where unsafe code fits in - I don't know enough about it to comment, I'm afraid.

Dynamic typing in C# 4 allows weakly typed **but still type-safe** code, as shown above.

Note that foreach performs an implicit conversion, making it a sort of hybrid:

ArrayList list = new ArrayList();

list.Add("foo");

foreach (FileStream stream in list)

{

...

}

This will compile (there was [another question](https://stackoverflow.com/questions/949798) on this recently) but will fail at execution time. Ironically, that's because you're *trying* to be strongly typed with respect to the stream variable, which means you have to perform a cast on the result of the iterator.

-

-

-

Neither "strongly typed" nor "type safety" have very precise definitions!

In C# it's common to talk about "strongly typed collections" which means a collection type that doesn't store elements as Object but as specific type (int, string etc.). This tipically means generic collections (like in List<int>) or non generic collections that are "specialized" for a particular type (like StringCollection). This implies that the collection allows only objects of a specific type (or derived from that type) and this minimizes errors, for example if you have a List<int> you can be sure that all elements are int but with ArrayList you can't be sure about that, someone may have added a string by mistake and when you try to cast it to int you'll get an exception.

It's also common to refer to the C# language (and runtime) as being "type safe". This basically means that the language (and runtime) will not allow a number of operations that are erroneus due to type mistmatches. For example you cannot cast a string to a int, depending on how you do it you will get a compiler error at compile time or a InvalidCastException at runtime.