

# Modul 318

## C# Syntax und Grundlagen





Autor: Urs Nussbaumer

Version: 1.1

Datum: 12.04.2017



## Inhaltsverzeichnis

---

Inhaltsverzeichnis .....	2
1 Code-Beispiele C/C++, Java, Visual Basic, C#.....	3
2 C# Referenz .....	6
2.1 Grundlegende Datentypen .....	6
2.2 Umwandlungs-Funktionen (Datentyp-Konvertierung).....	7
2.3 Operatoren .....	7
2.4 Arrays in C#.....	10
2.5 String-Funktionen.....	11
2.6 Listen und Dictionaries .....	12

## 1 Code-Beispiele C/C++, Java, Visual Basic, C#

### Kommentare

	C/C++	Java	VB	C#
Kommentar bis Zeilenende	// bis Zeilenende	// bis Zeilenende	' bis Zeilenende REM bis Zeilenende	// bis Zeilenende
Eingebetteter Kommentar	int /* Comment */ i;	int /* Comment */ i;		int /* Comment */ i;
Mehrzeiliger Kommentar	/* von bis zum */	/* von bis zum */		/* von bis zum */

### Visual Studio - Tipp:

- Kommentieren: <Ctrl>+K, <Ctrl>+C (comment)
- Ent-Kommentieren: <Ctrl>+K, <Ctrl>+U (uncomment)

### Variablen und Konstanten

	C/C++	Java	VB	C#
Lokale Variable	int i;	int i;	Dim i As Integer	int i;
Globale Variable	bool done;	<i>innerhalb statischer Klasse</i>	<i>innerhalb statischer Klasse</i>	<i>innerhalb statischer Klasse</i>
Instanzattribut (Member-Variable)	char next; // nur C++	private char next;	Private next As Char	private char next;
Statisches Klassenattribut	static double d;	static public double d;	Public Shared d As Double	public static double d;
Eigenschaft (Property)	-	-	Property Name As String ... End Property	public string Name { get; set; }
Konstanten	const int max = 100; // nur C++  #define MAX 100	final int max = 100;	Const Max As Integer = 100	const int Max = 100;

## Einfache Datentypen

	<b>C/C++</b>	<b>Java</b>	<b>VB</b>	<b>C#</b>
Ganzzahlig	char, short, int, long, long long, unsigned ...	byte, short, int, long	Byte, Short, Integer, Long	byte, short, int, long UInt32, UShort, ...
Zeichen	char	char	Char	char
Floating-point	float, double	float, double	Float, Double	float, double
Logik	bool	boolean	Boolean	bool
Text	std::string	String	String	string

## Iterationen (Schleifen)

	<b>C/C++</b>	<b>Java</b>	<b>VB</b>	<b>C#</b>
Zählende Schleifen	for (int i=0; i<8; i++) { ... }	for (int i=0; i<8; i++) { ... }	For i=8 To 0 Step 1 ... Next	for (int i=0; i<8; i++) { ... }
Kopf-gesteuerte Schleife	while (i < 8) ...	while (i < 8) ...	While i < 8 ... End While	while (i < 8) { ... }
Fuss-gesteuerte Schleife	do { ... } while (i < 8);	do { ... } while (i < 8);	Do ... Loop While i < 8  Do ... Loop Until i >= 8	do { ... } while (i < 8);
For-Each Schleife	std::for_each	for (El e : arrEl) ...	For Each i As Integer In ... ... Next	foreach (object o in ...) { ... }

## Ablaufsteuerung

	C/C++	Java	VB	C#
Verzweigung	<pre>if (i &gt; 8) {   ... }</pre>	<pre>if (i &gt; 8) {   ... }</pre>	<pre>If i &gt; 8 Then   ... End If</pre>	<pre>if (i &gt; 8) {   ... }</pre>
2-er Auswahl	<pre>if (i &gt; 8) {   ... } else {   ... }</pre>	<pre>if (i &gt; 8) {   ... } else {   ... }</pre>	<pre>If i &gt; 8 Then   ... Else   ... End If</pre>	<pre>if (i &gt; 8) {   ... } else {   ... }</pre>
Kaskade	<pre>if (i &gt; 8) {   ... } else if (i &lt; 0) {   ... }</pre>	<pre>if (i &gt; 8) {   ... } else if (i &lt; 0) {   ... }</pre>	<pre>If i &gt; 8 Then   ... ElseIf i &lt; 0   ... End If</pre>	<pre>if (i &gt; 8) {   ... } else if (i &lt; 0) {   ... }</pre>
Auswahl	<pre>switch (i) {   case 8:     ...     break;   case 0:     ...     // fall through   case 1:   case 2:     ...     break;   default:     ... }</pre>	<pre>switch (i) {   case 8:     ...     break;   case 0:     ...     // fall through   case 1:   case 2:     ...     break;   default:     ... }</pre>	<pre>Select Case i   Case 8     ...   Case 5     ... End Select</pre>	<pre>switch (i) {   case 8:     ...     break;   case 0:     ...     goto case 1;   case 1:   case 2:     ...     break;   default:     ...     break; }</pre>

## Strukturen und Klassen

	C/C++	Java	VB	C#
Wert-Semantik	<pre>struct MyStruct {   ... }; class MyClass {   ... };</pre>		<pre>Structure MyStruct   ... End Structure</pre>	<pre>struct MyStruct {   ... }</pre>
Objekt-Semantik		<pre>class MyClass {   ... }</pre>	<pre>Class MyClass   ... End Class</pre>	<pre>class MyClass {   ... }</pre>
Aufzählungen (Enumerationen)	<pre>typedef enum { Zero, One, Four=4 } MyNums;</pre>	<pre>enum MyNums { Zero, One, Four }</pre>	<pre>Enum MyNums   Zero   One   Four = 4 End Enum</pre>	<pre>enum MyNums {   Zero,   One,   Four=4 };</pre>

## 2 C# Referenz

### Inhalt

1. Grundlegende Datentypen
2. Umwandlungs-Funktionen
3. Operatoren
4. Arrays
5. String-Funktionen
6. Listen und Dictionaries

Für detailliertere Informationen siehe MSDN: <http://msdn.microsoft.com/de-de/library/618ayhy6.aspx>

### 2.1 Grundlegende Datentypen

C# unterstützt die üblichen Datentypen. Für jeden von C# unterstützten Datentyp ist ein entsprechender .NET Common Language Laufzeittyp vorhanden. So wird der Datentyp `int` in C# beispielsweise dem Typ `System.Int32` in der Laufzeitumgebung zugeordnet. `System.Int32` kann fast überall dort verwendet werden, wo `int` zum Einsatz kommt. Dieses Vorgehen wird jedoch nicht empfohlen, da es die Lesbarkeit des Codes herabsetzt.

Die grundlegenden Datentypen werden in der nachstehenden Tabelle beschrieben.

Typ	Speicherbedarf	Laufzeittyp	Wertebereich
<code>byte</code>	8 Bit	<code>System.Byte</code>	0 bis 255
<code>sbyte</code>	8 Bit	<code>System.SByte</code>	-128 bis 127
<code>short</code>	16 Bit	<code>System.Int16</code>	-32'768 bis 32'767
<code>ushort</code>	16 Bit	<code>System.UInt16</code>	0 bis 65'535
<code>int</code>	32 Bit	<code>System.Int32</code>	-2'147'483'648 bis 2'147'483'648
<code>uint</code>	32 Bit	<code>System.UInt32</code>	0 bis 4'294'967'295
<code>long</code>	64 Bit	<code>System.Int64</code>	-9'223'372'036'854'775'808 bis 9'223'372'036'854'775'807
<code>ulong</code>	64 Bit	<code>System.UInt64</code>	0 bis 18'446'744'073'709'551'615
<code>float</code>	32 Bit	<code>System.Single</code>	+/-1.5 x 10 <sup>45</sup> bis +/-3.4 x 10 <sup>38</sup>
<code>double</code>	64 Bit	<code>System.Double</code>	+/-5.0 x 10 <sup>324</sup> bis +/-1.7 x 10 <sup>308</sup>
<code>decimal</code>	128 Bit	<code>System.Decimal</code>	+/-1.0 x 10 <sup>28</sup> bis +/-7.9 x 10 <sup>28</sup>
<code>char</code>	16 Bit	<code>System.Char</code>	(ein Unicode-Zeichen)
<code>string</code>		<code>System.String</code>	(eine Zeichenfolge)
<code>bool</code>	8 Bit	<code>System.Boolean</code>	true, false

## 2.2 Umwandlungs-Funktionen (Datentyp-Konvertierung)

Funktion	Beschreibung	Beispiele
<code>Format()</code>	Div. Datentypen in formatierten String umwandeln (z.B. Datum)	<code>s = string.Format("Zeit: {0:T}", myDate);</code>
<code>ToString()</code>	Instanz-Methode von <code>object</code> um in einen String zu konvertieren.	<code>s = 25.ToString();</code>
<code>TryParse()</code>	Von bestimmten Klassen (z.B. <code>Int</code> ) angebotene statische Methode. Gibt <code>true</code> zurück, wenn Konvertierung erfolgreich, und speichert das Resultat der Konvertierung im übergebenen <code>out</code> -Parameter.	<code>Int.TryParse(zahlString, out zahl)</code>
<code>Convert.ToInt()</code>	Konvertieren in den Typ <code>int</code> .	<code>int i = Convert.ToInt("100");</code>
<code>Convert.ToDouble()</code>	Konvertieren in den Typ <code>double</code> .	<code>double d = Convert.ToDouble (100);</code>
<code>Convert.ToDateTime()</code>	Konvertieren in den Typ <code>DateTime</code> .	<code>DateTime dt = Convert.ToDateTime("12.05.2015");</code>
<code>(T)Object</code>	Casting mit dem <code>()</code> Operator: Typkonvertierung durch voranstellen des Zieltyps in Klammern.	<code>Dog d = (Dog)animal;</code>

## 2.3 Operatoren

### Arithmetische Operatoren

In der Rangfolge ihrer Auswertung.

Operator	Beschreibung	Beispiele
<code>+</code> , <code>-</code>	Unäre Identität und Negation (=Vorzeichen)	<code>Z = -14      // Z gibt -14</code>
<code>*</code> , <code>/</code>	Multiplikation und Gleitkomma-division	<code>Z = 14 / 4    // Z gibt 3.5</code>
<code>%</code>	Modulooperator (Rest von Ganzzahldivision)	<code>Z = 14 % 4    // Z gibt 2</code>
<code>+</code> , <code>-</code>	Addition und Subtraktion	<code>Z = 14 + 4    // Z gibt 18</code>



## Verkettungsoperatoren

Verknüpfung mehrerer Zeichenfolgen zu einer einzigen Zeichenfolge.

Operator	Beschreibung	Beispiele
+	Zeichenfolgenverkettung.  Liefert einen String oder eine Zahl, je nach verketteten Typen.	<code>Z = 14 + 4 // Z gibt 18</code> <code>Z = "14" + "4" // Z gibt 144</code>

## Logische Operatoren

Was diese Operatoren tun hängt davon ab, was für Datentypen damit verarbeitet werden:

- Bei Datentyp Boolean (Wahr/Falsch), also nur 1 Bit, dann machen sie eine logische Verknüpfung
- Bei Datentyp Zahl (z.B. Integer), machen sie eine bitweise Verknüpfung

Operator	Beschreibung	Beispiele
!	Ist die Bedingung NICHT Wahr?	<code>if (!(5&lt;3)) // True</code>
&& &	Logische Verknüpfung: Beide Seiten wahr?  Bitweise Und-Verknüpfung: Siehe Beispiel	<code>if (2&lt;3 &amp;&amp; 5&lt;3) // False</code> <code>Z = 5 &amp; 3 // Z gibt 1</code> (Binär: 101 AND 011 gibt 001)
 	Logische Verknüpfung: Mind. eine Seite wahr?  Bitweise Oder-Verknüpfung: Siehe Beispiel	<code>if (2&lt;3    5&lt;3) // True</code> <code>Z = 5   3 // Z gibt 7</code> (Binär: 101 OR 011 gibt 111)
^	Logische Verknüpfung: Genau eines wahr?  Bitweise XOR-Verknüpfung.	<code>If (2&lt;3 ^ 5&lt;3) // True</code> <code>Z = 5 ^ 3 '-- Z gibt 6</code> (Binär: 101 XOR 011 gibt 110)

## Bitschiebe Operatoren

Operator	Beschreibung	Beispiele (wenn Z bisher 14 war!)
<<	Zahl Binär um einige Bits nach links schieben	<code>Z = 14 &lt;&lt; 2 // Z gibt 56</code> (Binär: 1110 << 2 gibt 111000)
>>	Zahl Binär um einige Bits nach rechts schieben	<code>Z = 14 &gt;&gt; 2 // Z gibt 3</code> (Binär: 1110 >> 2 gibt 11)

## Zuweisungsoperatoren

Zuweisungsoperator gibt es eigentlich nur einen:

Operator	Beschreibung	Beispiele
=	Zuweisungsoperator (Kopieren von rechts nach links; bisheriger Inhalt links wird überschrieben)	<code>Z = 14 // Z gibt neu 14</code>

In C# können viele arithmetische Operatoren mit dem Zuweisungsoperator kombiniert werden. Der vorhandene Inhalt der Variablen links vom verknüpften Operator wird dann als erster Operand genommen und das Resultat wieder in die Variable links gespeichert, siehe folgende Beispiele:

Operator	Beschreibung	Beispiele (wenn Z bisher 14 war!)
<code>+=, -=</code>	Addition und Subtraktion mit bisherigem Wert	<code>Z += 4 // Z gibt neu 18</code>
<code>*=, /=</code>	Multiplikation und Gleitkommadivision	<code>Z *= 4 // Z gibt neu 56</code>
<code>\=</code>	Ganzzahldivision	<code>Z \= 4 // Z gibt neu 3</code>

## Weitere Verknüpfungen

Operator	Beschreibung	Beispiele (wenn Z bisher 14 war!)
<code>+=</code>	Neuen Text hinten an den bisherigen anfügen.	<code>Msg = "Z="</code> <code>Msg += "4" // Msg ist nun "Z=4"</code>
<code>&lt;&lt;=</code>	Bisheriges Z um einige Bits nach links schieben	<code>Z &lt;&lt;= 2 // Z gibt neu 56</code> (Binär: 1110 << 2 gibt 111000)
<code>&gt;&gt;=</code>	Bisheriges Z um einige Bits nach rechts schieben	<code>Z &gt;&gt;= 2 // Z gibt neu 3</code> (Binär: 1110 >> 2 gibt 11)

## Vergleichsoperatoren

Operator	Beschreibung	Beispiele
<code>&lt;</code>	Kleiner als Vorsicht: bei String-Typen wird alphabetisch (zeichenweise) verglichen!	<code>if (4 &lt; 14) // Wahr</code> <code>if ("4" &lt; "14") // Falsch</code>
<code>&lt;=</code>	Kleiner oder gleich	<code>if (4 &lt;= 4) // Wahr</code>
<code>&gt;</code>	Größer als	<code>if (4 &gt; 14) // Falsch</code> <code>if ("4" &gt; "14") // Wahr</code>
<code>&gt;=</code>	Größer oder gleich	<code>if (4 &gt;= 4) // Wahr</code>
<code>=</code>	Gleich (nur wenn als Bedingung verwendet, sonst ist es ein Zuweisungsoperator!)	<code>if (4 == 4) // Wahr</code>
<code>!=</code>	Ungleich	<code>if (4 != 4) // Falsch</code>
<code>is</code>	Überprüft den Datentyp einer Variablen oder eines Objekts	<code>object obj1 = new ClassX</code> <code>if (obj1 is ClassY) // Falsch</code>

## 2.4 Arrays in C#

### Array-Arten

Wie die meisten Programmiersprachen kennt auch C# eindimensionale und mehrdimensionale Arrays.

Folgend ein Beispiel für ein **eindimensionales** Array:

```
int[] a = new int[5];
```

```
a[0] = 1;
```

```
a[1] = 5;
```

a[0] a[1] a[2] a[3] a[4]

1	5			
---	---	--	--	--

**Mehrdimensionale** Arrays haben die Struktur einer Matrix:

```
int[,] a = new int[5,5];
```

```
a[0,0] = 5;
```

```
a[0,1] = 11;
```

```
a[1,0] = 28;
```

```
a[3,2] = 44;
```

```
a[4,4] = 53;
```

a [0, [1, [2, [3, [4,

,0]	5	11			
,1]	28				
,2]					
,3]			44		
,4]					53

Eine "Spezialform" der mehrdimensionalen Arrays sind die **Jagged**-Arrays was so viel wie "gezackt" bedeutet (bezieht sich auf die Form resp. das Profil der Datenstruktur):

```
int[][] a = new int[4][];
```

```
a[0] = new int[2];
```

```
a[1] = new int[4];
```

```
a[2] = new int[1];
```

```
a[3] = new int[3];
```

```
a[1][3] = 10 ;
```

a[0]				
a[1]				10
a[2]				
a[3]				

### Weblink

Arrays in C#: [http://msdn.microsoft.com/de-de/library/aa288453\(v=vs.71\).aspx](http://msdn.microsoft.com/de-de/library/aa288453(v=vs.71).aspx)

## 2.5 String-Funktionen

### Operatoren und Vergleiche

=	Zuweisung (Kopieren) eines Strings von rechts nach links	<code>myText = txtInput.Text</code>
+	Mehrere String-Stücke aneinander fügen (auch Variablen)	<code>myText = strTeil1 + " " + strTeil2</code>
== !=	Gleich oder Ungleich (z.B. Bedingung in Verzweigung oder Schleife)	<code>if (myText == "Käse")</code> <code>while (myText != "Beenden")</code>
""	Leeres String (zum Initialisieren oder Abfragen) Es kann auch die Konstante <code>string.Empty</code> verwendet werden.	<code>If (myText != "")</code>

### String-Funktionen für Länge, Teilstrings, Suchen und Ersetzen, etc.

Im .NET-Framework gibt es die Klasse "System.String", welche alle wichtigen Methoden für die String-Bearbeitung enthält. **Siehe MSDN-Hilfe.** Jede String-Variable hat automatisch alle diese Methoden zur Verfügung.

Einige Beispiele:

Length	Länge eines Strings ermitteln	<code>int laenge = myText.Length</code>
Substring	Teilstring herauskopieren (1. Zeichen = 0)	<code>int teil = myText.Substring(0, 3)</code>
Contains	Überprüft ob Teilstring in String vorkommt	<code>if (myText.Contains(teil))</code>
Compare	Vergleicht Strings (mehr Möglichkeiten als = / <>)	<code>string.Compare(myText1, myText2)</code>
Replace	Suchen und ersetzen	<code>myText = myText.Replace(alt, neu)</code>
Remove	Löscht Teile aus dem String (1. Zeichen = 0)	<code>teil = myText.Remove(3, 5)</code>
Split	Zerlegt einen String in Array-Elemente	<code>arrTeile = string.Split(ganzerText, ",")</code>
Join	Fügt Array-Elemente zu einem String zusammen	<code>ganzerText = string.Join(", ", arrTeile)</code>
Trim	Löscht führende und nachfolgende Whitespaces	<code>txtInhalt = myText.Trim()</code>
ToLower() ToUpper()	Zu Klein-/Grossschreibung ändern.	<code>string klein = myText.ToLower();</code> <code>string gross = myText.ToUpper();</code>
String.IsNullOrEmpty	Prüft, ob ein String leer ist.	<code>if (string.IsNullOrEmpty(MyText))</code>

## 2.6 Listen und Dictionaries

### Listen

Listen sind dazu da, um eine dynamische Anzahl von Elementen in Listenform zu verwalten. Dabei können die Elemente können sehr einfach per foreach oder über den Index per [] Operator angesprochen werden. Listen können alle möglichen Datentypen beinhalten. Der Typ der Liste wird in den spitzigen Klammern angegeben.

```
// a list taking a simple type
List<string> fruits = new List<string>();
fruits.Add("orange");
fruits.Add("banana");

string banana = fruits[1];
string orange = fruits.First();
fruits.Remove(orange);

public class Person
{
    public string firstName;
    public string lastName;
}

// a list taking a complex type
List<Person> persons = new List<Person>();
...
persons.Add(person1);
persons.Add(person2);

foreach(Person p in persons)
{
    Console.WriteLine(p.firstName + " " + p.lastName);
}
```

### Dictionaries

Dictionaries sind Listen, die für den Zugriff optimiert sind. Dies bringt bei einem foreach nichts, man kann aber die Elemente effizient über einen „Key“ (Schlüssel) direkt ansprechen. Hier ein Beispiel:

```
public class User
{
    public string password;
    public string username;
}

User user1 = new User() { password = "pwd123", username = "HansMueller" };
User user2 = new User() { password = "pwd321", username = "PeterMeier" };

//... imagine hundreds of users here

//key is a string, it must be unique -> we take username as the key
Dictionary<string, User> users = new Dictionary<string, User>();
users[user1.username] = user1;
users[user2.username] = user2;
```