**Advantages of .NET**

* Support for optional parameters - very handy for some COM interoperability.
* Support for late binding with Option Strict off - type safety at compile time goes out of the window, but legacy libraries which don't have strongly typed interfaces become easier to use.
* Support for named indexers.
* Various legacy VB functions (provided in the Microsoft.VisualBasic namespace, and can be used by other languages with a reference to the Microsoft.VisualBasic.dll). Many of these can be harmful to performance if used unwisely, however, and many people believe they should be avoided for the most part.
* The with construct: it's a matter of debate as to whether this is an advantage or not, but it's certainly a difference.
* Simpler (in expression - perhaps more complicated in understanding) event handling, where a method can declare that it handles an event, rather than the handler having to be set up in code.
* The ability to implement interfaces with methods of different names. (Arguably this makes it harder to find the implementation of an interface, however.)
* Catch ... When ... clauses, which allow exceptions to be filtered based on runtime expressions rather than just by type.
* The VB.NET parts of Visual Studio .NET compiles your code in the background. While this is considered as an advantage for small projects, people creating very large projects have found that the IDE slows down considerably as the project gets larger.

**Advantages of C#**

* XML documentation generated from source code comments. (This is coming in VB.NET with Whidbey (the code name for the next version of Visual Studio and .NET), and there are tools which will do it with existing VB.NET code already.)
* Operator overloading - again, coming to VB.NET in Whidbey.
* Language support for unsigned types (you can use them from VB.NET, but they aren't in the language itself). Again, support for these is coming to VB.NET in Whidbey.
* The using statement, which makes unmanaged resource disposal simple.
* Explicit interface implementation, where an interface which is already implemented in a base class can be re-implemented separately in a derived class. Arguably this makes the class harder to understand, in the same way that member hiding normally does.
* Unsafe code. This allows pointer arithmetic etc, and can improve performance in some situations. However, it is not to be used lightly, as a lot of the normal safety of C# is lost (as the name implies). Note that unsafe code is still managed code, i.e., it is compiled to IL, JITted, and run within the CLR.