
02 C#-Grundlagen

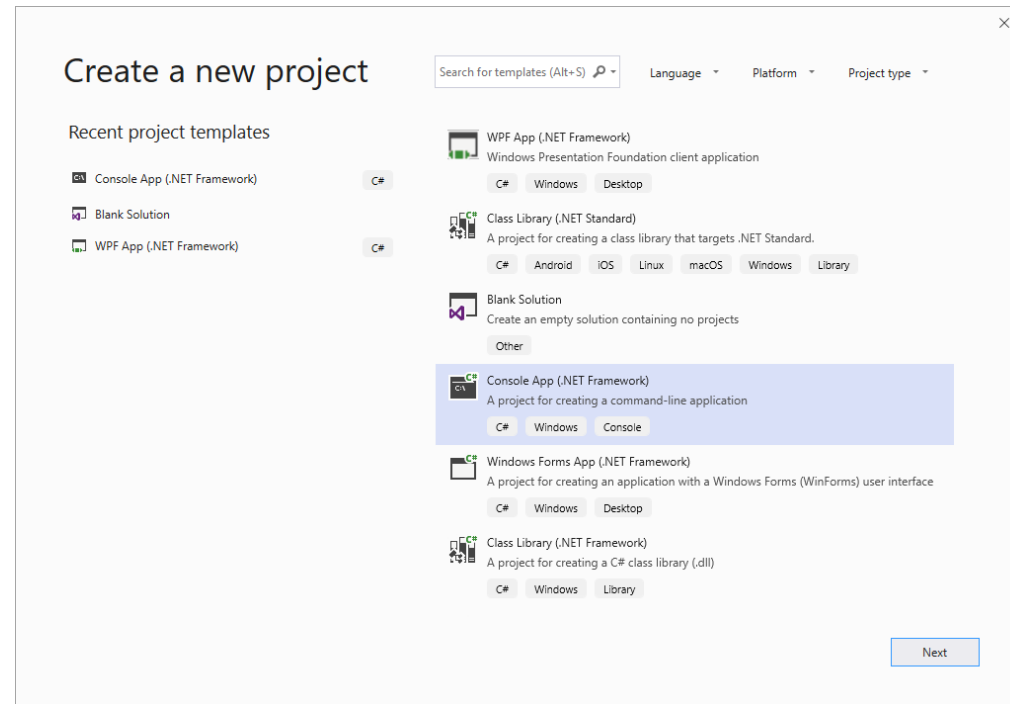
Agenda

- Konsolenanwendung in C#
 - C#-Syntax und Code-Struktur
 - Variablen, Datentypen, Konstanten
 - Strings und StringBuilder
 - Kontrollstrukturen und Programmschleifen
 - Konsolenein- und -ausgabe
 - Operatoren
 - Nullable Types
 - Datenfelder
 - Enumeratoren und Strukturen
-

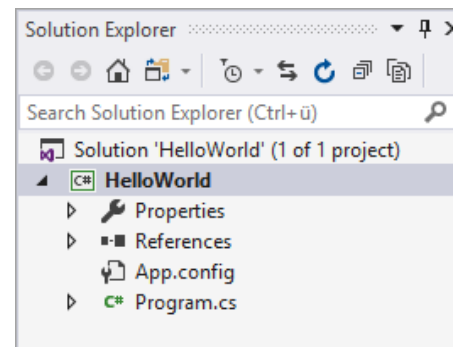
Konsolenanwendung in C#

Neues C#-Projekt:

- Menu: File → New → Project



- Resultierende Projektstruktur im Solution Explorer:



Konsolenanwendung in C#

C#-Code:

```
using System;
using System.Collections.Generic;
using System.Text;

namespace HelloWorld
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello, World!");
        }
    }
}
```

- Kompilieren und bauen: F6-Taste
 - Kompilieren, bauen, im Debugger starten: F5-Taste
-

C#-Syntax und Codestruktur

■ Anweisung abgeschlossen mit „;“

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

■ Anweisungsblöcke

```
{  
    Anweisung X;  
    {  
        Anweisung Y;  
        Anweisung Z;  
    }  
}
```

C#-Syntax und Codestruktur

■ Allgemeine Codestruktur

Codatei (Bsp. Program.cs)

using <namespace>

namespace <name>

Interface <name>

Delegate <name>

class <name>

Variablen, Eigenschaften, Konstanten, Ereignisse
(Daten speichern)

Konstruktor (Objekt erstellen/konstruieren)

Destruktor (Objekt löschen)

Methoden (Funktionalität)

C#-Syntax und Codestruktur

Kommentare (1/2)

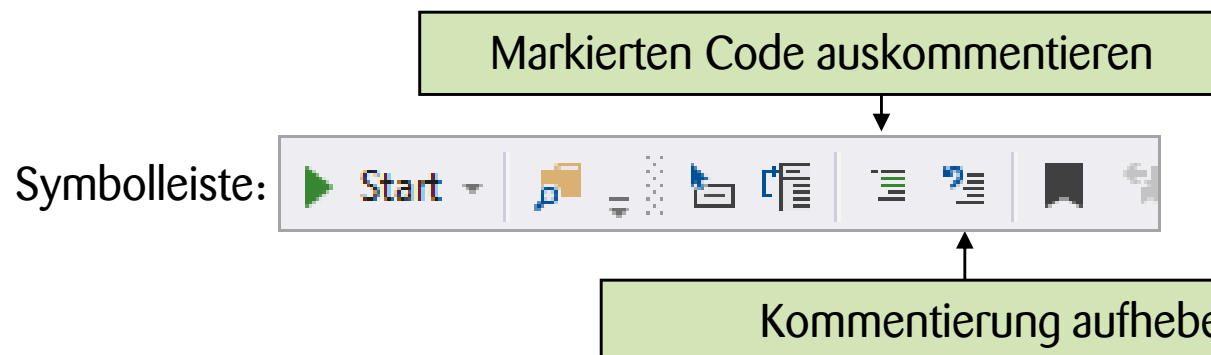
■ 3 Beispiele für Kommentare im C#-Code

```
// Das ist ein Kommentar

Console.WriteLine("Hallo"); // Dies ist auch ein Kommentar

/* Dies ist
   ein mehrzeiliger
   Kommentar
*/
```

■ Auskommentieren in Visual Studio Editor



Kommentare (2/2)

- Kommentare im C#-Code, um Dokumentation zu generieren

```
/// <summary>  
/// The main entry point for the application.  
/// </summary>
```

XML-Tags, um Textblöcke zu markieren



Variablen und Datentypen

Variablen deklarieren

■ Syntax:

```
<Datentyp> <Bezeichner>;
```

■ Bezeichner:

- Der Bezeichner muss eindeutig sein
- Gültig sind alphanumerische Zeichen und Unterstrich „_“
- Muss mit Buchstaben oder Unterstrich anfangen
- Darf nicht gleich sein wie Schlüsselwort, Name einer Prozedur, Klasse oder Objekt

■ Beispiele:

```
int a;  
int b = 20, c = 50;  
double _x, y_1, z1;
```

Variablen und Datentypen

Tabelle der Datentypen

C#-Alias	.NET-Typ	#Bytes	Wertebereich
sbyte	System.SByte	1	-128 ... 127
byte	System.Byte	1	0 ... 255
Short	System.Int16	2	$-2^{15} \dots 2^{15}-1$
ushort	System.UInt16	2	0 ... 65535
int	System.Int32	4	$-2^{31} \dots 2^{31}-1$
uint	System.UInt32	4	0 ... $2^{32}-1$
long	System.Int64	8	$-2^{63} \dots 2^{63}-1$
ulong	System.UInt64	8	0 ... $2^{64}-1$
float	System.Single	4	$1,4 * 10^{-45}$ bis $3,4 * 10^{38}$
double	System.Double	8	$5,0 * 10^{-324}$ bis $1,7 * 10^{308}$
decimal	System.Decimal	12	+/-79E27 ohne Dezimalpunkt; +/-7.9E-29, falls 28 Stellen hinter dem Dezimalpunkt angegeben werden. Die kleinste darstellbare Zahl beträgt +/-1.0E-29.
bool	System.Boolean	1	true oder false
char	System.Char	2	Unicode-Zeichen zwischen 0 und 65535
string	System.String	variabel	Ca. 2^{31} Unicode-Zeichen
object	System.Object	4	Ein Variable vom Typ Object kann jeden anderen Datentypen enthalten, ist also universell

Variablen und Datentypen

Typensuffix für Fließkommazahlen

Fließkommatyp	Suffix
float	F oder f
double (Standard)	D oder d
decimal	M oder m

Standard Datentyp für Fließkommazahlen.
Der Suffix ist nicht zwingend für double.

■ Beispiele:

```
float z1;  
double z2;  
decimal z3;  
z1 = 3.21F;  
z2 = 6543.23421D;  
z2 = 423.12312;  
z3 = 4223452.2345454323M;
```

Variablen und Datentypen

Ganze Zahlen in hexadezimaler Form zuweisen

- Prefix: 0x
- Beispiel:

```
int aHex;  
aHex = 0xFF; // Dezimal 255 zuweisen
```

Variablen und Datentypen

Implizit typisierte lokale Variable

- Statt dem Datentyp, das Schlüsselwort `var` verwenden (C# 3.0, Nov. 2007)
- Es existieren drei Fälle
 - Fall 1: `var` kann *nicht* verwendet werden, wenn der Typ unbekannt ist
 - Fall 2: `var` *kann* verwendet werden, wenn der Typ bekannt ist
 - Fall 3: `var` *muss* verwendet werden bei anonymen Typen (Modul C# 2, Hauptverwendungsfall: LINQ-Abfragen z.B. auf Objektlisten)

```
// Fall 1
var j; // var kann nicht verwendet werden
j = 2;

// Fall 2
int i1 = 1; // Explizit typisiert
var i2 = 1; // Implizit typisiert

// Fall 3
var k = new { Farbe = "rot" } // Anonymer Typ
                             // var muss verwendet werden
```

Variablen und Datentypen

Implizit typisierte lokale Variable

- Kann nur bei lokalen Variablen verwendet werden
- Entscheidungskriterium für den Kann-Fall
 - `var` verwenden, wenn man den Code verkürzen kann, ohne damit die Verständlichkeit zu verschlechtern

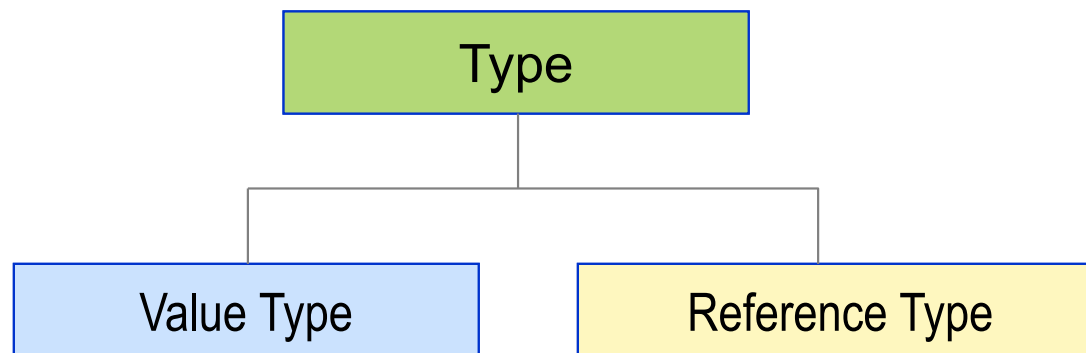
```
// Verwendung sinnvoll
MeinLangerKlassenname k = new MeinLangerKlassenname();
var k = new MeinLangerKlassenname(); // Besser, da kürzer ohne
                                         // schlechtere Lesbarkeit

// Verwendung nicht sinnvoll
Person p = HoleErstesElement();
var p = HoleErstesElement(); // Schlechter, da unklar, was
                               // der Rückgabewert der Methode ist
```

Variablen und Datentypen

Value Types und Reference Types

- Das .NET Common Type System (CTS) unterscheidet zwischen Werte- und Referenztypen
- Pointer wie in C/C++ werden nicht unterstützt



Value Types und Reference Types

■ Value (Wert) Type:

- Enthalten direkt Werte
- Operationen auf Wertetypen haben nie Seiteneffekte auf andere Variablen

■ Reference Type:

- Speichern Referenzen auf die Speicherstellen (Heap), welche die Daten enthalten
- Zwei Referenzvariablen können auf den selben Speicherbereich (Heap) verweisen
- Operationen auf einer Referenzvariablen können Auswirkungen auf eine andere Referenzvariable haben, wenn die beiden auf den selben Speicher (Heap) verweisen

Variablen und Datentypen

Gültigkeitsbereich von Variablen

- Mehrfache Definition von Variablen mit denselben Namen ist nicht zulässig

```
{  
    int i;  
    ...  
    {  
        int i;  
        ...  
    }  
}
```

Nicht zulässig → Fehler beim Kompilieren

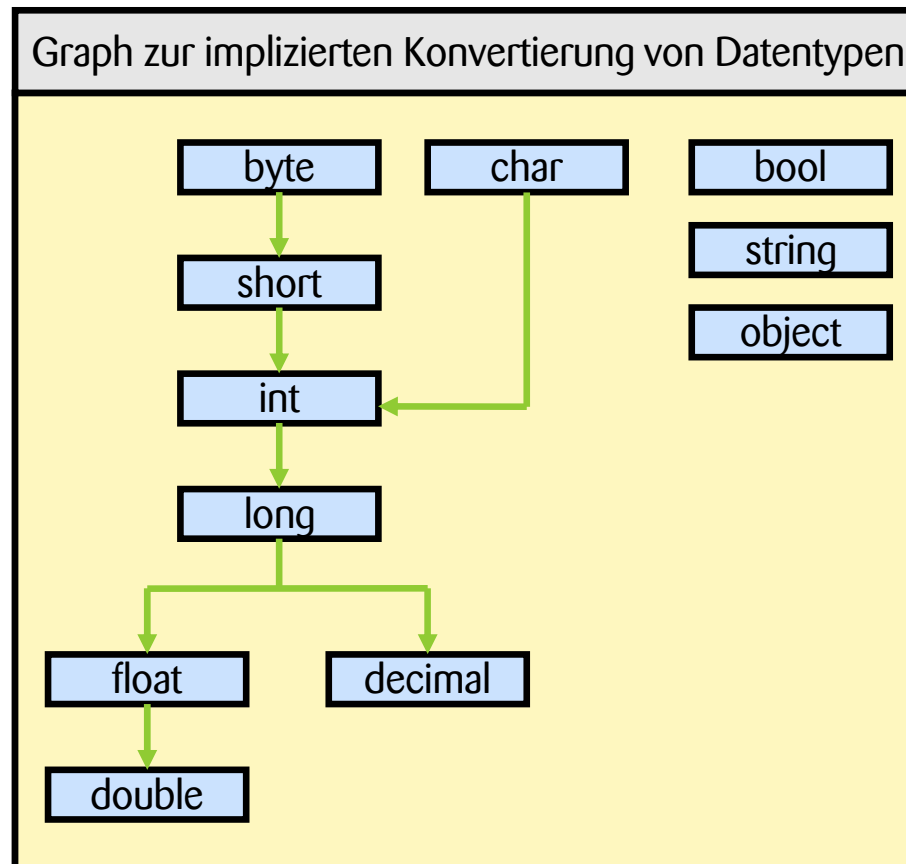
Variablen und Datentypen

Datentyp-Konvertierung (1/5)

■ Implizierte Konvertierung

■ Beispiele

```
byte b = 255;  
int i = b;  
double d = i;
```



Datentyp-Konvertierung (2/5)

- Explizierte Konvertierung
- Syntax: `(<Ziel Datentyp>) <Ausdruck>;`

- Beispiele

```
byte b = 65;  
char c = (char)b;  
  
double d = 245343.12345;  
float f = (float)d;
```

Variablen und Datentypen

Datentyp-Konvertierung (3/5)

```
int a = 12;  
string str = a.ToString();
```

- Typ x nach String mit Methode ToString ()
- String nach CTS-Datentyp x mit Methoden:
 - Parse () // Parst und konvertiert den String
// Wirft im Fehlerfall eine System.FormatException

```
string str = "200";  
int a = Int32.Parse(str);
```

- TryParse () // Versucht den String zu parsen und zu konvertieren
// Im Erfolgsfall: Gibt true und den Wert zurück
// Im Fehlerfall: Gibt false und den Wert 0 zurück

```
string str = "3927.21";  
double d;  
if (Double.TryParse(str, out d))  
    { Console.WriteLine(d); }  
else  
    { Console.WriteLine("Fehler: Falsches Format"); }
```

Datentyp-Konvertierung (4/5)

- Typ x nach Typ y mit Methoden der Klasse `Convert`

Methode	Beschreibung
<code>ToBoolean(expr)</code>	Konvertiert expr nach bool
<code>ToByte(expr)</code>	Konvertiert expr nach byte
<code>ToChar(expr)</code>	Konvertiert expr nach char
<code>ToDecimal(expr)</code>	Konvertiert expr nach decimal
<code>ToDouble(expr)</code>	Konvertiert expr nach double
<code>ToInt16(expr)</code>	Konvertiert expr nach short
<code>ToInt32(expr)</code>	Konvertiert expr nach int
<code>ToInt64(expr)</code>	Konvertiert expr nach long
<code>ToSByte(expr)</code>	Konvertiert expr nach sbyte
<code>ToSingle(expr)</code>	Konvertiert expr nach float
<code>ToString(expr)</code>	Konvertiert expr nach string
<code>ToUInt16(expr)</code>	Konvertiert expr nach ushort
<code>ToUInt32(expr)</code>	Konvertiert expr nach uint
<code>ToUInt64(expr)</code>	Konvertiert expr nach ulong

Datentyp-Konvertierung (5/5)

■ Beispiele mit der Klasse Convert

```
bool b = true;  
double dou = 245343.92345;  
  
int i = Convert.ToInt32(dou);  
  
decimal dec = Convert.ToDecimal(dou);  
  
decimal decBoo = Convert.ToDecimal(b);
```

■ Den Datentyp eines Typs / eines Aliases ermitteln:

```
System.Type type = typeof(int);
```

Konstanten deklarieren

■ Syntax:

```
Zugriffsmodifizierer const <Datentyp> <Bezeichner> = <Wert>;
```

■ Bezeichner:

- Der Bezeichner muss eindeutig sein
- Gültig sind alphanumerische Zeichen und Unterstrich „_“
- Muss mit Buchstaben oder Unterstrich anfangen
- Darf nicht gleich sein wie Schlüsselwort, Name einer Methode, Klasse oder Objekt.

■ Beispiele:

```
public const double Pi = 3.14;
```

String (1/2)

■ Allgemeines:

- Klasse `String` wird verwendet, um Zeichenketten abzubilden
- Die Klasse befindet sich im Namespace `System`
- `string` ist ein Alias für `System.String`
- Immer Datentyp `string` verwenden
- Strings sind unveränderlich, d.h. bei Neuweisung wird ein neuer String erzeugt

■ Deklaration:

```
System.String greeting = "Hello World!"; // Deklaration
string greeting2 = "alias";              // Alias

string message = "They said, \"Hello!\""; // Escape
string path = @"c:\User\someone";        // Verbatim String
string concatenation = $"{a} {b}";        // Zusammenfügung
```


String (2/2)

■ Vergleiche:

```
if ("" == string.Empty) {}  
if (string.IsNullOrEmpty(null)) {}
```

■ Einige Methoden:

```
string message = " abcde ";           // Leerzeichen vorne/hinten  
Console.WriteLine(message.Length);    // 7  
Console.WriteLine(message.Trim());    // "abcde"  
Console.WriteLine(message.ToUpper()); // " ABCDE "  
Console.WriteLine(message);           // Wieder " abcde ", da  
                                     // Strings unveränderlich  
Console.WriteLine(message.StartsWith(" ab")); // true
```

StringBuilder (1/2)

■ Allgemeines:

- Namespace: `System.Text`
- Die Klasse `StringBuilder` bietet effiziente Methoden, um Strings zusammenzusetzen
- Im Gegensatz zur Klasse `String` wird bei Änderungen nicht der ganze String neu kopiert
- Die Kapazität kann beim Instanzieren oder durch das Setzen der Eigenschaft `Capacity` definiert werden
- Die Eigenschaft `Length` enthält die Anzahl Zeichen des Strings
- Wird die Kapazitätslimite überschritten, so wird der String in einen neuen Speicherbereich kopiert und die Kapazität automatisch verdoppelt

■ Beispielcode

```
StringBuilder sb1 = new StringBuilder();
StringBuilder sb2 = new StringBuilder("Hallo");
sb2.Capacity = 50;
StringBuilder sb3 = new StringBuilder("Hallo", 50);
Console.WriteLine(sb3.Length); // Ausgabe: 5
Console.WriteLine(sb3.Capacity); // Ausgabe: 50
```

StringBuilder (2/2)

■ Strings verändern:

Methode	Beschreibung
Append	Zeichenkette am Ende hinzufügen
AppendFormat	Zeichenkette mit Formatierung am Ende hinzufügen
Insert	Fügt eine Zeichenkette beim angegebenen Index ein
Remove	Eine Anzahl Zeichen entfernen
Replace	Ersetzt ein bestimmtes Zeichen in der Zeichenkette

■ Beispielcode

```
StringBuilder sb = new StringBuilder("Programmieren ");
sb.Append("ist spannend?");
sb.Insert(0, "C#-");
sb.Replace('?', '!');
sb.Replace("spannend", "cool");
Console.WriteLine(sb); // Ausgabe: C#-Programmieren ist cool!
string text = sb.ToString(); // Konvertierung zu String
```

Kontrollstrukturen

Syntax:

```
if (Bedingung)
    Anweisung 1;
else
    Anweisung 2;
```



```
// Beispiel
int anzKarten = 36;
if (anzKarten == 36)
    Console.WriteLine("Neues Spiel");
else
    Console.WriteLine("Karten spielen");
```

Syntax mit Block:

```
if (Bedingung)
{
    Anweisung A1;
    ...
}
else
{
    Anweisung B1;
    ...
}
```



```
// Beispiel
int anzKarten = 36;
if (anzKarten == 36)
{
    Console.WriteLine("Neues Spiel");
}
else
{
    Console.WriteLine("Karten spielen");
}
```

Kontrollstrukturen

Syntax:

```
if (Bedingung 1)
    Anweisung 1;
else if (Bedingung 2)
    Anweisung 2;
else
    Anweisung 3;
```



```
// Beispiel
int anzKarten = 36;
if (anzKarten == 36)
{
    Console.WriteLine("Neues Spiel");
    ...
}
else if (anzKarten == 0)
{
    Console.WriteLine("Spielende");
    ...
}
else
{
    Console.WriteLine("Karte spielen");
    ...
}
```

Syntax:

```
switch (Ausdruck)
{
    case Ausdruck1 :
        Anweisungen;
        break;
    case Ausdruck2 :
        Anweisungen;
        break;
    default :
        Anweisungen;
        break;
}
```

```
// Beispiel
int anzKarten = 36;
switch (anzKarten)
{
    case 36:
        Console.WriteLine("Neues Spiel");
        break;
    case 0:
        Console.WriteLine("Spielende");
        break;
    default:
        Console.WriteLine("Karten spielen");
        break;
}
```

Programmschleifen

While-do-Schleife, Syntax:

```
while (Abbruchbedingung)
{
    Anweisungen;
}
```

Do-while-Schleife, Syntax:

```
do
{
    Anweisungen;
}
while (Abbruchbedingung);
```

Programmschleifen

For-Schleife, Syntax:

```
for (Zähler = Anfangswert; Abbruchbedingung; Zähler = NeuerWert)
{
    Anweisungen;
}
```

Foreach-Schleife, Syntax:

```
foreach (Element in Array | Collection)
{
    Anweisungen;
}
```


Konsolenein- und -ausgabe

Eingabe

■ Klasse Console

- Ein Zeichen lesen: `Console.Read()`
- Zeichenzeile lesen, bis <Enter>: `Console.ReadLine()`

Allgemeiner
Ablauf

1. String einlesen



2. Eingabe validieren



3. Eingabe konvertieren
und speichern

Beispielcode:

```
int aInt;  
string inputString;  
  
1 inputString = Console.ReadLine();  
2 3 if (Int32.TryParse(inputString, out aInt))  
   {  
       Console.WriteLine(aInt);  
   }
```

Konsolenein- und -ausgabe

Ausgabe

■ Klasse Console

- Zeichenausgabe: `Console.Write()`
- Zeichenausgabe mit Zeilenumbruch: `Console.WriteLine()`

Allgemeiner
Ablauf

1. Daten nach String
konvertieren



2. Ausgabestring
formatieren



3. Formatierten String
ausgeben

Code Beispiel:

```
int aHex;  
aHex = 0xFF; // Dezimal 255 zuweisen  
  
Console.WriteLine($"Zahl = {aHex}");  
①②③ // String-Interpolation
```

Konsolenein- und -ausgabe

Ausgabe formatieren

- **Methode:** `String.Format(<Formatierungsausdruck>, Parameter...)`
- Zur Formatierung der Ausgabe kann ein Formatierungsausdruck in geschweiften Klammern verwendet werden
- **Syntax:** `{N [,M] [: Format]}`
 - N = Parameter-Index beginnend mit 0
 - M = Breite der Ausgabe (optional)
 - : Format = Formatangabe (optional)

- **Beispiel**

```
int zeit = 5;  
Console.WriteLine($"Es ist {zeit,2} Uhr.");  
string text = String.Format($"Es ist {zeit,2} Uhr.");
```

Ausgabe:

Es ist 5 Uhr.

2 Zeichen breit

Konsolenein- und -ausgabe

Formatangaben

Formatangabe	Anzeige
C	Zahl im lokalen Währungsformat
D	Zahl als Integer
E	Zahl im wissenschaftlichen Format
F	Festpunktformat
G	Numerische Zahl in kompakteren Form aus Festpunkt- oder wissenschaftlichem Format
N	Zahl inkl. Kommaseparator
P	Zahl in Prozent
X	Zahl im hexadezimalen Format

Konsolenein- und -ausgabe

Beispiele mit Formatangabe

```
double zahl = 1203.5;

Console.WriteLine($"Zahl = {zahl:C}");
// Ausgabe: Zahl = SFr. 1'203.50

Console.WriteLine($"Zahl = {zahl:E}");
// Ausgabe: Zahl = 1.203500E+003

Console.WriteLine($"Zahl = {zahl:E2}");
// Ausgabe: Zahl = 1.20E+003

Console.WriteLine($"Zahl = {zahl:F2}");
// Ausgabe: Zahl = 1203.50

Console.WriteLine($"Zahl = {zahl:P2}");
// Ausgabe: Zahl = 120'350.00%

int iHex = 254;

Console.WriteLine("Zahl = {0:X}", iHex);
// Ausgabe: Zahl = FE
```

Escape-Zeichen

- Escape-Zeichen repräsentieren Sonderzeichen in Strings resp. bei der Textausgabe

Escape-Zeichen	Beschreibung / Representierung
\‘	Hochkomma
\“	Anführungsstriche
\\	Backslash
\a	Alarmton ausführen
\b	Ein Zeichen zurück (Backspace)
\f	Formularvorschub (nur für Drucker)
\n	Zeilenvorschub
\r	Wagenrücklauf (wie Home-Taste)
\t	Tabulator
\u	Fügt ein Unicode-Zeichen ein
\v	Vertikaltabulator (nicht für Konsole geeignet)

Operatoren

Unterscheidung: Unäre und binäre Operatoren

Unärer Operator:

- Ein Argument
- Ein Resultat

- Beispiele:

```
negativesResultat = -verlust  
istFalsch = !true
```

Binärer Operator:

- Zwei Argumente
- Ein Resultat

- Beispiele:

```
summe = summand1 + summand2  
resultat = isReadOnly == true
```

Operatoren

Arithmetische Operatoren

Operator	Beschreibung	Beispiele
+	Addiert zwei Operanden Positive Zahl	$2 + 4$ $+4.5$
-	Subtrahiert zwei Operanden Negative Zahl	$6 - 4$ -9
*	Multipliziert zwei Operanden	$6 * 4$
/	Dividiert zwei Operanden	$10 / 2$
%	Liefert den Restwert der Division zweier Operanden (Modulo)	$7 \% 2$ ergibt 1
++	Erhöht den Inhalt des Operanden um 1	$a++$
--	Verringert den Inhalt des Operanden um 1	$a--$

Übung 2.1



Hexadezimalrechner

- Schreibe eine Konsolenapplikation, um einfache Rechenoperationen mit hexadezimalen Zahlen durchzuführen.
- A) Die Operanden sollen in hexadezimaler Form ein- und ausgegeben werden.
- B) Es sollen einfache Operationen wie Addition, Subtraktion, Multiplikation und Division möglich sein.

Vergleichsoperatoren

Operator	Beschreibung
$a == b$	Ergebnis ist true, wenn a gleich b ist
$a != b$	Ergebnis ist true, wenn a ungleich b ist
$a < b$	Ergebnis ist true, wenn a kleiner b ist
$a > b$	Ergebnis ist true, wenn a grösser b ist
$a \leq b$	Ergebnis ist true, wenn a kleiner gleich b ist
$a \geq b$	Ergebnis ist true, wenn a grösser gleich b ist

Logische Operatoren

Operator	Beschreibung	Beispiele
!	Negationsoperator	!a ist true, wenn a false ist
&&	Logischer AND-Operator	a && b ist true, wenn beide true sind
	Logischer OR-Operator	a b ist true, wenn a oder b true ist
^	Logischer XOR-Operator	a ^ b ist true, wenn nur einer von a oder b true ist

Bitweise Operatoren

Operator	Beschreibung	Beispiele
~	Invertiert jedes Bit	~ 8 (0x00000008) = 0xffffffff7
	Aus $x y$ resultiert ein Wert, bei dem die korrespondierenden Bits von x und y OR-verknüpft werden.	5 7 (= 7)
&	Aus $x \& y$ resultiert ein Wert, bei dem die korrespondierenden Bits von x und y AND-verknüpft werden.	3 & 6 (= 2)
^	Aus $x \wedge y$ resultiert ein Wert, bei dem die korrespondierenden Bits von x und y XOR-verknüpft werden.	4 ^ 5 (= 1)
<<	Aus $x \ll y$ resultiert ein Wert, der durch die Verschiebung der Bits des ersten Operanden x um die durch im zweiten Operanden y angegebene Zahl nach links entsteht.	8 << 2 (= 32)
>>	Aus $x \gg y$ resultiert ein Wert, der durch die Verschiebung der Bits des ersten Operanden x um die durch im zweiten Operanden y angegebene Zahl nach rechts entsteht.	8 >> 1 (= 4)

Zuweisungsoperatoren

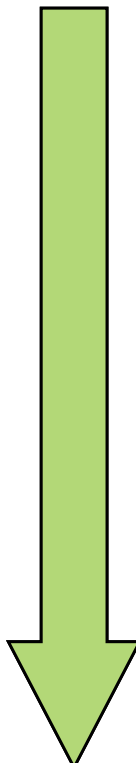
Operator	Beschreibung / Beispiele
=	$x = y$ weist x den Wert von y zu.
+=	$x += y$ weist x den Wert von $x + y$ zu.
-=	$x -= y$ weist x den Wert von $x - y$ zu.
*=	$x *= y$ weist x den Wert von $x * y$ zu.
/=	$x /= y$ weist x den Wert von x / y zu.
%=	$x \% = y$ weist x den Wert von $x \% y$ zu.
&=	$x \& = y$ weist x den Wert von $x \& y$ zu.
=	$x = y$ weist x den Wert von $x y$ zu.
^=	$x \wedge = y$ weist x den Wert von $x \wedge y$ zu.
<<=	$x \ll = y$ weist x den Wert von $x \ll y$ zu.
=>>	$x \gg = y$ weist x den Wert von $x \gg y$ zu.

Sonstige Operatoren

Operator	Beschreibung	Beispiele
.	Zugriff auf die Eigenschaften oder Methoden einer Klasse verwendet	Math.Sin(x)
[]	Wird für Arrays, Indexer und Attribute verwendet	myArray[2]
()	Reihenfolge der Operatoren festlegen oder für Typenkonvertierung verwendet	(2 + 3) * 5 (int) 3.54
?:	Einfache Variante der if-Bedingungsprüfung	s = (i>0) ? "positiv" : "negativ"
new	Instanziierung einer Klasse	var p = new Person()
is	Prüft den Laufzeittyp eines Objekts mit einem angegebenen Typ	if (s is StringBuilder)
typeof	Ruft das «System.Type»-Objekt für einen Typ ab	type = typeof(int)
checked/ unchecked	Steuert die Reaktion der Laufzeit bei einem arithmetischen Überlauf	checked { ... }

Operatoren

Reihenfolge der Auswertung

Gruppe	Operator	Stark Zuerst berücksichtigt
1	x.y (Punktoperator), a[x], x++, x--, new, typeof, checked, unchecked	
2	+ (unär), - (unär), !, ~, ++x, --x, (<Typ>)x	
3	*, /, %	
4	+ (additiv), - (additiv)	
5	<<, >>	
6	<, >, <=, >=, is	
7	==, !=	
8	&	
9	^	
10		
11	&&	
12		
13	?:	
14	=, *=, /=, %=, +=, -=, <<=, >>=, &=, ^=, =	Schwach Zuletzt berücksichtigt

Operatoren

Geprüfte und ungeprüfte Ausdrücke

Geprüfter Ausdruck:

- Überlauf wird detektiert
- OverflowException
- Performance-Einbusse

Ungeprüfter Ausdruck:

- Überlauf nicht detektiert
- Performanter
- Default für Release

```
checked
{
    byte a = 55;
    byte b = 210;
    byte c = (byte) (a+b);
}
```

```
unchecked
{
    byte a = 55;
    byte b = 210;
    byte c = (byte) (a+b);
}
```


Übung 2.2



Überlauf / Overflow

- Schreibe ein Programm, um 2 Zahlen im Zahlenbereich $[0, 255]$ einzugeben und deren Summe wieder auf der Konsole auszugeben.
- A) Benutze den Datentyp `byte`, um die Daten zu speichern.
- B) Untersuche, was geschieht, wenn die Summe kleiner oder grösser als 255 ergibt.
- C) Wie kann der Überlauf detektiert werden?

Nullable Types

Nullable Types

- Kann der Wert `null` zugewiesen werden
=> sie haben keinen Wert zugewiesen
- Deklaration mit `<Datentyp>?`
- Nullable Types sind Instanzen von `System.Nullable<T>`
- Beispiele:

```
int? number = null;  
int? counter = 10;  
double? amount = null;  
bool? editMode = true;  
bool? isGoodChessPlayer = null;  
char? letter = 'z';  
int?[] lottoNumbers = new int?[6];
```

Nullable Types

Nullable Types Members

■ HasValue

- Ist vom Typ bool und ist true, wenn ein Wert zugeordnet ist

■ Value

- Enthält den zugewiesenen Wert, wenn nicht null; wenn HasValue true ist
- Wirft eine `InvalidOperationException`, falls darauf zugegriffen wird und kein Wert zugewiesen ist; wenn HasValue false ist

■ Beispielcode:

```
int? number = 10;
if (number.HasValue)
{
    System.Console.WriteLine(number.Value);
}
else
{
    System.Console.WriteLine("nicht definiert");
}
```

Nullable Types

Auf zugewiesenen Wert testen

- Beispielcode: Testen, ob Betrag in CHF vorhanden ist

```
double? amountInSwissFrancs = null;
double exchangeRateChfToEuro = 1.108;
double amountInEuro = 0;

if (amountInSwissFrancs.HasValue == true) // Testen ob Wert zugewiesen ist
{
    amountInEuro = amountInSwissFrancs.Value * exchangeRateChfToEuro;
}
else
{
    amountInEuro = 0;
}
Console.WriteLine("Betrag in Euro: {0}", amountInEuro);
```

- Oder mit Vergleichsoperatoren != und ==

```
if (amountInSwissFrancs != null)
```

```
if (amountInSwissFrancs == null)
```

Nullable Types

Der «Null coalescing»-Operator ??

to coalesce = verschmelzen

- Der Operator ?? weist einen Standardwert zu, falls der Wert einer Nullable Type Variable gleich `null` ist.

```
int? dbItemId = null; int id = dbItemId ?? -1;
```

Standardwert, falls der Wert gleich `null` ist

- Beispielcode: Testen, ob Betrag vorhanden ist

```
double? amountInSwissFrancs1 = null;
double? amountInSwissFrancs2 = 100;
double exchangeRateChfToEuro = 1.108;
double amountInEuro = 0;

amountInEuro = (amountInSwissFrancs1 ?? 0) * exchangeRateChfToEuro;
Console.WriteLine("Betrag in Euro: {0}", amountInEuro);
amountInEuro = (amountInSwissFrancs2 ?? 0) * exchangeRateChfToEuro;
Console.WriteLine("Betrag in Euro: {0}", amountInEuro);
```

Console Output:
Betrag in Euro: 0
Betrag in Euro: 110.8

Nullable Types

Der «Safe navigation»-Operator ? .

- Der Operator ? . weist «null» zu, falls der überprüfte Ausdruck null ist.
- Beispiel: Nicht instanziierte Arrays

```
int[] zahlen = null;  
int? length = zahlen?.Length;  
int? zahl1 = zahlen?[0];  
int? zahl2 = zahlen?[1];
```

Weist null zu, da das Array null ist

- Beispiel: Instanziiertes Arrays mit 2 Werten

```
int[] zahlen = new int[2];  
int? length = zahlen?.Length;  
int? zahl1 = zahlen?[0];  
int? zahl2 = zahlen?[1];
```

Weist die Array Länge 2 zu

Weist den Wert 0 zu

Nullable Types

Der «Safe navigation»-Operator ? .

- Beispiel: Falls null, null zurückgeben, ansonsten Rückgabewert der Methode.

```
string eingabe = null;  
string text = eingabe?.ToUpper();
```

← Weist text = null zu, da «eingabe» null ist

```
string eingabe = "Hallo";  
string text = eingabe?.ToUpper();
```

← Weist text = «HALLO» zu

Bedingungsoperator

Der Bedingungsoperator ? : (ternary operator, Elvis-Operator)

■ Beispiel:

- Rabatt = 10% wenn Bestellsumme > 100
- Ansonsten Rabatt = 2%

```
int a = 90;  
int b = 20;  
  
int discountPercent = a + b > 100 ? 10 : 2;
```

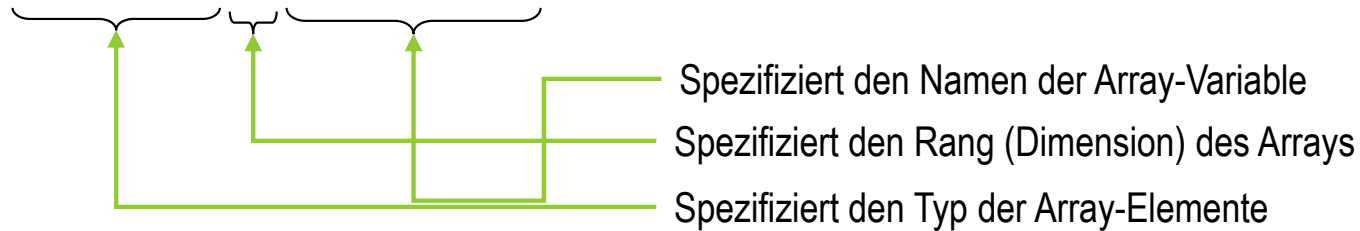
Wenn $a + b > 100 \Rightarrow 10$

Ansonsten $\Rightarrow 2$

Array deklarieren

■ Syntax:

```
<Datentyp> [ ] <Bezeichner>;
```



■ Beispiel:

```
int[] intArray;
```

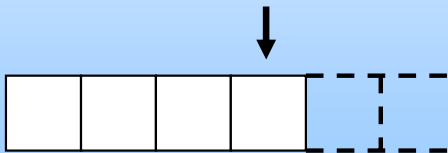
- Das Array ist deklariert, aber noch nicht initialisiert. D.h. es wurde noch kein Speicher für den Inhalt der Werte reserviert.

Rang eines Arrays

- Der Rang wird auch als Array-Dimension bezeichnet
- Die Anzahl Indizes, welche mit jedem Element assoziiert werden

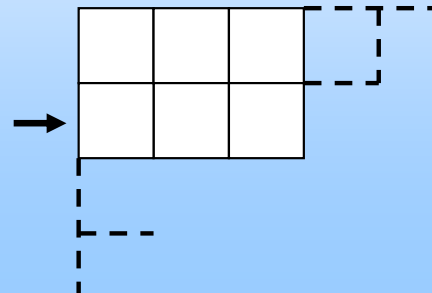
```
long[ ] row;
```

Rang 1: Eindimensional
Ein Index assoziiert mit jedem
long-Element



```
int[,] grid;
```

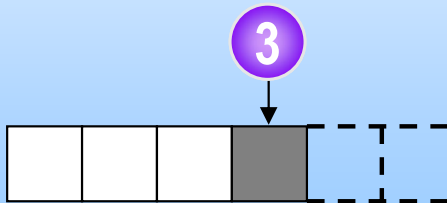
Rang 2: Zweidimensional
Zwei Indizes assoziieren mit
jedem **int**-Element



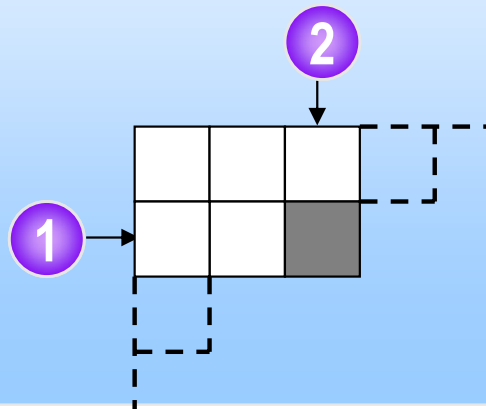
Zugriff auf die Elemente des Arrays

- Für jeden Rang muss ein Integer-Index angegeben werden
- Indizes sind nullbasiert

```
long[ ] row;  
...  
row[3] = 12;
```



```
int[,] grid;  
...  
grid[1,2] = 27;
```

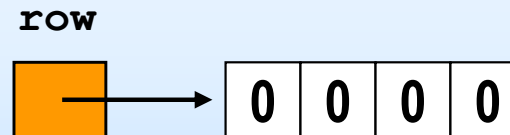


Initialisierung des Arrays

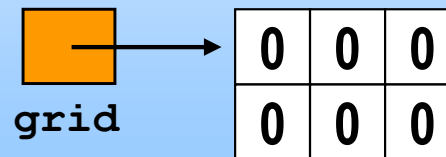
- Arrays müssen explizit mit `new` erzeugt werden
- Alle Array-Elemente werden standardmässig mit Null initialisiert



```
long[ ] row = new long[4];
```



```
int[,] grid = new int[2,3];
```



Initialisierung von Array-Elementen

- Die Elemente eines Arrays können explizit initialisiert werden
- Es gibt eine praktische Kurzschreibweise

```
long[ ] row = new long[4] {0, 1, 2, 3};
```

```
long[ ] row = {0, 1, 2, 3};
```

← Equivalent ↑

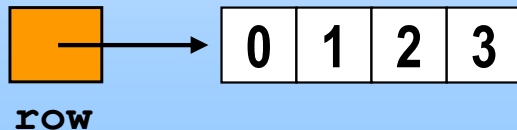


Abbildung im Speicher

```
int[] intArray = {23, 7, 90, 34, 88};
```

Speicher

Adresse	Inhalt
4006	*
4005	intArray[4] = 88
4004	intArray[3] = 34
4003	intArray[2] = 90
4001	intArray[1] = 7
4000	intArray[0] = 23
3999	*

intArray[] → Startadresse

Reservierter Speicherbereich für das Array intArray

Initialisierung multidimensionaler Array-Elemente

- Alle Elemente müssen spezifiziert werden

```
int[, ] grid = {  
    {5, 4, 3},  
    {2, 1, 0}  
};
```

```
int[, ] grid = {  
    {5, 4, 3},  
    {2, 1 }  
};
```

← Implizit ein neues int[2,3] Array



grid

5	4	3
2	1	0



Fehler: Element [2,3] fehlt!

Anzahl Elemente ermitteln

- Gesamtzahl der Elemente: `myArray.Length;`
- Elemente einer Dimension: `myArray.GetLength(<dim>);`
- Beispiele:

```
int[] arr = new int[5];  
int[,] arr2Dim = new int[3, 4];  
  
Console.WriteLine(arr.GetLength(0));           // Ausgabe: 5  
Console.WriteLine(arr.Length);                 // Ausgabe: 5  
Console.WriteLine(arr2Dim.GetLength(0));       // Ausgabe: 3  
Console.WriteLine(arr2Dim.GetLength(1));       // Ausgabe: 4  
Console.WriteLine(arr2Dim.Length);             // Ausgabe: 12
```


Übung 2.3:

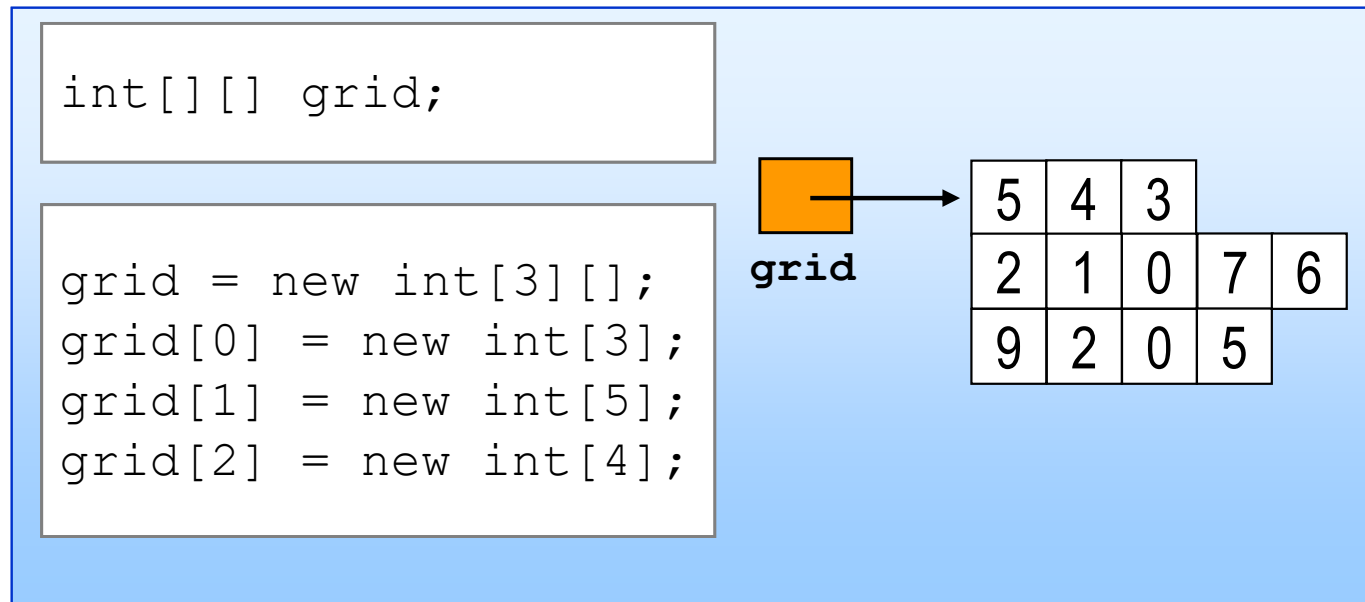


Mit Arrays arbeiten

- Schreibe ein Programm, welches 10 Zahlen von der Tastatur einliest und diese in ein Array ablegt.
- Mit den eingegebenen Werten sollen die folgenden drei Teilaufgaben gelöst werden:
 - a) Den Durchschnitt ausgeben
 - b) Das Minimum und Maximum ausgeben
 - c) Die Zahlen aufsteigend sortiert ausgeben

Speicher-optimierte Jagged Arrays (Verzweigte Datenfelder)

- Jagged Arrays haben unterschiedlich lange Datenfelder
- Jagged Arrays können optimiert im Speicher abgelegt werden



Zugriff mit foreach

- Das foreach-Anweisung vereinfacht die Bearbeitung von Arrays, wenn alle Elemente des Arrays berücksichtigt werden sollen

```
// Gibt alle Kommandozeilen-Argumente aus

class Beispiel {

    static void Main(string[] args) {
        foreach (string arg in args) {
            System.Console.WriteLine(arg);
        }
    }
}
```

Die Klasse `Array`

- Die Klasse `Array` stellt diverse Methoden zur Verfügung, um Arrays zu erstellen, verändern, suchen und sortieren
- Beispiel: `Sort()`

```
int[] arr = {32, 16, 8, 4, 2, 1};  
  
Array.Sort(arr);  
// Sortiert das Array um nach: [1, 2, 4, 8, 16, 32]
```

Übung 2.4:



Sieb des Eratosthenes

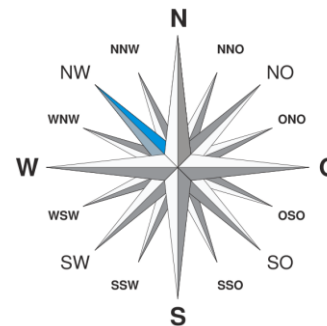
- Schreibe ein Programm, welches Primzahlen mit dem Prinzip des “Sieb des Eratosthenes” ermittelt
- Die Beschreibung des Algorithmus ist auf dem separaten Blatt ersichtlich, bzw. siehe Animation auf Wikipedia-Seite:

https://en.wikipedia.org/wiki/Sieve_of_Eratosthenes

	2	3	4	5	6	7	8	9	10	Prime numbers
11	12	13	14	15	16	17	18	19	20	
21	22	23	24	25	26	27	28	29	30	
31	32	33	34	35	36	37	38	39	40	
41	42	43	44	45	46	47	48	49	50	
51	52	53	54	55	56	57	58	59	60	
61	62	63	64	65	66	67	68	69	70	
71	72	73	74	75	76	77	78	79	80	
81	82	83	84	85	86	87	88	89	90	
91	92	93	94	95	96	97	98	99	100	
101	102	103	104	105	106	107	108	109	110	
111	112	113	114	115	116	117	118	119	120	

Enum

- Enumeratoren sind stark typisierte Konstanten – d.h. ein Enum von einem Typ kann nicht einem anderen Enum-Typ implizit zugewiesen werden
- Erlauben symbolische Namen für ganzzahlige Werte
- Standardmässig beginnend mit 0; Werte können zugewiesen werden
- Code wird wartbarer
- Beispiel: Statt die Himmelsrichtungen mit 1, 2, 3, 4 zu bezeichnen, kann man mit Enum die symbolischen Namen Nord, Ost, Süd, West verwenden



Enumeratoren

Enum deklarieren

- Beispiel: HiFi-Volumen in drei Stufen (Low, Medium, High)

```
// Deklariert einen enum
public enum Volume
{
    Low,
    Medium,
    High
}
```

- Werte nach obiger Deklaration:
Low = 0, Medium = 1, High = 2

Enum – Anwendung

■ Beispiel: Entscheidungen treffen anhand des Enum-Wertes

```
class EnumSwitch
{
    static void Main()
    { // Eine neue Enum Instanz erstellen und initialisieren
        Volume myVolume = Volume.Medium;
        // Entscheidung auf Basis des Enum Wertes
        switch (myVolume)
        {
            case Volume.Low:
                Console.WriteLine("The volume has been turned Down.");
                break;
            case Volume.Medium:
                Console.WriteLine("The volume is in the middle.");
                break;
            case Volume.High:
                Console.WriteLine("The volume has been turned up.");
                break;
        }
    }
}
```


Enum – Werte konvertieren

■ Beispiel: Benutzereingabe nach Enum konvertieren

```
Console.WriteLine("Volume Settings:");
Console.WriteLine("Please select (0-Low, 1-Medium, or 2-High): ");

// Wert von der Console als string lesen
string volString = Console.ReadLine();
// Wert in int umwandeln
int volInt = Int32.Parse(volString);

// Explizite Konvertierung von int nach Volume enum type
Volume myVolume = (Volume)volInt;

// Entscheidung treffen Aufgrund der Benutzer Eingabe
switch (myVolume)
{
    case Volume.Low:
        ...
}
```

Enum – Werte und Namen

- Beispiel: Enum-Werte und -Namen auf die Konsole ausgeben

```
// Alle Werte und Namen abfragen und ausgeben
foreach (int val in Enum.GetValues(typeof(Volume)))
{
    Console.WriteLine("Volume Value: {0}\n Member: {1}",
        val, Enum.GetName(typeof(Volume), val));
}
```

DateTime (1/2)

■ Allgemeines:

- DateTime wird verwendet, um einen Zeitpunkt abzubilden
- DateTime befindet sich im Namespace System

■ Deklaration:

```
DateTime date1 = new DateTime(2013, 9, 16, 17, 30, 52);  
DateTime date2 = DateTime.Now;
```

■ Ausgabenformatierung:

```
DateTime.Now.ToString("MM dd, yyyy"); // 09.10.2013  
DateTime.Now.ToString("MMM dd, yyyy"); // Sep 10, 2013  
DateTime.Now.ToString("MMMM dd, yyyy"); // September 10, 2013
```

DateTime (2/2)

■ Einige Methoden:

```
// Addition
DateTime now = DateTime.Now;
Console.WriteLine(now.AddDays(1));
Console.WriteLine(now.AddMonths(2));

// Parsen
DateTime input1 = DateTime.Parse("16.09.2013");
DateTime input2 = DateTime.Parse("2013-09-16");

// Zeitspanne
DateTime startDate = new DateTime(2013, 9, 16);
DateTime endDate = new DateTime(2013, 9, 20);
TimeSpan elapsed = endDate.Subtract(startDate);
double daysAgo = elapsed.TotalDays;
Console.WriteLine(daysAgo);    // Ausgabe: 4
```

Structs (1/2)

■ Allgemeines:

- Ein Struct (Struktur) wird verwendet, um eine zusammen gehörende Variablen zu kapseln, z.B. Merkmale eines Buches
- Structs sind Werttypen (Klassen sind Referenztypen)
- Structs können ohne den Operator `new` instanziiert werden
- Structs können Konstruktoren deklarieren die über Parameter verfügen

■ Deklaration:

```
public struct Book
{
    public string Title;
    public string Author;
    public decimal Price;
    public string ISDN;
}
```

Structs (2/2)

■ Code Beispiel:

– Deklaration

```
public struct Position
{
    public double X;
    public double Y;

    public Position(double x, double y)
    {
        X = x;
        Y = y;
    }
}
```

– Instanziierung

```
Position p1 = new Position(3.4, 87.3);
Position p2;
p2.X = 32.5;
p2.Y = 872.32;
```

Übung 2.5:



Enum – Wochentage

1. Definiere einen Enum für die 7 Wochentage
2. Schreibe ein Programm, welches die Wochentage auf die Konsole schreibt.
3. Erweitere das Programm, so dass die Wochentage auf der Konsole eingegeben werden können (z.B. «Montag»). Konvertiere die Eingabe nach Enum «Wochentage».
4. Erweitere das Programm mit einem Switch-Case, damit der eingegebene Wochentag ausgegeben wird als Wochentagname (Enum zu String konvertieren), der Enum-Wert und der Wochentagname in Englisch.

