

Datentypen

Bezeichnung	Erklärung	Darstellung
string	Zeichenkette	"Name"
char	einzelne Zeichen	'a'
bool	Abfragetyp	true / false
int	ganze Zahlen	123
float	Dezimalzahlen (mit ~7 Nachkommastellen)	1,23...
double	Dezimalzahlen (mit ~15 Nachkommastellen)	1,23...
decimal	Dezimalzahlen (mit ~29 Nachkommastellen)	1,23...

Datentypen werden genutzt, um die Art einer Variablen zu definieren.

allgemeiner Aufbau

Methoden	Sichtbarkeit	public
	Rückgabotyp	int
	Methodenname	Zahl
	(<i>Parameter</i>)	(int z,...)
	{ <i>Code</i> }	{ <i>Code</i> }
Klassen	class	class
	Klassenname	Zahl
	{ <i>Code</i> }	{ <i>Code</i> }

Zugriffsmodifizierer (Sichtbarkeit)

public	Der öffentlicher Zugriff ist die uneingeschränkteste Zugriffsebene. Es gibt keine Einschränkungen für den Zugriff auf öffentliche Member.
private	Private Member sind nur innerhalb der Klasse oder Struktur nutzbar.
partial	Dies ist nur ein Teil der Definitionen der Klasse. Weitere Teile der Definition sind in anderen Dateien enthalten.
protected	Auf einen geschützten Member kann innerhalb seiner Klasse und von Instanzen abgeleiteter Klasse zugegriffen werden.
const (konstant)	Konstante Felder und lokale Felder sind keine Variablen und können daher nicht geändert werden.
static	Ein static-Member kann nicht über eine Instanz verwiesen werden. Stattdessen wird er über den Typnamen verwiesen.

Werden Zugriffsmodifizierer in Verbindung mit Variablen genutzt, wird hinter den Zugriffsmodifizierer der Datentyp geschrieben.



Rückgabetypen

Datentypen (int, double, string,...)	Benötigt ein <code>return Variable;</code>
void	Ist einen "leerer" Rückgabetyt

Schleifen

Typ	Allgemein	Beispiel
if-Schleife	<code>if (Bedingung) {Code}</code>	<code>if (a>b) {a=b+1}</code>
if-else-Schleife	<code>if (Bedingung) {Code}</code> <code>else {Code}</code>	<code>if (a>b) {a=b+1} else {b=a+1}</code>
while-Schleife	<code>while (Bedingung) {Code}</code>	<code>while (a>b) {a=a+1}</code>
do-while-Schleife	<code>do {Code} while (Bedingung)</code>	<code>do {a=a+1} while (a>b)</code>
for-Schleife	<code>for (Bedingung) {Code}</code>	<code>for (int i=0; i<a; i+=1) {a=a+1}</code>
try-catch-Schleife	<code>try {Code} catch (Exception) {Code}</code>	<code>try {string s=... ; double x=Convert.ToDouble(s);} catch (Exception ex){MessageBox.Show(ex.Message);}</code>
foreach-Schleife	<code>foreach (Element in Array) {Code}</code>	<code>int[] numbers = { 4, 5, 6, 1, 2, 3, -2, -1, 0 }; foreach (int i in numbers) {Console.WriteLine("{0}", i);}</code>

Ausnahmen (Exceptions)

Auslösen	<code>throw new Exception ("Ausnahme");</code>	
Abfangen	<code>try{Code}</code>	
	<code>catch (Exception ex){MessageBox.Show(ex.Message);}</code>	Alle restlichen Ausnahmen „fangen“
	<code>catch (DivideByZeroException){}</code>	Division durch Null
	<code>catch (FormatException){}</code>	Falsches Zahlenformat
	<code>catch{Code}</code>	auch ohne Argument möglich

Array

allgemeine Definition	<code>Datentyp [] Name = new Datentyp [Größe]</code>
Definition mit konstanter Größe	<code>const Datentyp Name = Größe;</code> <code>Datentyp [] Name2 = new Datentyp [Name];</code>
Definition mit veränderbarer Größe	<code>Datentyp Name = (Datentyp) numerisch; Value;</code> <code>Datentyp2 [] Name2 = new Datentyp [Name];</code>
Ausgabe	<code>for(int i=0; i<n; i=i+1){label1.Text = Name [i]}</code>

Die Nummer des ersten Speicherplatzes ist immer 0.



Arbeiten mit Dateien

Datei öffnen zum lesen	<pre>FileStream fs; fs = new FileStream(fileName, FileMode.Open);</pre>
Datei öffnen zum Schreiben	<pre>FileStream fs; fs = new FileStream(fileName, FileMode.OpenOrCreate);</pre>
Datei lesen	<pre>StreamReader sr = new StreamReader(fs); while (!sr.EndOfStream) { string line = sr.ReadLine(); textBox1.Text += line + System.Environment.NewLine;}</pre>
Datei schreiben	<pre>StreamWriter sw = new StreamWriter(fs); sw.WriteLine("Hello World!");</pre>
Datei schließen	<pre>fs.Close(); sw.Close();</pre>
vorhandene Datei (lesen)	<pre>OpenFileDialog fn = new OpenFileDialog(); fn.ShowDialog();</pre>
vorhanden oder neue Datei (schreiben)	<pre>SaveFileDialog fn = new SaveFileDialog(); fn.ShowDialog();</pre>

Vererbung

<pre>public class Tier {public virtual void Greet() {Console.WriteLine("Hallo, ich bin eine Art Tier!");}}</pre>	Die Klasse "Hund" erbt von der Klasse "Tier". Es werden die Eigenschaften der Klasse "Tier" weitergegeben.
<pre>public class Hund : Tier {public override void Begrüßung() {Console.WriteLine("Hallo, ich bin ein Hund!");}}</pre>	Durch "override" sollen die übergebenen Eigenschaften überschrieben werden und können dadurch angepasst werden.

Vererbung ermöglicht es, eine Basisklasse zu definieren, die eine bestimmte Funktionalität bietet (Daten und Verhalten), und abgeleitete Klassen zu definieren, die diese Funktionalität entweder übernehmen oder außer Kraft setzen.

Input und Output

Input	<pre>Console.ReadLine();</pre>	
Output	<pre>Console.WriteLine();</pre>	Der Output wird auf verschiedene Zeilen geschrieben
	<pre>Console.WriteLine();</pre>	Der Output wird auf die gleiche Zeile geschrieben



Mathematik

Addition	+
Subtraktion	-
Multiplikation	*
Division	/
Restwert	%
Exponent	Math.Pow(x,y)
Wurzel (square-root)	Math.Sqrt(x)

Operatoren

gleich	==
ungleich	!=
kleiner als	<
kleiner gleich	<=
größer als	>
größer gleich	>=
und	&&
oder	

Convert Methoden

Konvertierung zu string	ToString
Konvertierung zu char	ToChar
Konvertierung zu int16 (16-bit)	ToInt16
Konvertierung zu int32 (32-bit)	ToInt32
Konvertierung zu double	ToDouble
Konvertierung zu dezimal	ToDecimal
Konvertierung zu boolean	ToBoolean

Konvertierung des Wertes eines Typs (int, float, double usw.) in einen anderen Typ

Konvertierung kann auch durch davor schreiben des gewünschten Datentypen erzielt werden (z.B. (int) *Variablenname*)

Kommentare

eine Zeile	//
mehrere Zeilen	/* ... */

