

C# Symbole

Syntax

Name = (letter | '_' | '@') {letter | digit | '_'}

- sämtliche Unicodezeichen sind erlaubt. Aus Gründen der Lesbarkeit sollte man aber nur a-z A-Z 0-9 und den _ verwenden
- Groß/Kleinschreibung ist signifikant
- "@" am Anfang dient zur Unterscheidung von Namen und Schlüsselwörtern
 - if ... Schlüsselwort
 - @if ... Der Name if
- Können Unicode-Escapesequenz enthalten (z.B. \u03c0 für π).

Beispiele

someName

sum_of3

_10percent

@while

Der Name while

\u03c0

Der Name π

b\u0061ck

Der Name back

C# Schlüsselwörter

abstract	as	base	bool	break
byte	case	catch	char	checked
class	const	continue	decimal	default
delegate	do	double	else	enum
event	explicit	extern	false	finally
fixed	float	for	foreach	goto
if	implicit	in	int	interface
internal	is	lock	long	namespace
new	null	object	operator	out
override	params	private	protected	public
readonly	ref	return	sbyte	sealed
short	sizeof	stackalloc	static	string
struct	switch	this	throw	true
try	typeof	uint	ulong	unchecked
unsafe	ushort	using	virtual	void
while				

76 Schlüsselwörter in C# im Gegensatz zu 47 Schlüsselwörtern in Java

C# Namenskonventionen

Indirekt aus Base Class Library ableitbar

Groß/Kleinschreibung

- Jeden Wortanfang groß schreiben (z.B. ShowDialog)
- Anfangsbuchstabe groß (→ Pascalschreibweise), außer bei Variablen, Konstanten und Feldern, die man nicht von außen sieht (lokal und geschützt) (→ Kamelschreibweise)

Konstanten	klein	maximum (public-Konstanten groß, z.B. MaxValue)
Variablen	klein	i,txp, sum
Felder	klein	width, bufferLength (public-Felder groß)
Properties	groß	Length, FullName
Enum-Konstanten	groß	Red , Blue
Methoden	groß	Add , IndexOf
Typen	groß	StringBuilder (vordefinierte Typen klein: int, string)
Namespaces	groß	System, Collections

Erstes Wort

- void-Methoden sollten mit Verb beginnen (z.B. GetHashCode)
- Alles andere sollte mit Substantiv beginnen (z.B. anzahl, IndexOf ,Collections)
- enum-Konstanten oder bool-Members können mit Adjektiv beginnen (Red, Empty)

C# Ganze Zahlen

Syntax

```
DecConstant = digit {digit} {IntSuffix}.  
HexConstant = "0x" hexDigit {hexDigit} {IntSuffix}.  
IntSuffix = 'u' | 'U' | 'l' | 'L'.
```

Typ

ohne Suffix:	kleinster aus int, uint, long, ulong
Suffix u, U:	kleinster aus uint, ulong
Suffix l, L:	kleinster aus long, ulong

Beispiele

17	int
9876543210	long
17L	long
17u	uint
0x3f	int
0x10000	long
0x3fL	long

C# Gleitkommazahlen

Syntax (vereinfacht)

RealConstant = [Digits] ["." [Digits]] [Exp] [RealSuffix].
 muß zumindest 1 Ziffer und entweder ".", Exp oder RealSuffix enthalten

Digits = digit {digit}.

Exp = ("e" | "E") ["+" | "-x] Digits.

RealSuffix = "f" | "F " | "d" | "D " | "m" | "M".

Typ

ohne Suffix: double
 Suffix f, F: float
 Suffix d, D: double
 Suffix m, M: decimal

Beispiele

3.14m decimal
 1E-2 double
 .1 double
 10f float

C# Zeichen und Zeichenketten (1)

Syntax

```
CharConstant = ' char ' .  
StringConstant = " {char} " .
```

char kann sein

beliebiges Zeichen außer Ende-Hochkomma, Zeilenende oder \

Escape-Sequenz

\'	'
\"	"
\\	\
\0	0x0000
\a	0x0007 (alert)
\b	0x0008 (backspace)
\f	0x000c (form feed)
\n	0x000a (new line)
\r	0x000d (carriage return)
\t	0x0009 (horizontal tab)
\v	0x000b (vertical tab)

Unicode- oder Hex-Escape-Sequenz

```
\u0061  a    \x0061  a
```

C# Zeichen und Zeichenketten (2)

Beispiele für Escape-Sequenzen in Zeichenketten

"Datei \\\"C:\\sample.txt\\\"" → Datei "C:\sample.txt"

"Datei \x0022C:\u005c sample.txt\x0022"

Verbatim-Strings - beginnen mit einem @ Zeichen

- enthält keine Steuerzeichen und Escapesequenzen, (\ nicht als Metazeichen)
- das " muss zur Ausgabe doppelt angegeben werden
- dürfen Zeilenumbrüche enthalten

Vorteil: einfachere Schreibweise von Dateinamen:

@\"Datei \"\"C:\sample.txt\"\""

Datei "C:\sample.txt"

C# Kommentare

Zeilenende-Kommentare

`// a comment`

Klammerkommentare

`/* a comment */`

Dürfen nicht geschachtelt werden

Dokumentationskommentare

`/// a documentation comment`

Wird zur automatisierten Generierung von Dokumentationen genutzt. Mit der Compiler-Option `/doc` kann die Dokumentation in eine XML-Datei geschrieben und weiterverarbeitet werden.

C# Konsolenausgabe

System.Console

Über die Klasse Console stehen verschiedene Methoden zur Datenausgabe und Dateneingabe zur Verfügung

System.Console.Write() und ...WriteIn()

Mittels der Methode Write wird ein Text ausgegeben, die Methode WriteIn schließt die Ausgabe dagegen mit einem zusätzlichen Zeilenumbruch ab.

```
Console.Write ("dieser Text ");  
Console.WriteLine ("wird in einer Zeile ausgegeben");  
Console.WriteLine ("dieser Text in einer neuen Zeile!");
```

Neben Text lassen sich auch Daten wie z.B. Zahlen ausgeben. Die einzelnen Daten lassen sich mit + verknüpft ausgeben oder mit Formatangaben:

```
Console.WriteLine("Die Fläche beträgt "+15+" cm²");  
Console.WriteLine("Die Fläche beträgt {0} cm²", flaeche);
```

In geschweiften Klammern werden dazu Parameter, mit 0 beginnend, als Platzhalter eingesetzt. Dabei können auch Formatierungsangaben gemacht werden:

C# Konsolenausgabe

```
Console.WriteLine ("Zahl {0,-15}.", 100);  
Console.WriteLine ("Zahl {0,15}.", 100);
```

Ausgabe:

```
    Zahl 100      .  
    Zahl          100.
```

Übung zur formatierten Ausgabe:

C# Konsoleneingabe

```
int i = Console.Read ( ); // → liest ein Zeichen von Tastatur und speichert  
// es als int
```

```
String s = Console.ReadLine ( ); // → liest eine ganze Zeile, die mit  
// Enter abgeschlossen wurde
```

Will man Zahlen einlesen, muss man den eingegebenen String mit Hilfe der in der Klasse **Convert** enthaltenen Methoden in den entsprechenden Zahlentyp konvertieren.

```
String s = Console.ReadLine ( );  
int i = Convert.ToInt32 (s); // → wandelt Eingabe in int um  
double d = Convert.ToDouble (s); // → wandelt Eingabe in double um
```

Übung: Eingabe von verschiedenen Daten