Cheatography

C# Cheat Sheet

by anna (anna-ars) via cheatography.com/152662/cs/35521/

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
	Rückgabetyp	int
	Methodenname	Zahl
	(*Parameter*)	(int z,...)
	{ *Code* }	{ *Code* }
Klassen	class	class
	Klassenname	Zahl
	{ *Code* }	{ *Code* }

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 `return Variable;`
void	Ist einen "leerer" Rückgabetyp

Schleifen

Typ	Allgemein	Beispiel
if-Schleife	`if (Bedingung) {Code}`	`if (a>b) {a=b+1}`
if-else-Schleife	`if (Bedingung) {Code} else {Code}`	`if (a>b) {a=b+1} else {b=a+1}`
while-Schleife	`while (Bedingung) {Code}`	`while (a>b) {a=a+1}`
do-while--Schleife	`do {Code} while (Bedingung)`	`do {a=a+1} while (a>b)`
for-Schleife	`for (Bedingung) {Code}`	`for (int i=0; i<a; i+=1) {a=a+1}`
try-catch-Schleife	`try {Code} catch (Exception) {Code}`	`try {string s=... ; double x=Conv ert ToD oul be(s);} catch( Exc - eption ex){Me ssa geB ox.S ho w(e x.M ess age);}`
foreach-Schleife	`foreach (Element in Array){Code}`	`int[] numbers = { 4, 5, 6, 1, 2, 3, -2, -1, 0 };`
		`foreach (int i in numbers) {Conso le.W ri te( "{0} ", i);}`

Ausnahmen (Exceptions)

Auslösen	`throw new Exception ("Ausnahme");`	
Abfangen	`try{Code}`	
	`catch( Exc eption ex){Me ssa geB ox.S how w(" + ex.Mes - sage);}`	Alle restlichen Ausnahmen „fangen“
	`catch( Div ide ByZ ero Exc eption ex){ Code}`	Division durch Null
	`catch( For mat Exc eption ex){ Code}`	Falsches Zahlenformat
	`catch{Code}`	auch ohne Argument möglich

Array

allgemeine Definition	`Datentyp [ ] Name = new Datentyp [Größe]`
Definition mit konstanter Größe	`const Datentyp Name = Größe;`
	`Datentyp [ ] Name2 = new Datentyp [Name];`
Definition mit veränderbarer Größe	`Datentyp Name = (Datentyp) numeri cUp Dow n1.V alue;`
	`Datentyp2 [ ] Name2 = new Datentyp [Name];`
Ausgabe	`for(int i=0; i<n; i=i+1){ label1.Text = Name [ i ]}`

Die Nummer des ersten Speicherplatzes ist immer 0.

Arbeiten mit Dateien

Datei öffnen zum lesen	`FileStream fs;`
	`fs = new FileSt ream(FileName, FileMo de.O pen);`
Datei öffnen zum Schreiben	`FileStream fs;`
	`fs = new FileSt ream(FileName, FileMo de.O pe nOr Cre ate);`
Datei lesen	`Stream Reader sr = new Stream Rea der (fs);`
	`while (!sr.E ndO fSt rea m){ string line = sr.Rea dLi ne();`
	`textBo x1.Text += line +Syste m.E nvi ron men t.N ewL ine;}`
Datei schreiben	`Stream Writer sw = new Stream Wri ter (fs1);`
	`sw.Wri te( " Hello World! ");`
Datei schließen	`fs.Clo se();`
	`sw.Clo se();`
vorhandene Datei (lesen)	`OpenFi leD ialog fn = new OpenFi leD ial og();`
	`fn.Sho wDi alog();`
vorhanden oder neue Datei (schreiben)	`SaveFi leD ialog fn = new SaveFi leD ial og();`
	`fn.Sho wDi alog();`

Vererbung

`public class Tier {public virtual void Greet() {Conso - le.W ri teL ine ("Hallo, ich bin eine Art Tier!");}}`	Die Klasse "Hund" erbt von der Klasse "Tier". Es werden die Eigenschaften der Klasse "Tier" weitergegeben.
`public class Hund : Tier {public override void Begrüs - sung() {Conso le.W ri teL ine ("Hallo, ich bin ein Hund!");}}`	Durch "override" sollen die übergebenen Eigenschaften überschrieben werden und können dadurch angepasst werde.

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	`Consol e.R ead Line;`	
Output	`Consol e.W rit eLi ne();`	Der Output wird auf verschiedene Zeilen geschrieben
	`Consol e.W rite();`	Der Output wird auf die gleich 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	/ ..../