

Was ist ein Programm?

Ein Programm ist eine in Befehlen verfasste Anweisung, die du deinem Computer aufträgst. Normale deutsche Sprache versteht der aber erst mal nicht. Von wegen hochdeutsches Vokabular, Genitiv, 2.Person Präsens!

Das ist erst mal nicht drin. Normale Sprachen sind, wenn man es genau nimmt, viel zu komplex. Sicher kannst du gutes Deutsch sprechen. Denk aber auch daran, dass du etwa sechs Jahre deiner Kindheit gebraucht hast, um es zu lernen. Diese Eingewöhnungszeit wirst du in der Uni nicht bekommen. Du wirst sie aber auch nicht brauchen.

Dein Computer versteht also kein Deutsch, Japanisch oder Englisch, sondern eben Sätze aus Programmiersprachen. Diese haben ein sehr kleines Vokabular, jedoch eine große Anzahl an Regeln auch Syntax genannt. Die Syntax ist, kann man sagen, die Grammatik der Programmiersprache. Aber auch hier gilt "weniger ist mehr" und du wirst in Uni-Kursen die wichtigsten dieser Regeln kennenlernen. Sätze, die du deinem Computer nach Abschluss dieser Kurse vermitteln kannst, sind (ins Deutsche übersetzt):

"Gib mir die Einkaufsliste aus der Datei und addiere die Preise, die ich dahinter notiert habe."

"sag mir welchen Wochentag wir vor vier Jahren, fünf Monaten und vier Tagen hatten."

Wie diese Sätze vermuten lassen, geht es also um die Basics der Programmierung.



einfache Programme in C#

Hello-World Programm

```
Console.WriteLine("Hello World!");
```

Das Konsolenfenster öffnet sich und gibt wieder, was zwischen den Anführungszeichen steht. Warum dies sehr nützlich ist, wirst du später noch genauer erfahren, sobald du weißt, was eine Variable ist. Kopiere die Zeile gerne in dein Projekt und ändere den Text in den " ". Schau was passiert.

Daten

Ein Programm besteht immer aus einer Abfolge aus Instruktionen. Diese legen fest, was das Programm erledigt, wenn du es ausführst.

z.B Das Addieren der Preise auf deiner Einkaufsliste.

Von deiner Liste versteht dein Programm aber erst mal nur Bahnhof. Dass die Preise auf deiner Liste addierbare Zahlen sind, weiß es nämlich erst mal nicht. Um Ihm dies zu vermitteln, nutzen wir in C# vordefinierte Datentypen.

Deine Preise sind Daten und "Zahl" ist der Typ dieser Daten.

1. Daten

Daten haben einen Informationsgehalt, den unser Programm auswerten kann.

1.1 Datentypen

Die zwei grundlegendsten Typen für Daten sind Zahlen und Zeichenketten (Strings)

Zahlen

Ziffern: Von 0 bis 9

Zahl: Aneinanderreihung von Ziffern

„Eine Ziffer ist höchstens einstellig, während eine Zahl mehrere Stellen besitzen kann“

als ganze Zahl(int):

918, 182, 1, 2, 3

als Kommazahl(double):

1.0283, 209983.29823

Zeichenketten (Strings)

Zeichen: kann alles sein, was du auf deiner Tastatur findest und noch mehr.

'a', 'b', 'c', '.', '_', '1', '%', '&',

Zeichenkette: Aneinanderreihung von Zeichen, also ein Text.

"Peter", "1&1", "40% auf alles außer Tiernahrung", "\$t&jjghfr?halloundtschüss"

„Ein Zeichen ist höchstens einstellig, während eine Zeichenkette mehrere Stellen besitzen kann“

Variablen

Daten werden in Variablen gespeichert. Wenn wir Daten brauchen oder ändern wollen, kann unser Programm diese Variablen ansprechen.

Dem Sinne entsprechend könnten Angaben wie folgt lauten:

"Hey Variable, gib mir die Zahl des Preises, den du gespeichert hast!"

"Der alte Preis reduziert sich um 2%, speichre das!"

1. Erstellung von Variablen

Die Variablen müssen immer von dir benannt werden, außerdem muss der Variablentyp angegeben werden. Der Variablentyp entspricht dem Typ der Daten, die in der Variable gespeichert werden sollen.

C# stellt dir verschiedene Variablentypen zur Verfügung:

Texttypen

- **char**(Typ für Zeichen),

```
char meinZeichen = 'c';
```

Variable mit der Bezeichnung meinZeichen vom Typ char, die den Text 'c' speichert

- **string**(Typ für Zeichenketten),

```
string meinString = "Hallo";
```

Variable mit der Bezeichnung meinString vom Typ string, die den Text "Hallo" speichert

Zahlentypen

- **int**(Typ für ganze Zahlen),


- **double**(Typ für Kommazahlen)

```
int ganzeZahl = 1;
```

```
double kommaZahl = 2.5;
```

Variable mit der Bezeichnung ganzeZahl/kommaZahl vom Typ int/double, die den Wert 1/2.5 speichert

Schlüsselwort	Datentyp	syntaktische Regel
<i>char</i>	Zeichen	Zeichen werden immer in einfachen Anführungszeichen ' ' eingeschlossen Entweder kein ' ' oder maximal ein Zeichen z.B 'a'
<i>string</i>	Zeichenketten	Zahlen werden immer in zweifachen Anführungszeichen " " eingeschlossen Entweder keine "" oder beliebig viele Zeichen z.B "wow so ein toller String"
<i>int</i>	ganze Zahlen	Speichert nur Zahlen ohne Nachkommastellen
<i>double</i>	Kommazahlen	Speichert ganze Zahlen und Zahlen mit Nachkommastellen Weil: eine Ganzzahl z.B 1 kann auch wie eine Kommazahl behandelt werden, denn 1 = 1.0

 Es gibt noch weitaus mehr Variablentypen für Zahlen. Sie lassen sich jedoch alle in jeweils in eine von zwei Kategorien unterscheiden. Entweder es handelt sich um einen ganzzahligen Variablentyp oder einen, der auch Zahlen mit Komma zulässt. Im Normalfall kommst du mit int oder double aus. Andere Datentypen werden in diesem Skript nicht behandelt.

2. Variablen ändern

Der Typ einer Variable wird bei der Erstellung festgelegt. Dieser kann nicht mehr geändert werden. Wenn ein neuer Wert in die Variable gespeichert wird, muss er dem Typ der Variable entsprechen.

Variable überschreiben

```
int zahl = 1;
```

Wir weisen der Variable Zahl nun den Wert 2 zu.

```
zahl = 2;
```

```
Console.WriteLine(zahl); // in zahl ist nun der Wert 2 gespeichert
```

Mit Variablen arbeiten

Mit dem Wissen, das du bisher über Variablen gesammelt haben, kannst du nun z.B eine Einkaufsliste in dein Programm übertragen. Angenommen du willst 2x Milch und 1x Kellogs kaufen, dann würde dein Programm wie folgt aussehen.

```
string produkt1 = "Milch";
double preis1 = 1.45;
int anzahl1 = 2;
string produkt2 = "Kellogs";
double preis2 = 2.40;
int anzahl2 = 1;
```

Die Preise und die Mengeneinheiten für den Einkauf sind nun als Zahlen in das Programm implementiert. Mit diesen Zahlvariablen kannst du nun auch Rechnungen anstellen.

Zahlen

Berechnung der Einkaufskosten:

```
double kostenEinkauf = (preis1 * anzahl1) + (preis2 * anzahl2);
Console.WriteLine(kostenEinkauf);
```

Wie in der Schulmathematik „Punkt vor Strich“, „Klammern zuerst“. Erstelle Rechnungen mit Zahldaten, genauso wie du sie auf Papier umsetzen würdest.

String

Manche der Operationszeichen kannst du auch auf Strings anwenden. So zum Beispiel das plus. Mit + kannst du Strings zusammenfügen.

```
Console.WriteLine("Dein Einkauf kostet: " + kostenEinkauf + "Euro");
```

Variablen können auch direkt in einen String eingesetzt werden. Bei vielen Variablen und langen Strings erleichtert das die Lesbarkeit deines Codes erheblich.

```
Console.WriteLine($"Dein Einkauf fuer {produkt1} und {produkt2} kostet: {kostenEinkauf} Euro");
```

Division mit und ohne Rest

Dividierst du zwei ganze Zahlen, so ist das Ergebnis auch eine ganze Zahl.

D.h. Der Rest geht verloren bzw. Was hinter einem Komma stehen würde, wird abgeschnitten.

Bsp. Der Preis für Mikrowellen hat sich erhöht. Früher haben diese 40€ gekostet. Jetzt kostet Sie 46€. Die ganzzahlige Division ergäbe für den neuen Wert das 1-fache des alten Preises. Also keine Preissteigerung.

```
int alterPreis = 40;
int neuerPreis = 46;
int preisSteigerung = neuerPreis / alterPreis;
Console.WriteLine($"Der neue Preis beträgt das {preisSteigerung}-fache des alten Preises");
```

Rest berechnen:

```
int geldMehr = neuerPreis % alterPreis;
Console.WriteLine($"jedoch zahlen wir {geldMehr} Euro mehr");
```

Modulo % gibt uns den Rest einer ganzzahligen Division.

Um das echte Vielfache des neuen Preises zu erhalten, müssen wir die Typen der Variablen zu double ändern.

```
double alterPreis = 40;
double neuerPreis = 46;
double preisSteigerung = neuerPreis / alterPreis;
Console.WriteLine($"Der neue Preis beträgt das {preisSteigerung}-fache des neuen Preises");
```

Symbol	Bezeichnung	Bedeutung
+, -, *	Plus, Minus, Mal	Addition, Subtraktion, Multiplikation
/	Geteilt	Mit int ganzzahlige Division. Mit double Division mit Komma.
%	Modulo	Rest einer ganzzahligen Division

Escape-Sequenzen

Zwischen den "" deines Strings kann jeder von dir gewählte Text stehen. Der Text besteht aus den von dir gewählten Zeichen z.B. 'a', 'j', '@'. Meist werden diese 1 zu 1 übersetzt. Es gibt jedoch auch Zeichenkombinationen, die Textelemente erzeugen, welche nicht durch einzelne Zeichen dargestellt werden können. z.B. Zeilenumbrüche, Backspaces, Anführungszeichen.

Escape Sequenzen bzw. \ in Strings

Diese Kombinationen werden immer mit einem \ eingeleitet.

Ein Zeilenumbruch wird beispielsweise durch ein "\n" erzeugt.

Bsp.

```
string beschrArtikel = "Zustand: unbenutzt\nneu";
Console.WriteLine(beschrArtikel);
```

Nach unbenutzt wird ein Zeilenumbruch eingefügt. Möchtest du dass ein \ wörtlich genommen wird, musst du dem \ ein weiteres \ voranstellen.

Wahrheitswerte(bool)

Ein bool ist ein weiterer grundlegender Variablentyp in c#.

Er gibt an, ob das wofür unsere Variable steht, der Wahrheit entspricht.

Ist die Variable wahr so wird Sie auf true gesetzt, wenn nicht erhält sie den Wert false.

Bsp.

```
```cs
bool IchBinTrocken = true;
Console.WriteLine("es fängt an zu regnen");
IchBinTrocken = false;
```
```

Die logischen-Operatoren

Erinnern wir uns zurück. Mit den Rechenoperatoren +, - oder * können wir

ints oder doubles also Zahlen miteinander verrechnen. Nach den mathematischen Regeln erhalten wir ein Ergebnis,

das auch eine Zahl ist. Ähnlich können wir auch bools miteinander verknüpfen. Das Ergebnis ist auch wieder ein

bool, also true oder false. Es gelten dabei die Regeln der logischen Operationen.

```
```cs
bool esRegnetNicht = false;
bool regenschirmDabei = true;
bool duBistTrocken = esRegnetNicht || regenschirmDabei; // duBistTrocken ist true, weil du einen
Regenschirm dabei hast
```
```

Beispiel für eine oder-Verknüpfung zweier bools mit ||. DuBistTrocken ist das Ergebnis der Verknüpfung und dann wahr,

wenn es nicht regnet oder du einen Regenschirm dabei hast. Mindestens Eine der beiden Variablen muss

also true sein, damit das Ergebnis true ist.

Gleichbedeutend: Nur wenn beide false sind ist auch das Ergebnis false.

Diese Regel gilt für alle Verknüpfungen mit ||.

| Zeichen | Bezeichnung | Bedeutung |
|---------|-----------------|--|
| | Oder-Operator | Ergebnis ist nur dann false, wenn beide Eingangs-bools false sind. |
| && | Und-Operator | Ergebnis ist nur dann true, wenn beide Eingangs-bools true sind. |
| == | Gleich-Operator | Ergebnis ist nur dann true, wenn Eingangs-bools gleich sind. |

7 Variablen

|!= | Ungleich-Operator | Ergebnis ist nur dann true, wenn Eingangs-bools ungleich sind. |
|! | Negations-Operator | kehrt den Wert des folgenden bools um. Aus false wird true und umgekehrt |

Verzweigungen

if-Blöcke und else-Blöcke

Ein if-Block nimmt einen bool entgegen. Wenn er true ist, wird der Code in den {...} ausgeführt.
Ein else-Block wird nur in Kombination mit einem if-Block verwendet.
Er wird nur dann ausgeführt, wenn der if-Block nicht ausgeführt wurde.

```
```cs
bool duBistTrocken = true;
bool esRegnetNicht = true;
Console.WriteLine("es fängt an zu regnen");
esRegnetNicht = false;
if(esRegnetNicht == true){
 Console.WriteLine("Es regnet nicht und du setzt dich vor ein Cafe")
}
else{
 Console.WritLine("Du öffnest den Regenschirm");
}
```
```

Du siehst durch unseren bool konnten wir unser Program entscheidend steuern.
Das Program macht definitiv das eine oder eben das andere "schwarz oder weiß".

else if Blöcke

else if Blöcke können wie if-Blöcke bools auswerten. Wie ein else-Block, werden sie an einen if-Block gehängt und nur dann ausgewertet, wenn der vorhergehende if oder if-else Block nicht ausgeführt wurde.
Merken: Eine Verzweigung beginnt in jedem Fall mit einem if-Block und endet mit einem oder keinem else-Block.

Dazwischen können beliebig viele else-if-Blöcke stehen.

Bsp.

```
```cs
Console.WriteLine("Du bist aufgestanden und schaust auf den Kalender.");
bool esIstUnterDerWoche = false;
bool duHastEinenTermin = true;
bool duMusstLernen = true;
if (esIstUnterDerWoche)
{
 Console.WriteLine("Du hast Uni.");
 Console.WriteLine("...Also packst du deine Sachen und fährst los");
}
else if(duHastEinenTermin && duMusstLernen)
{

```

```
 Console.WriteLine("Du beeilst dich rechtzeitig zu deinem Termin zu kommen");
 Console.WriteLine("Danach...");
 Console.WriteLine("Zack! PC hochfahren, Lernen, erschöpft sein");
}
else if (duHastEinenTermin)
{
 Console.WriteLine("Du beeilst duich rechtzeitig zu deinem Termin zu kommen");
}
else if (duMusstLernen)
{
 Console.WriteLine("Zack! PC hochfahren, Lernen, endlich frei haben!");
}
else
{
 Console.WriteLine("Freizeit digga!");
 Console.WriteLine("...Läuft!");
}
...
```