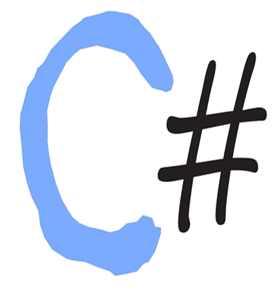
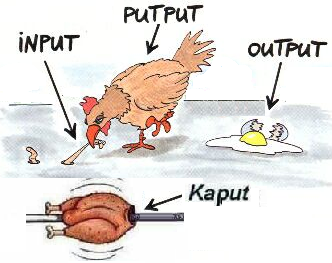
**Modul 319**

***Block 01***

***- EVA mit c# umsetzen -***



**Voraussetzung:**

Dieser Block setzt keine Kenntnisse voraus.

**Dokumentenversion:**

|  |  |
| --- | --- |
| V1.1 | Erstellung durch Roland Bucher |
| V1.2 | Integration der Übungen aus dem Stützkurs |

**Aufbau dieses Blocks:**

Der Block beschreibt zuerst den Unterschied zwischen kompilierenden und interpretierenden Programmiersprachen und vermittelt den Stellenwert von interpretierten Sprachen aus heutiger Sicht. Im zweiten Kapitel wird den Lernenden gezeigt, wie man mittels Verwendung der Programmiersprache c# eine Applikation erstellen kann, die einfache EVA-Aufgaben (Eingabe-Verarbeitung-Ausgabe) mit Texten und ganzen Zahlen auf der konsolenebene ermöglicht. Im dritten Kapitel werden die grundlegenden Datentypen und deren wichtigste Operationen erklärt. Das letzte Kapitel des Dokumentes enthält div. Übungen zu diesen Themen.

**Ziele dieses Blocks:**

* Die Lernenden können mit der Programmiersprache c# einfache Eingabe-Verarbeitung-Ausgabe-Applikationen auf der Konsolenebene erstellen.
* Die Lernenden kennen die Begriffe: Variablen, Datentyp, Deklaration, Initialisierung und Definition von Variablen und können diese Kenntnisse konkret anwenden um geeignete Variablen zu erstellen.
* Die Lernenden können beschreiben, was man unter einem Datentyp versteht und für gegebene Situationen den passendsten aussuchen und verwenden.
* Die Lernenden können Eingaben vom Benutzer entgegennehmen, durch matematische Grundoperationen (+,-,\*,/,%) verarbeiten und ein Ergebnis ausgeben.

**Zeitaufwand:**

Für die Abarbeitung dieses Blockes sind im Unterricht 7 Lektionen vorgesehen. Der Rest ist als Hausaufgabe zu lösen.

**Inhaltsverzeichnis:**

[1 Übersicht über Programmiersprachen 2](#_Toc92015981)

[1.1 Kompilierende versus interpretierende Programmiersprachen 2](#_Toc92015982)

[2 Einfache c# - Konsolenanwendungen erstellen 4](#_Toc92015983)

[3 Grundlegende Datentypen und deren wichtigste Operationen 6](#_Toc92015984)

[3.1 Datentypen 6](#_Toc92015985)

[3.2 Variablen erstellen, implizites und explizites Casting 7](#_Toc92015986)

[3.3 Die wichtigsten Operationen der Datentypen 9](#_Toc92015987)

[4 Übungen (160’) 11](#_Toc92015988)

# Übersicht über Programmiersprachen

## Kompilierende versus interpretierende Programmiersprachen

Der Zentralprozessor bzw. die ALU (Arithmetic and Logic Unit) eines Computers kann nur Maschinen­befehle lesen. Maschinenbefehle sind Elemente einer Sprache, die aus Zahlen bestehen (0 und 1) und sich deshalb schwer programmieren lassen. Aus diesem Grund wurden höhere Programmier­sprachen entwickelt, die mit sprachlichen Elementen arbeiten. Programme, die in solchen Sprachen geschrieben sind, müssen in die Maschinensprache des jeweiligen Prozessors übersetzt werden. Die Aufgabe des Übersetzens übernimmt normalerweise ein Interpreter oder ein Compiler.

Interpreter: übersetzen Programme Zeile für Zeile **zur Laufzeit** in Maschinensprache und werden vom Prozessor ausgeführt. Aus diesem Grund laufen die Programme **langsamer** als kompilierte Programme ab, und es können aus zeitlichen Gründen weniger Optimierungen des Programmcodes vorgenommen werden.

Compiler: übersetzen Programme vollständig in Maschinensprache und speichern das Ergebnis in einer ausführbaren Datei (z.B. exe oder dll). Während der Kompilierung optimiert der Compiler die Programmgrösse und -geschwindigkeit. Dadurch laufen diese Programme viel **schneller** ab als interpretierte Programme.

**Gruppenpuzzleaufgabe «Wichtige Programmiersprachen kennen»:**

Form: Gruppenpuzzle

Zeit: ~35 Minuten + 15 Minuten Hausaufgabe (Step 01 und Step 02)

Beschreibung: Sie bearbeiten den aus dem c’t gescannten Auszug «Uebersicht über Programmiersprachen.pdf». Step 01 und Step 02 werden als Hausaufgabe erledigt

**Step 01:** **Einstieg in das Thema (5’)**

Lesen sie zuerst, jeder für sich, die ersten beiden Seiten (19 und 20) durch und notieren sie sich Fragen und Unklarheiten. Klären sie die Fragen gleich anschliessend im Plenum mit der Lehrperson.

**Step 02: Einarbeitung in eine Sprache / Experte werden (10’)**

Auf je einer Seite werden im Dokument Programmierspachen erklärt. Die Lehrperson weist einer Programmiersprachemind. zwei, im idealfall drei bis vier, Lernende als «Experten» zu. Die Experten lesen zuerst jeder für sich die zugewiesene Seite durch und treffen sich danach in der Expertengruppe (Step 03).

|  |  |  |  |
| --- | --- | --- | --- |
| **Sprachen** | **Experten-Lernende** | **Sprachen** | **Experten-Lernende** |
| c und c++ |  | Scala |  |
| c# |  | Python |  |
| php |  | Swift |  |
| JavaScript (JS) |  | Go |  |
| Java |  | R |  |
| Kotlin |  | Haskell |  |

**Step 03: Expertenaustausch (10’)**

Tauschen Sie sich in der Expertengruppe aus um eine gemeinsame Basis zu bekommen, welche Informationen wichtig sind und den «Novizen» (Nicht-Experten) innert einer Minute mitgeteilt werden sollte.

**Step 04: Austausch (15’)**

Sie treffen sich nun in der Austauschgruppe bei der ein «Experte» jeder Sprache dabei ist. Jeder Experte hat nun 1 Minute Zeit um den Novizen (Nicht-Experten) das Wichtigste weiterzugeben. Die Gruppe erstellt parallel dazu eine gemeinsame Zusammenfassung in Textform oder als Mindmap.

**Step 05: Vorstellung und Abgabe der Zusammenfassung (10’)**

Die entstandenen Zusammanfassungen werden unter den Austauschgruppen ausgetauscht und betrachtet.

**Rechercheaufgabe «Die Idee hinter c# und NET»:**

Form: Einzelarbeit ohne jegliche Kommunikation

Zeit: 25 Minuten

Beschreibung: Lesen sie im Herdt-Buch die beiden Kapitel 2.1 und 2.2 (Seiten 7-10) durch und versuchen sie folgenden Fragen zu beantworten:

**01) Mit welchen Programmiersprachen können sie in der Entwicklungsumgebung Visual Studio programmieren?**

* Visual Basic
* (Visual) C#
* (Visual) C++
* (Visual) F#
* (XAML)
* (Javascript, HTML und CSS) 🡪keine Programmiersprachen

**02) In welchem Verhältnis stehen die Begriffe c#, CLR und .NET Framework zueinander?**

Das sind Libraries die man mit C# ansprechen kann. Also schon ein vorprogrammierter Code den man einfach aufrufen kann. Es wird in eine Zwischensprache übersetzt (plattformneutral). C# ist eine der möglichen Programmiersprachen des .NET Frameworks. CLR (Common Language Runtime) ist ein Teil des .NET Frameworks. Es ist die Laufzeitumgebung welche die vom Compiler erzeugte Zwischensprache ausführen kann.

**03) Was ist der grosse Vor- und Nachteil von «.NET Core» gegenüber «.NET Framework» ?**

.NET Core ist neuer und ist die Zukunft des Frameworks. Das .NET-Framework ist schon eher älter und nicht portabel (nur für Windows). .NET Core ist teils noch experimentell und darum ist das .NET-Framework als Basis oft zuverlässiger.

Vorteil: Es läuft auf mehreren Plattformen.

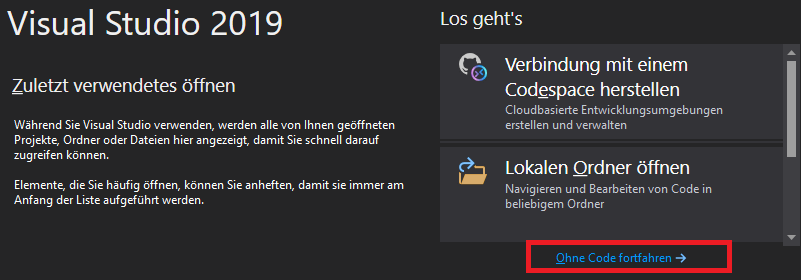
Nachteil: experimentell, unzuverlässiger

**04) Werden mit c# entwickelte Programme kompiliert oder interpretiert?**

Es wird kompiliert. Zuerst wird unser Programmcode in den programmiersprachenunabhängigen Code (Common Intermediate Language, CIL) kompiliert, danach durch die CLR beim Starten der Applikation interpretiert. Man spricht nicht von Interpretieren sondern von Just-In-Time-Compilieren.

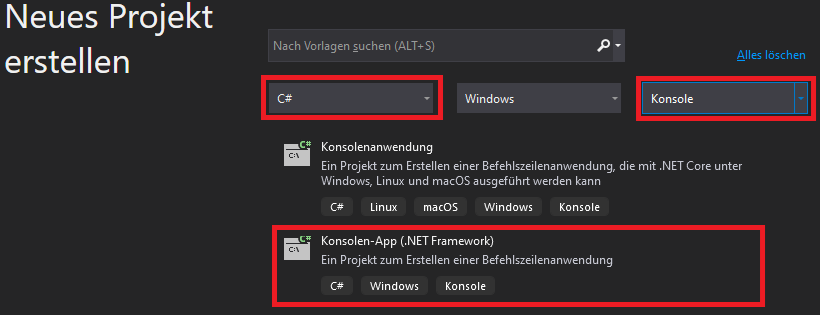
# Einfache c# - Konsolenanwendungen erstellen

Starten Sie das VisualStudio 2019 und wählen sie beim Eingangsfenster «ohne Code fortfahren».

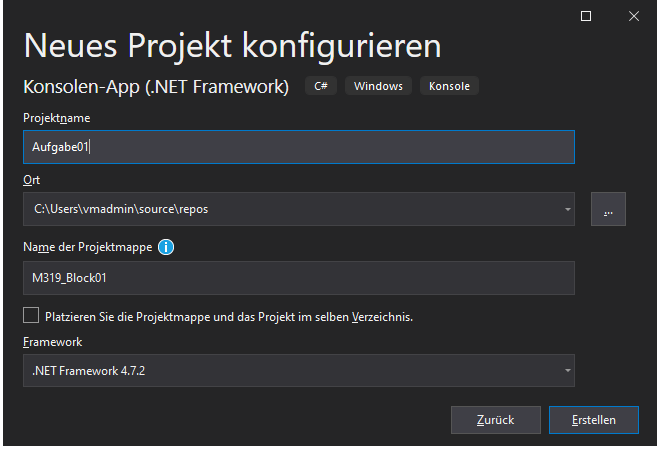


Wählen sie den Menüpunkt Datei🡪 Neu 🡪 Projekt.

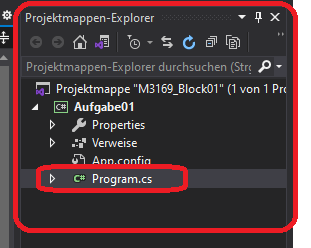
Filtern Sie mit der linken Auswahlliste die Programmiersprache c# und mit der rechten Auswahlliste als Umgebung die Konsole aus. Wählen sie die Konsolenanwendung nach .NET Framework aus



Wählen sie Projektname, Projektmappenname, Lokation und das zu verwendende Framework:



Im Projektmappenexplorer sehen sie die Dateistruktur des Projektes.Klicken sie auf die Datei Program.cs. Passen sie den Programmcode der Hauptfunktion (Main-Methode) wie folgt an:



using System;

namespace Kap02\_Aufgabe01

{

class Program

{

static void Main(string[] args)

{

string vorname="";

Console.Write("Bitte geben Sie ihren Vornamen ein:");

vorname = Console.ReadLine();

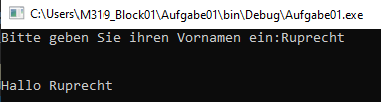
Console.WriteLine("\n\nHallo " + vorname);

Console.ReadLine();

}

}

}



Starten sie das Programm mit der Funktionstaste «F5» oder dem grünen «Starten» Button in der Toolbar



Daraufhin sollte nebenstehende Ausgabe erscheinen:

Die Lehrperson erklärt ihnen den Programmcode. Machen sie sich Notizen zu den fünf Befehlszeilen.

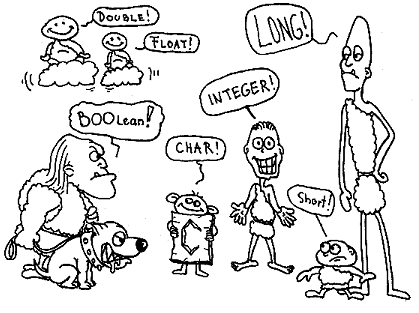
|  |  |
| --- | --- |
| string vorname=""; | Es wird eine Variable erstellt.  Eine Variable entspricht einem im Arbeitsspeicher für uns reservierten Bereich auf den wir lesend und schreibend zugreifen können.  Die hier erstellte Variable können wir unter der Bezeichnung **vorname** adressieren.  Je nach dem auf welcher Seite des Gleiches (=) die Variablenbezeichnung ist, wird gelesen oder geschrieben: \*) vorname = ….. (Schreiben in die Speicherstelle)  \*) ….= vorname (Lesen aus der Speicherstelle)  Jede Variable basiert auf einem Datentyp. Hier ist der Datentyp string. String bedeutet, dass die Informationen, die dort gespeichert sind, einer Zeichenkette entsprechen. Es werden also Unicode Werte für jedes Zeichen gespeichert bis und mit Schlusszeichen (\0). |
| Console.Write("Bitte...Vornamen ein:"); | Die Methode Write() der Klasse Console gibt dem Benutzer einen Text aus, ohne am Schluss einen Zeilenumbruch zu machen:    Was ausgegeben werden soll muss als Zeichenkette übergeben werden. Eine Zeichenkette steht immer zwischen doppelten Anführungs- und Schlusszeichen: "…" |
| vorname = Console.ReadLine(); | Der Aufruf von Console.ReadLine() wird komplett durch die Eingabe des Benutzers ersetzt. ReadLine() liefert immer nur Zeichenketten (string) zurück.  Hier wird als die Eingabe des Benutzers in der Variablen vorname (im Arbeitspeicher) gespeichert.  Die Methode ReadLine() der Klasse Console verlangt im Gegensatz zur Write() Methode keinen Parameter, darum steht nichts zwischen den Parmeterklammern ( ). |
| Console.WriteLine("\n\nHallo "+vorname); | Die Methode WriteLine() der Klasse Console gibt dem Benutzer einen Text aus. Am Schluss wird automatisch ein Zeilenumbruch gemacht. Der Text , der ausgegeben werden soll, muss innerhalb der Parameterklammern ( ) übergeben werden.  Hier wird eine Zeichenkette aus zwei Zeichenketten gebildet. Das + bedeutet bei Zeichenketten, dass die Zeichenketten aneinander gehängt werden.  Die erste Zeichenkette besteht aus zwei Zeilenumbrüchen \n und dem Text Hallo. Die zweite Zeichenkette wird aus dem Arbeitsspeicher, aus der Variablen vorname gelesen. |
| Console.ReadLine(); | Die letzte Anweisung ReadLine() benötigen wir nur, damit das Programm sich nicht automatisch beendet, ohne dass wir die Ausgabe lesen können. |

Kapitel 6.3 Seite 45 unten und Seite 46 den Abschnitt Aufbau der Startklasse einer Konsolenanwendung

# Grundlegende Datentypen und deren wichtigste Operationen

*(Eine Zusammenfassung aus* *ihrem Herdt-Buch C# - Grundlagen der Programmierung mit VisualStudio 2019, Kap. 6.6 bis 6.9)*

## Datentypen



Damit ein Computerprogramm eine Eingabe speichern kann, muss eine Variable erstellt werden. Einer Variable wird vom Betriebssystem ein Platz im Arbeitsspeicher zugewiesen. Je nach Art der zu speichenden Information benötigen wir mehr oder weniger Platz. Auch können die Struktur und der Aufbau der 0 und 1 Informationen ganz unterschiedlichen Typs sein (z.B. UniCode vs. Integer oder FloatingPoint Format). Dies alles regelt der Datentyp. Eine Variable basiert bei uns immer auf einem Datentyp.

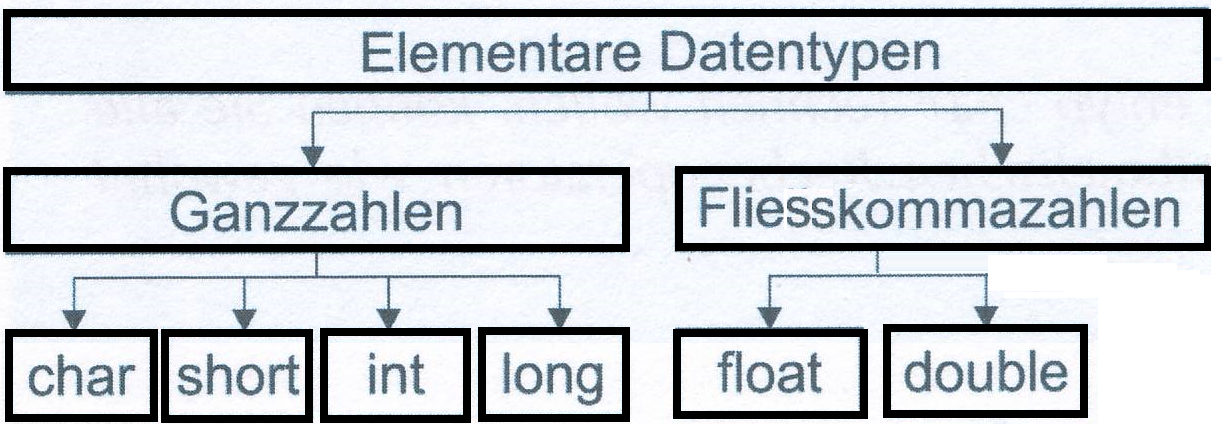
**int groesse = 12;**

Die obenstehende Anweisung definiert eine Variable des Typs Integer mit dem Namen groesse. Variablen des Typs Integer können nur ganze Zahlen im Bereich von -2’147’483’648 bis +2'147'483’647 speichern.

Man unterscheidet folgende Hauptkategorien von Datenttypen:

* Zahlen (numerische Daten, z.B. 15 oder 120.98 oder -36.50),
* Zeichen (alphanumerische Daten, z.B. «Meier)
* boolesche (logische Daten, z.B. true oder false)
* Datumsangaben (z.B. 03.12.2025).

**Numerische Datentypen:**Die numerischen Datentypen werden dann benötigt, wenn im Programm mit Zahlenwerten gearbeitet werden soll. Es werden Integer-Datentypen (ganze Zahlen) und Gleitkomma-Datentypen (gebrochene Zahlen) unterschieden.



Datentypen um ganze Zahlen zu speichern:

|  |  |  |
| --- | --- | --- |
| Datentyp | Wertebereich | Speichergrösse |
| sbyte | -128 … +127 | 1 Byte |
| short (Int16) | -32768 … +32767 | 2 Byte |
| int (Int32) | -2'147'483'648 … +2'147'483'647 | 4 Byte |
| long (Int64) | -9'223'372'036'854'775'808 … +9'223'372'036'854'775'807 | 8 Byte |
| byte | 0.. +255 | 1 Byte |
| ushort (UInt16) | 0.. +65’535 | 2 Byte |
| uint (UInt32) | 0.. +4'294'967’295 | 4 Byte |
| ulong (UInt64) | 0.. +18'446'744'073'709'551’615 | 8 Byte |

Datentypen um gebrochene Zahlen zu speichern:

|  |  |  |  |
| --- | --- | --- | --- |
| Datentyp | Wertebereich | Genauigkeit | Speichergrösse |
| float (Single) | ~ +/- 1.4 x 10-45 … 3.4 x 1038 | 7-8 Stellen | 4 Byte |
| double (Double) | ~ +/- 4.9 x 10-324 … 1.8 x 10308 | 15-16 Stellen | 8 Byte |

Der Datentyp decimal (Decimal) besitzt eine hohe Genauigkeit (28 Nachkommastellen) und ist vor allem für Finanz- und Währungsberechnungen geeignet, da Rundungsfehler seltener auftreten als bei double oder float. Dennoch sind Rundungsprobleme nicht ausgeschlossen. Man kann diesen Datentyp als ganze Zahl ➀ oder als Zahl mit festen Nachkommastellen ➁ einsetzen.

|  |  |  |  |
| --- | --- | --- | --- |
| Datentyp | Wertebereich | Genauigkeit | Speichergrösse |
| decimal (Decimal) | ➀ 0 … +/- 79'228'162'514'264'337'593'543'950’335 | 28 Stellen | 16 Byte |
| ➁ +/- 1.0 x 10-28 … 7.9 x 1028 |

*INFO: Ab dem .NET Framework 4.0 wurden zum Rechnen mit sehr grossen sowie mit komplexen Zahlen die Typen BigInteger und Complex bereitgestellt. Allerdings handelt es sich hierbei nicht mehr um primitive (grundlegende) Datentypen.*

**Zeichen Datentypen:**Zeichen-Datentypen in C# können beliebige Zeichen oder Zeichenketten des sogenannten Unicode-Zeichensatzes enthalten.

|  |  |  |
| --- | --- | --- |
| Datentyp | Wertebereich | Speichergrösse |
| char (Char) | Ein Unicode-Zeichen (0…65535) | 2 Byte |
| string (String) | Mehrere Unicode-Zeichen (0…65535) | Max. 4 GByte |

*INFOS: Achtung!! Unterschied zwischen 'a' und "a" erkennen!*

*\t = Tabulator, \n = Zeilenumbruch, \u… = Spezialzeichen (z.B. \u0045 = E)*

|  |  |
| --- | --- |
| *Reguläres String-Literal* | *Verbatim String-Literal* |
| *"c:\\Daten\\M319"* | *@"c:\\Daten\\M319"* |
| *"Klicke auf* ***\"****GO****\"*** *"* | *"Klicke auf* ***""****GO****""*** *"* |

**Boolescher (logischer) Datentyp:**

Zur Repräsentation von Wahrheitswerten (wahr bzw. falsch) dient der Datentyp bool. Ein Wahrheitswert kann nur die Literale true oder false als Wert annehmen.

|  |  |  |
| --- | --- | --- |
| Datentyp | Wertebereich | Speichergrösse |
| bool (Boolean) | True, false | 1 Byte |

**Datumsangaben:**Datumsangaben lassen sich mit dem Datentyp DateTime speichern. Hier handelt es sich allerdings nicht mehr um einen primitiven (grundlegenden) Datentypen. Das erkennt man auch an der Grossschreibung

Intern wird das Datum als ganze Zahl gespeichert.

|  |  |  |
| --- | --- | --- |
| Datentyp | Wertebereich | Speichergrösse |
| DateTime | 1. Jan. 0001 bis 31.Dez. 9999, 0:00:00 bis 23:59:59 | 8 Byte |

*INFOS: Datumsangaben können Sie nicht direkt durch eine Wertzuweisung eingeben:*

*DateTime datum1 = Convert.ToDateTime("01.01.2020");*

*DateTime datum2 = DateTime.Parse("01.01.2020");*

## Variablen erstellen, implizites und explizites Casting

Die Deklaration von Variablen wird mit dem Namen des Datentyps eingeleitet. Anschliessend folgt der Variablenname (identifier = Bezeichner). Mehrere Variablen desselben Typs können Sie in einer Anweisung definieren. Die Variablennamen werden dabei durch Kommata getrennt. Lokale Variablen beginnen üblicherweise mit einem Kleinbuchstaben.

|  |
| --- |
| **type1 identifier1 [, identifier2 …] ;** |

Beispiele für Variablendeklarationen:

|  |
| --- |
| **Korrekte Variablendeklarationen:**  int zahl;  double anzahl, gewicht, preis;  char zeichen;  **Falsche Variablendeklarationen:**  int &zahl; //geht nicht, weil der Bezeichner nicht mit \_ oder a..Z beginnt  double Kosten der Rechnung; //geht nicht, weil der Bezeichner keine Leerzeichen enthalten darf |

**Implizite und explizite Typkonversion**

Der Compiler nimmt nicht in jedem Fall eine automatische Konvertierung von einem zum anderen Datentyp vor. Dies geschieht nur, wenn Sie z.B. den Inhalt einer int-variablen einer float-Variablen zuweisen. Man spricht dabei von einer impliziten Typkonversion. Implizite Typkonversion geschieht nur, wenn ein Umwandeln in einen kompatiblen Typen betreffend Wertebereich und Genauigkeit und mit genügend Speichergrösse erfolgt.

|  |  |
| --- | --- |
| **Implizite Typkonversion** | |
| **Von Typ** | **in Typ** |
| byte | short, char, int, long, float, double |
| short | int, long, float, double |
| char | int, long, float, double |
| int | long, float, double |
| long | float, double |
| float | double |

Bei zuweisungskompatiblen Datentypen führt C# automatisch eine implizite Typkonversion durch. So können Sie beispielsweise eine Ganzzahl (Typ int) problemlos einer Variablen vom Typ double zuweisen. Sind die Datentypen nicht zuweisungskompatibel, so können Sie eine explizite Typkonversion durchführen. Die explizite Typkonversion wird auch als Casten (engl. casting) bezeichnet.

**Möglichkeiten zur expliziten Typkonversion:**

Die Klasse Casting:

|  |
| --- |
| short zahl1 = 5000;  int zahl2 = **(short)**zahl1; |

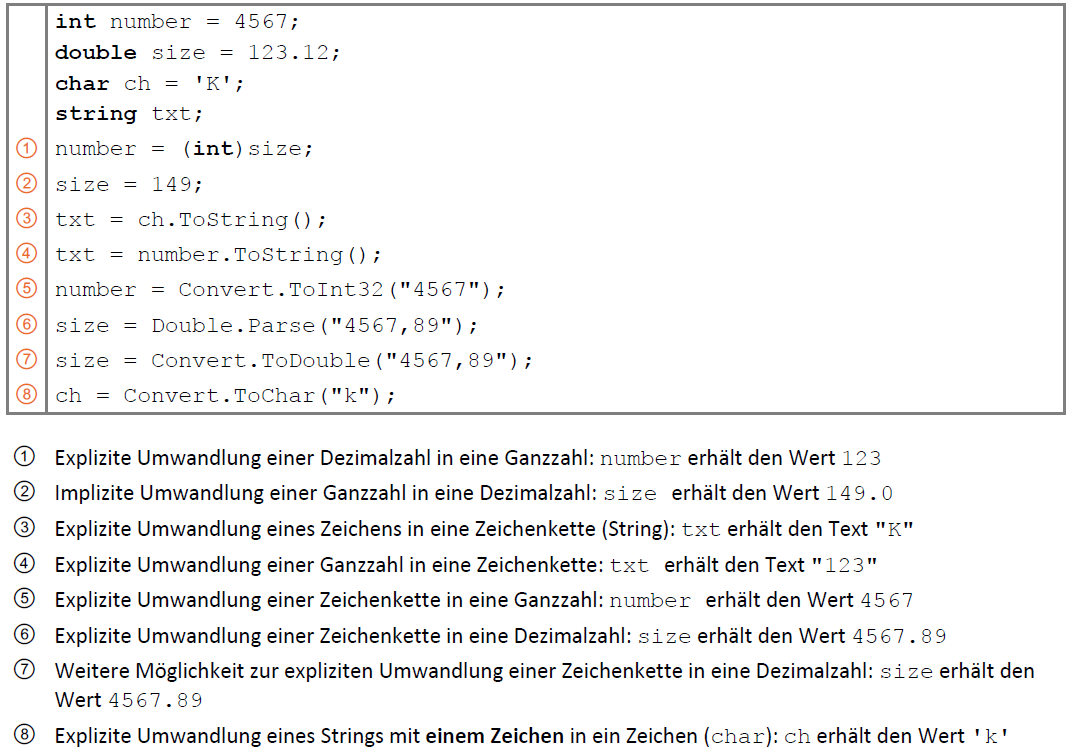
Die Klasse Convert:

|  |
| --- |
| string eingabe = "6024";  int zahl =**Convert.ToInt32(**s**)**; |

Die Methoden ToString() und Parse():

|  |
| --- |
| int zahl1 = 5000;  string s = **zahl1.ToString()**; //entspricht … = Convert.ToString(zahl1);  string eingabe = "6024";  zahl1 = **Int32.Parse(**eingabe**)**; |

Beispiele für Typumwandlungen (Typkonversionen):



## Die wichtigsten Operationen der Datentypen

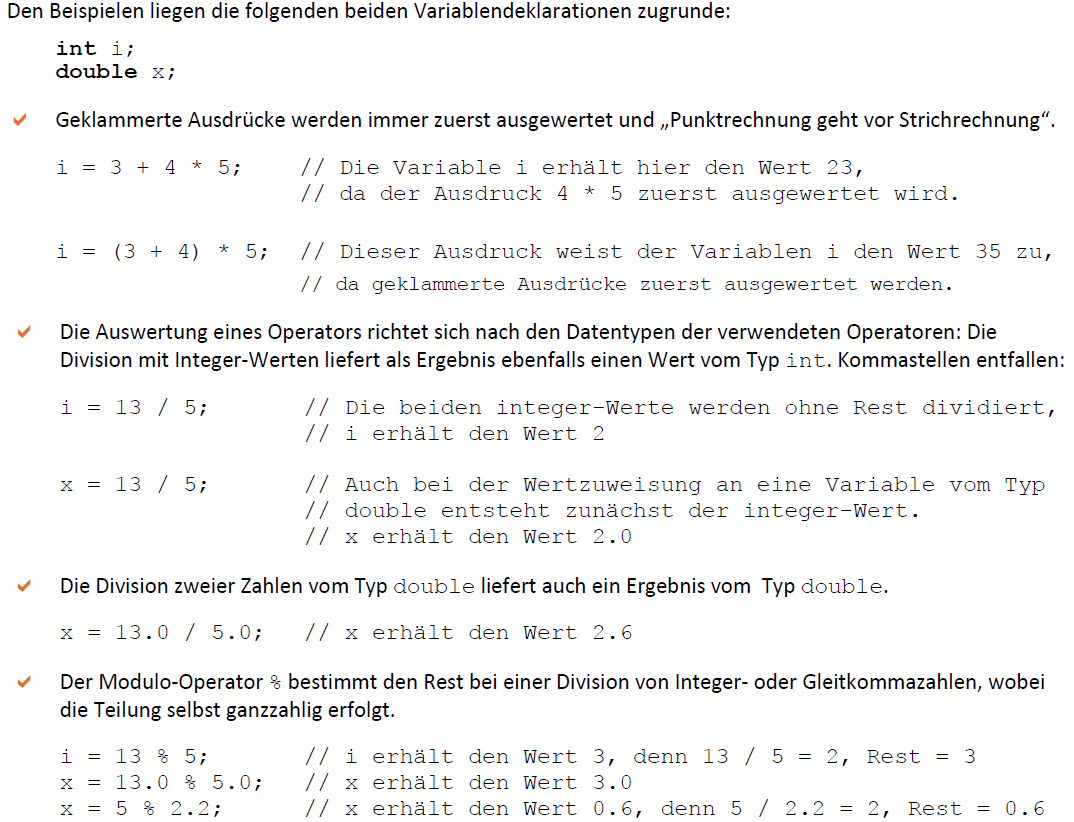
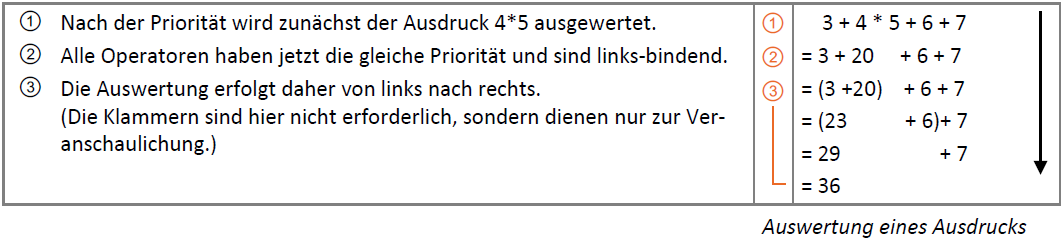
Folgende Operatoren können Sie in C# verwenden:

* **Arithmetische Operatoren,** z. B. für die einfachen Grundrechenarten
* **Vorzeichenoperatoren** für die Kennzeichnung positiver bzw. negativer Zahlen
* **Verkettungsoperatoren** zur Verbindung von Zeichenketten
* **Inkrementierungs-/Dekremetierungsoperatoren** zur Erhöhung oder Verringerung eines Wertes um den Betrag 1
* **Vergleichsoperatoren** zum Vergleich zweier Werte, z. B. auf Gleichheit oder Ungleichheit
* **Verknüpfungsoperatoren** zur logischen Verknüpfung boolescher Ausdrücke bzw. zur binären Verknüpfung binärer Werte
* **Zuweisungsoperatoren** für die verkürzte Schreibweise von Berechnungen mit **arithmetischen Operatoren** und anschliessender **Wertzuweisung**

**Arithmetische Operatoren (+, -, \*, /, %):**

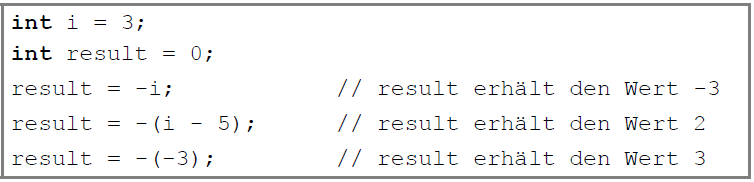
Zu den arithmetischen Operatoren gehören in C# die Grundrechenarten Addition (+), Subtraktion (-) , Multiplikation (\*), Division (/) und der Modulo-Operator (%) der die Rest der Division bestimmt.

Die Grundrechenarten können auf Integer- und auf Gleitkommawerte angewendet werden. Bei der Anwendung der Operatoren gilt wie in der Mathematik die Regel „**Punktrechnung geht vor Strichrechnung**“.



**Vorzeichenoperatoren (+/-):**

Vorzeichenoperatoren (+, -) werden als unäre Operatorenbezeichnet. Die unären Vorzeichenoperatoren können nicht nur auf Zahlen, sondern auch auf Variablen bzw. auf ganze Ausdrücke angewendet werden. Sie besitzen eine höhere Priorität als die binären (arithmetischen) Operatoren.



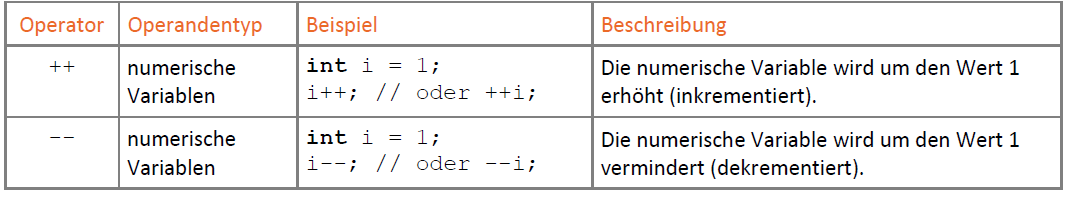
**Verkettungsoperatoren (+):**



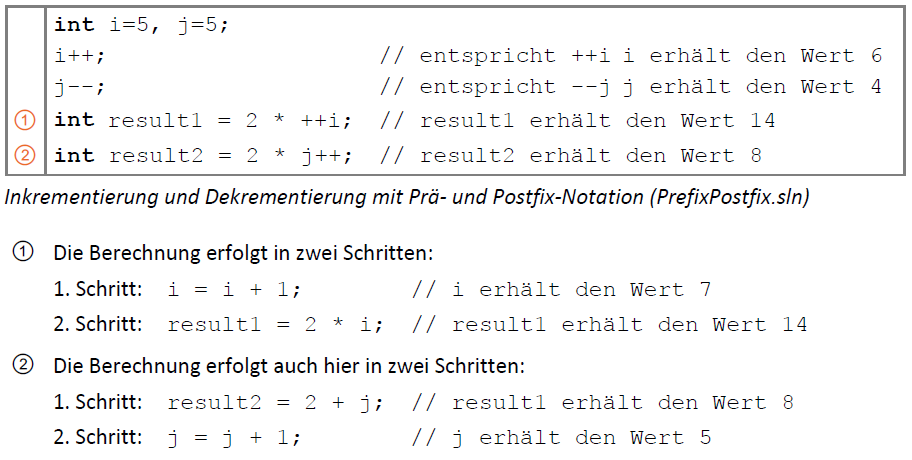
Mehrere Zeichenketten können Sie zu einer Zeichenkette zusammenfügen. Dazu verwenden Sie den Verkettungsoperator +.

**Inkrementierungs-/Dekremetierungsoperatoren (++, --):**

Da man sehr häufig eine Variable um 1 erhöhen oder erniedrigen muss, gibt es diese speziellen Operatoren. Bei den Operatoren ++ und -- wird zwischen Postfix- und Präfix-Notation unterschieden. Die Operatoren stehen entsprechend hinter bzw. vor der Variablen. Bei der Postfix-Notation wird die Variable nach dem Auswerten des Ausdrucks inkrementiert bzw. dekrementiert, bei der Präfix-Notation davor.



Beispiele:



**Vergleichs und Verknüpfungsoperatoren:**

Betrachten wir im nächsten Block (Kontrollstrukturen).

**Zuweisungsoperatoren für eine verkürzte Schreibweise verwenden:**

Häufig treten Anweisungen auf, mit denen der Wert einer Variablen ausgelesen, verändert und anschliessend wieder in derselben Variablen gespeichert wird. Für diese Anweisungen gibt es in C# verkürzte Schreibweisen:

# Übungen (160’)

**Aufgabe 01 (Einfache Eingaben und Ausgaben entwickeln können)**

**Lernziele**: Sie kennen die verschiedenen Datentypen und können vom Benutzer eingegebene Werte in sinnvollen primitiven Datentypen speichern und wieder ausgeben.

**Zeit**: 25’

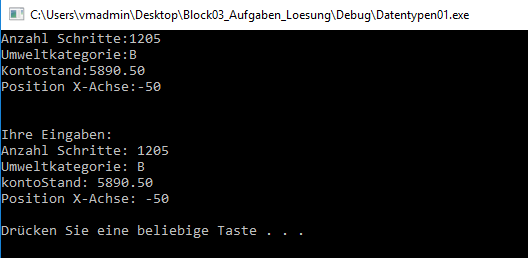
**Aufgabe**: Erstellen Sie ein Programm, das den Benutzer nach dem Vornamen, Namen und Alter fragt. Gibt der Benutzer z.B. Gwendolin Huggentobler und 20 ein, soll danach die Ausgabe «Hallo Gwendolin Huggentobler, bald wirst du 21» ausgegeben werden.

**Ihre Lösung (Kopieren sie am Schluss ihre Lösung hier hinein):**

|  |
| --- |
| static void Main(string[] args) {  string vorname = "";  string nachname = "";  int alter = 0;  Console.Write("Bitte geben Sie Ihren Vornamen ein: ");  vorname = Console.ReadLine();  Console.Write("Bitte geben Sie Ihren Nachnamen ein: ");  nachname = Console.ReadLine();  Console.Write("Bitte geben Sie Ihr Alter ein: ");  alter = Convert.ToInt32(Console.ReadLine());  Console.Write("Hallo " + vorname + " " + nachname);  alter++;  Console.Write(" du wirst " + alter + " nächstes Jahr");  Console.ReadLine();  } |

**Aufgabe 02a (Programmcode schreiben, der sinnvoll gewählte Variablen einliest und ausgibt)**

**Lernziele**: Sinnvolle Variablen deklarieren und diese ein- und ausgeben können



**Zeit**: 25’

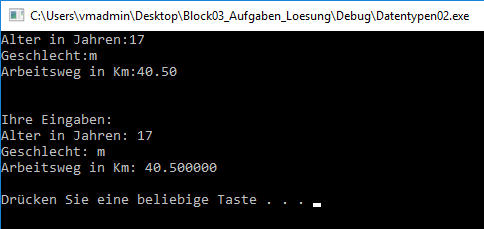
**Aufgabe**: Versuchen Sie den Programmcode der nachfolgend per Printscreen abgebildeten Software zu realisieren. Zuerst wird der Benutzer aufgefordert, vier Werte einzugeben. Danach werden nach zwei Zeilenumbrüchen die eingegebenen Werte ausgegeben. Erstellen sie dazu zuerst die nötigen Variablen auf der Basis von sinn­vollen und ressourcen­schonenden Datentypen ). Danach erfolgt die Erfassung der Benutzerdaten mittels Console.ReadLine() und am Schluss die Ausgabe mittels Console.WriteLine()-Befehl.

**Ihre Lösung (Kopieren sie am Schluss ihre Lösung hier hinein):**

|  |
| --- |
| static void Main(string[] args) {  int schritte = 0;  double kontostand = 0;  double xAchse = 0;  String adresse = "";  Console.Write("Anzahl Schritte: ");  schritte = Convert.ToInt32(Console.ReadLine());  Console.Write("Kontostand: ");  kontostand = Convert.ToDouble(Console.ReadLine());  Console.Write("Position der X-Achse: ");  xAchse = Convert.ToDouble(Console.ReadLine());  Console.Write("Ihre Adresse: ");  adresse = Console.ReadLine();  Console.WriteLine("");  Console.WriteLine("Ihre Eingaben:");  Console.WriteLine(" ");  Console.WriteLine("Anzahl Schritte: " + schritte);  Console.WriteLine("Ihr Kontostand: " + kontostand);  Console.WriteLine("Position der X-Achse: " + xAchse);  Console.WriteLine("Ihre Adresse: " + adresse);  Console.ReadLine();  } |

**Aufgabe 02b (Programmcode schreiben, der sinnvoll gewählte Variablen einliest und ausgibt)**

**Lernziele**: Sinnvolle Variablen deklarieren und diese ein- und ausgeben können



**Zeit**: 25’

**Aufgabe**: Versuchen Sie den Programmcode der nachfolgend per Printscreen abgebildeten Software zu realisieren. Zuerst wird der Benutzer aufgefordert, vier Werte einzugeben. Danach werden nach zwei Zeilenumbrüchen die eingegebenen Werte ausgegeben. Erstellen sie dazu zuerst die nötigen Variablen auf der Basis von sinn­vollen und ressourcen­schonenden Datentypen ). Danach erfolgt die Erfassung der Benutzerdaten mittels Console.ReadLine() und am Schluss die Ausgabe mittels Console.WriteLine()-Befehl.

**Ihre Lösung (Kopieren sie am Schluss ihre Lösung hier hinein):**

|  |
| --- |
| static void Main(string[] args) {  Console.WriteLine("Aufgabe 02b: Informationen einlesen und ausgeben");  Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  Console.ReadLine();  } |

**Nicht machenAufgabe 03 (Auf der Basis von Eingaben neue Informationen berechnen und ausgeben)**

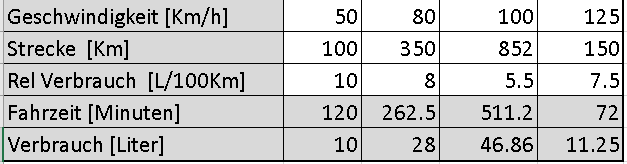
**Lernziele**: Sie kennen die arithmetischen Operationen und können mit deren Hilfe einfache Berechnungsalgorithmen realisieren

**Zeit**: 25’

**Aufgabe**: Erstellen Sie ein Programm, das folgende Aufgabe löst:

Fragen Sie den Benutzer wie schnell sich ein Auto bewegt in Km/h. Fragen sie danach nach der zu fahrenden Distanz in Km und dem Verbrauch des Autos in Liter pro 100Km. Geben sie am Schluss die Fahrzeit in Minuten und den Benzinverbrauch in Liter aus

**Testdaten**:



**Ihre Lösung (Kopieren sie am Schluss ihre Lösung hier hinein):**

|  |
| --- |
| static void Main(string[] args) {  double geschwindigkeit = 0;  double distanz = 0;  double verbrauch = 0;  Console.Write("Geschwindigkeit in Km/h: ");  geschwindigkeit = Convert.ToDouble(Console.ReadLine());  Console.Write("Distanz in Km: ");  distanz = Convert.ToDouble(Console.ReadLine());  Console.Write("Verbrauch in Liter pro 100Km: ");  verbrauch = Convert.ToDouble(Console.ReadLine());  Console.WriteLine("Die Fahrzeit beträgt " + distanz / geschwindigkeit \* 60 + " Minuten");  Console.WriteLine("");  Console.WriteLine("Verbrauch in Liter: " + (distanz / 100) \* verbrauch);  Console.ReadLine();  } |

**Aufgabe 04 (Auf der Basis von Eingaben neue Informationen berechnen und ausgeben)**

**Lernziele**: Die Lernenden können Eingaben vom Benutzer entgegennehmen, durch matmatische Grundoperationen (+,-,\*,/,%) verarbeiten und ein Ergebnis ausgeben

**Zeit**: 25’

**Aufgabe**: Schreiben Sie ein Programm, das den Benutzer zur Eingabe einer ganzen Zahl im Bereich von 0 bis 100’000 auffordert und die Eingabe in einer passenden Variable speichert. Im Verarbeitungteil verändert ihr Programm die eingegebene Zahl so, dass auf das nächstgelegene Vielfache von 100 auf- bzw. abgerundet wird. Die so gerundete Zahl wird am Schluss dem Benutzer ausgegeben.

Zum Beispiel wird 350 auf 400 aufgerundet, 12’446 jedoch auf 12’400 abgerundet.

**Tipp:** Die bestehende Funktion Math.Round() rundet auf den nächsten ganzzahligen Wert.  
int x = 46.59

Int res = Math.Round(x); // der Variablen res wird 47 zugewiesen

**Ihre Lösung (Kopieren sie am Schluss ihre Lösung hier hinein):**

|  |
| --- |
| static void Main(string[] args)  {  double x = 0;  Console.WriteLine("Geben Sie eine Zahl ein: ");  x = Convert.ToDouble(Console.ReadLine())/100;  Console.WriteLine("Der nächste Hunderter von der eingegebenen Zahl ist " + Math.Round(x)\*100); Console.WriteLine("");  Console.Write("Drücken sie eine beliebige Taste zum verlassen des Programmes");  Console.ReadKey(true);  } |

**Aufgabe 05 (Auf der Basis von Eingaben neue Informationen berechnen und ausgeben)**

**Lernziele**: Die Lernenden können Eingaben vom Benutzer entgegennehmen, durch matmatische Grundoperationen (+,-,\*,/,%) verarbeiten und ein Ergebnis ausgeben

**Zeit**: 20’

**Aufgabe**: Erstellen Sie ein Programm, das folgende Aufgabe löst:

Recherchieren sie im Internet das Thema BMI (BodyMassIndex). Realisieren Sie danach ein Programm, dass nach den nötigen Werten fragt und den BodymassIndex ausgibt

**Ihre Lösung (Kopieren sie am Schluss ihre Lösung hier hinein):**

|  |
| --- |
| static void Main(string[] args) {  double gewicht = 0;  double groesse = 0;  Console.Write("Körpergewicht in kg: ");  gewicht = Convert.ToDouble(Console.ReadLine());  Console.Write("Körpergrösse in m: ");  groesse = Convert.ToDouble(Console.ReadLine());  Console.WriteLine("Ihr Bodymassindex (BMI) ist: " + gewicht / (groesse \* groesse));  Console.Write("Drücken sie eine beliebige Taste zum verlassen des Programmes");  Console.ReadKey(true);  } |

**Aufgabe 06 (Auf der Basis von Eingaben neue Informationen berechnen und ausgeben)**

**Lernziele**: Die Lernenden können Eingaben vom Benutzer entgegennehmen, durch matmatische Grundoperationen (+,-,\*,/,%) verarbeiten und ein Ergebnis ausgeben

**Zeit**: 25’

**Aufgabe**: Schreiben Sie ein Programm, das den Benutzer zur Eingabe von drei ganzzahligen Werte für Stunden, Minuten und Sekunden auffordert und diese Werte in Variablen speichert. Verwenden sie für die Speicherung der Stunden den Datentyp Integer (**int**), für die Sekunden den Datentyp unsigned short (**ushort**) und für die Sekunden den Datentyp **byte.** Im Verarbeitungteil rechnet ihre App die eingegebenen Werte in das metrische System (Zehnersystem) mit der Masseinheit "Stunden" um und gibt den berechneten Wert aus.

Gibt der Benutzer zum Beispiel folgende Werte ein:

Stunden : 2

Minuten : 32

Sekunden: 50

Gibt das Programm als Resultat 2.5472222222 h aus.

**Ihre Lösung (Kopieren sie am Schluss ihre Lösung hier hinein):**

|  |
| --- |
| static void Main(string[] args)  {  } |