

# Datentyp String

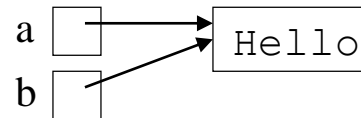
```
string a, b;
```

Bibliothekstyp für Zeichenarrays

```
a = "Hello";
```

Stringkonstante (unter doppelten Hochkommas)

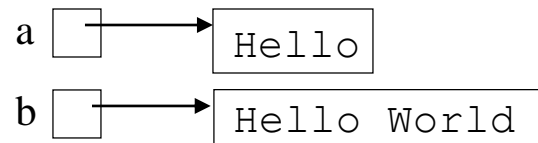
```
b = a;
```



Stringvariablen sind Zeiger auf Stringobjekte  
Stringzuweisung ist eine Zeigerzuweisung  
Stringobjekte sind nicht veränderbar:

nicht erlaubt: `a[4]='5';`

```
b = a + " World";
```



Verkettung mit "+" erzeugt neues Stringobjekt  
(relativ teure Operation)

# Stringvergleiche

```
string s = Console.ReadLine(); // liest ein Wort, z.B. Hello
```

```
if (s == "Hello") ...           // liefert true in C# (in Java kann String so nicht verglichen werden)
```

Weitere Möglichkeit zum Stringvergleich mittels Equals  
(in Java nur so möglich):

```
if (s.Equals("Hello")) ...      // liefert true! (Wertvergleich)  
                                // Diese Variante sollte immer verwendet werden!
```

# Stringoperationen

```
string s = "a long string";
```

0	1	2	3	4	5	6	7	8	9	10	11	12
a		l	o	n	g		S	t	r	i	n	g

```
int len = s.Length;
```

// liefert Anzahl der Zeichen in s (hier 13)

// nach Length keine Klammern → da Length eine Property ist (2.JG)

```
char ch = s[3];
```

// liefert das Zeichen mit Index 3 (hier 'o')

# Umwandeln Zeichen → Zahl

```
string s = "a2b3";  
char ch = s[1];
```

// liefert das Zeichen mit Index 1 und speichert dieses in die char-Variable ch  
(hier '2')  
// intern wird in char der numerische Unicode-Wert des Zeichens gespeichert  
(hier 50)

```
int digit = ch - '0';
```

// zur Umwandlung in die entsprechende Ziffer muss der Unicode-Wert der Ziffer  
'0' abgezogen werden. (Hier: 50-48 = 2)

# Zeichenkodierung: ASCII

- American Standard Code for Information Interchange

- 1. Version 1963
- Aktuelle Version:  
ANSI X3.4-1986

!"#\$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNO PQRSTUVWXYZ[\]^_
`abcdefghijklmnopqrstuvwxyz{ }~

- 7-Bit Kodierung =  $2^7 = 128$  Zeichen

- [0 .. 127] bzw. [0x00 .. 0x7F]
- [0x00 .. 0x1F]: 32 Steuerzeichen (nicht druckbar)
- [0x20 .. 0x7E]: 95 druckbare Zeichen: Englische Buchstaben, arabische Ziffern, einige Satzzeichen => entspricht einer englischen Tastaturbelegung / Schreibmaschine
- [0x7F]: Steuerzeichen „DEL“ (nicht druckbar)

# ASCII: Steuerzeichen

Dez.	Hex.	C / Java	ISO	Bedeutung
0	0x00	\0	NUL	Zeichenkettenterminator in C
7	0x07	\a	BEL	Tonsignal
8	0x08	\b	BS	Backspace
9	0x09	\t	HT	Horizontal Tab
10	0x0A	\n	LF	Line Feed (Zeilenende in Unix/MacOSX)
11	0x0B	\v	VT	Vertical Tab
12	0x0C	\f	FF	Form Feed (Seitenvorschub)
13	0x0D	\r	CR	Carriage Return (CR+LF = Zeilenende in DOS/Windows)
127	0x7F		DEL	Delete

# ASCII: Druckbare Zeichen

\V/  
- (@ @) -  
--oOO-- ( \_ ) --OOo--

Dez.	Bedeutung
65 – 90	Lateinische Großbuchstaben
97 – 122	Lateinische Kleinbuchstaben
48 - 57	Arabische Ziffern
32	Space (Blank, Leerzeichen)
Rest	Symbole (Satzzeichen, Klammern, Operatoren, ...)

!"#\$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOQRSTUVWXYZ[\]^_
`abcdefghijklmnopqrstuvwxyz{ }~

# ASCII-Tabelle

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	<b>NUL</b> (null)	32	20	040	&#32;	<b>Space</b>	64	40	100	&#64;	<b>@</b>	96	60	140	&#96;	<b>`</b>
1	1	001	<b>SOH</b> (start of heading)	33	21	041	&#33;	<b>!</b>	65	41	101	&#65;	<b>A</b>	97	61	141	&#97;	<b>a</b>
2	2	002	<b>STX</b> (start of text)	34	22	042	&#34;	<b>"</b>	66	42	102	&#66;	<b>B</b>	98	62	142	&#98;	<b>b</b>
3	3	003	<b>ETX</b> (end of text)	35	23	043	&#35;	<b>#</b>	67	43	103	&#67;	<b>C</b>	99	63	143	&#99;	<b>c</b>
4	4	004	<b>EOT</b> (end of transmission)	36	24	044	&#36;	<b>\$</b>	68	44	104	&#68;	<b>D</b>	100	64	144	&#100;	<b>d</b>
5	5	005	<b>ENQ</b> (enquiry)	37	25	045	&#37;	<b>%</b>	69	45	105	&#69;	<b>E</b>	101	65	145	&#101;	<b>e</b>
6	6	006	<b>ACK</b> (acknowledge)	38	26	046	&#38;	<b>&amp;</b>	70	46	106	&#70;	<b>F</b>	102	66	146	&#102;	<b>f</b>
7	7	007	<b>BEL</b> (bell)	39	27	047	&#39;	<b>'</b>	71	47	107	&#71;	<b>G</b>	103	67	147	&#103;	<b>g</b>
8	8	010	<b>BS</b> (backspace)	40	28	050	&#40;	<b>(</b>	72	48	110	&#72;	<b>H</b>	104	68	150	&#104;	<b>h</b>
9	9	011	<b>TAB</b> (horizontal tab)	41	29	051	&#41;	<b>)</b>	73	49	111	&#73;	<b>I</b>	105	69	151	&#105;	<b>i</b>
10	A	012	<b>LF</b> (NL line feed, new line)	42	2A	052	&#42;	<b>*</b>	74	4A	112	&#74;	<b>J</b>	106	6A	152	&#106;	<b>j</b>
11	B	013	<b>VT</b> (vertical tab)	43	2B	053	&#43;	<b>+</b>	75	4B	113	&#75;	<b>K</b>	107	6B	153	&#107;	<b>k</b>
12	C	014	<b>FF</b> (NP form feed, new page)	44	2C	054	&#44;	<b>,</b>	76	4C	114	&#76;	<b>L</b>	108	6C	154	&#108;	<b>l</b>
13	D	015	<b>CR</b> (carriage return)	45	2D	055	&#45;	<b>-</b>	77	4D	115	&#77;	<b>M</b>	109	6D	155	&#109;	<b>m</b>
14	E	016	<b>SO</b> (shift out)	46	2E	056	&#46;	<b>.</b>	78	4E	116	&#78;	<b>N</b>	110	6E	156	&#110;	<b>n</b>
15	F	017	<b>SI</b> (shift in)	47	2F	057	&#47;	<b>/</b>	79	4F	117	&#79;	<b>O</b>	111	6F	157	&#111;	<b>o</b>
16	10	020	<b>DLE</b> (data link escape)	48	30	060	&#48;	<b>0</b>	80	50	120	&#80;	<b>P</b>	112	70	160	&#112;	<b>p</b>
17	11	021	<b>DC1</b> (device control 1)	49	31	061	&#49;	<b>1</b>	81	51	121	&#81;	<b>Q</b>	113	71	161	&#113;	<b>q</b>
18	12	022	<b>DC2</b> (device control 2)	50	32	062	&#50;	<b>2</b>	82	52	122	&#82;	<b>R</b>	114	72	162	&#114;	<b>r</b>
19	13	023	<b>DC3</b> (device control 3)	51	33	063	&#51;	<b>3</b>	83	53	123	&#83;	<b>S</b>	115	73	163	&#115;	<b>s</b>
20	14	024	<b>DC4</b> (device control 4)	52	34	064	&#52;	<b>4</b>	84	54	124	&#84;	<b>T</b>	116	74	164	&#116;	<b>t</b>
21	15	025	<b>NAK</b> (negative acknowledge)	53	35	065	&#53;	<b>5</b>	85	55	125	&#85;	<b>U</b>	117	75	165	&#117;	<b>u</b>
22	16	026	<b>SYN</b> (synchronous idle)	54	36	066	&#54;	<b>6</b>	86	56	126	&#86;	<b>V</b>	118	76	166	&#118;	<b>v</b>
23	17	027	<b>ETB</b> (end of trans. block)	55	37	067	&#55;	<b>7</b>	87	57	127	&#87;	<b>W</b>	119	77	167	&#119;	<b>w</b>
24	18	030	<b>CAN</b> (cancel)	56	38	070	&#56;	<b>8</b>	88	58	130	&#88;	<b>X</b>	120	78	170	&#120;	<b>x</b>
25	19	031	<b>EM</b> (end of medium)	57	39	071	&#57;	<b>9</b>	89	59	131	&#89;	<b>Y</b>	121	79	171	&#121;	<b>y</b>
26	1A	032	<b>SUB</b> (substitute)	58	3A	072	&#58;	<b>:</b>	90	5A	132	&#90;	<b>Z</b>	122	7A	172	&#122;	<b>z</b>
27	1B	033	<b>ESC</b> (escape)	59	3B	073	&#59;	<b>;</b>	91	5B	133	&#91;	<b>[</b>	123	7B	173	&#123;	<b>{</b>
28	1C	034	<b>FS</b> (file separator)	60	3C	074	&#60;	<b>&lt;</b>	92	5C	134	&#92;	<b>\</b>	124	7C	174	&#124;	<b> </b>
29	1D	035	<b>GS</b> (group separator)	61	3D	075	&#61;	<b>=</b>	93	5D	135	&#93;	<b>]</b>	125	7D	175	&#125;	<b>}</b>
30	1E	036	<b>RS</b> (record separator)	62	3E	076	&#62;	<b>&gt;</b>	94	5E	136	&#94;	<b>^</b>	126	7E	176	&#126;	<b>~</b>
31	1F	037	<b>US</b> (unit separator)	63	3F	077	&#63;	<b>?</b>	95	5F	137	&#95;	<b>_</b>	127	7F	177	&#127;	<b>DEL</b>

Source: [www.LookupTables.com](http://www.LookupTables.com)

HTL LE  NDING



- Unicode Consortium ([www.unicode.org](http://www.unicode.org))
  - Ziel: Abbildung sämtlicher lebender und historischer Zeichenvorräte




Version	Sprachumfang	Codepoints
1.0.0 Okt. 1991	Europäische, nahöstliche, indische Sprachen, 7.161 Zeichen	Max. 65536 (16-Bit) U+0000 .. U+FFFF
1.0.1 Mai 1992	+ ostasiatische Sprachen, 28.359 Zeichen	Max. 65536 (16-Bit) U+0000 .. U+FFFF
2.0.0 Juli 1996	Erweiterung auf bis zu 4 Byte 38.950 Zeichen	Max 1.114.112 U+0000 .. U+10FFFF
7.0.0 Juni 2014	123 Sprachfamilien, 113 021 Zeichen	

# Unicode-Tabelle

- Die ersten 128 Plätze („**Codepoints**“) sind identisch mit ASCII
- Die insgesamt 1.114.112 Codepoints sind in Ebenen („**Planes**“) unterteilt
  - Die meisten / gebräuchlichsten Zeichen liegen in der „**Basic Multilingual Plane**“ (BMP)
    - Das sind die Codepoints zwischen 00 00 und FF FD (0 und 65.533) => 2 Byte reichen für die Darstellung!
  - Für alle anderen Planes wird ein drittes Byte benötigt (1 00 00 – 10 FF FD)

# Unicode: Beispiele

<http://unicode-table.com/de/>

Name des Zeichens	Zeichen	Unicode	Unicode dezimal
Latin Capital Letter V	V	U+0056	86
Latin Small Letter a with Diaresis	ä	U+00E4	228
Cyrillic Capital Letter Nje	Њ	U+040A	1.034
Devanagari Letter Ca	च	U+091A	2.330
Cjk unified ideograph	变	U+53D8	21.464
Gothic Letter Thiuth	ψ	U+10338	66.360
Ambulance		U+1F691	128.657

## Achtung:

Es hängt vom eingestellten Font (Zeichensatz) ab, ob ein Unicode-Zeichen auch wirklich richtig am Bildschirm dargestellt werden kann!

# Ist ein Zeichen eine Zahl?

```
string s = "a2b3";  
char ch = s[1];
```

```
/*
```

...liefert das Zeichen mit Index 1 und speichert dieses in die char-Variable ch (hier '2')  
intern wird in char der numerische Unicode-Wert des Zeichens gespeichert (hier 50)

Zur Prüfung ob das in ch gespeicherte Zeichen eine Zahl ist, wird überprüft ob der Unicode-Wert des zu prüfenden Zeichens größer oder gleich dem Unicode-Wert des Zeichens '0' ist und kleiner oder gleich dem Unicode-Wert des Zeichens '9'. \*/

```
if ((ch >= '0') && (ch <= '9'))  
{  
    Console.WriteLine("Dieses Zeichen ist eine Zahl!");  
}
```

//Hierfür gibt es auch eine fertige Funktion der Klasse char:

```
if (char.IsDigit(ch) == true)  
{  
    Console.WriteLine("Dieses Zeichen ist eine Zahl!");  
}
```

# Einige Stringfunktionen

```
string s = "a long string";
```

0	1	2	3	4	5	6	7	8	9	10	11	12
a		l	o	n	g		S	t	r	i	n	g

```
string x;
```

```
x = s.ToUpper();
```

// liefert s in Großbuchstaben (hier „A LONG STRING“)

```
x = s.ToLower();
```

// liefert s in Kleinbuchstaben (hier: „a long string“)

```
int i = s.IndexOf("ng");
```

// liefert Index des 1. Vorkommens von "ng" in s (hier 4) oder -1 wenn  
// der gesuchte Teilstring nicht vorkommt.

```
i = s.IndexOf("ng", 5);
```

// liefert Index des 1. Vorkommens von "ng" ab Index 5 (hier 11)

```
i = s.LastIndexOf("ng");
```

// liefert Index des letzten Vorkommens von "ng" (Varianten wie oben)

```
x = s.Substring(2);
```

// liefert Teilstring ab Index 2 (hier "long string")

```
x = s.Substring(2, 4);
```

// liefert Teilstring ab Index 2 werden die nächsten 4 Zeichen geliefert  
// (hier: "long")

```
if (s.StartsWith("abc")) ...
```

// liefert true, falls s mit "abc" beginnt

```
if (s.EndsWith("abc")) ...
```

// liefert true, falls s mit "abc" ended

# Weitere Stringfunktionen

- Suche mit einer Suchmaschine (z.B. Google) nach C# Stringfunktionen.
- Die Microsoft C# (.NET) Dokumentation ist ebenfalls hilfreich:

[http://msdn.microsoft.com/de-de/library/system.string\\_methods%28v=vs.110%29.aspx](http://msdn.microsoft.com/de-de/library/system.string_methods%28v=vs.110%29.aspx)

# Stringkonversionen

```
int i = int.Parse("123");  
i = Convert.ToInt32("123");  
double d = double.Parse("3.14");  
d = Convert.ToDouble("3.14");
```

// String  $\Rightarrow$  int

// String  $\Rightarrow$  double

```
int x;
```

```
if (int.TryParse("123", out x) == true)...
```

// Konvertierung mit Gültigkeitsprüfung:  
// Wenn die Konvertierung gut geht,  
// dann liefert die Methode true und  
// der konvertierte Wert steht in x

```
double d;
```

```
if (double.TryParse("3.14", out f) == true)...
```

// Diese Möglichkeit der Konvertierung  
// gibt es für alle einfachen Typen  
// short, long, int, byte....

```
string s;  
s = Convert.ToString(123);  
s = Convert.ToString(3.14);
```

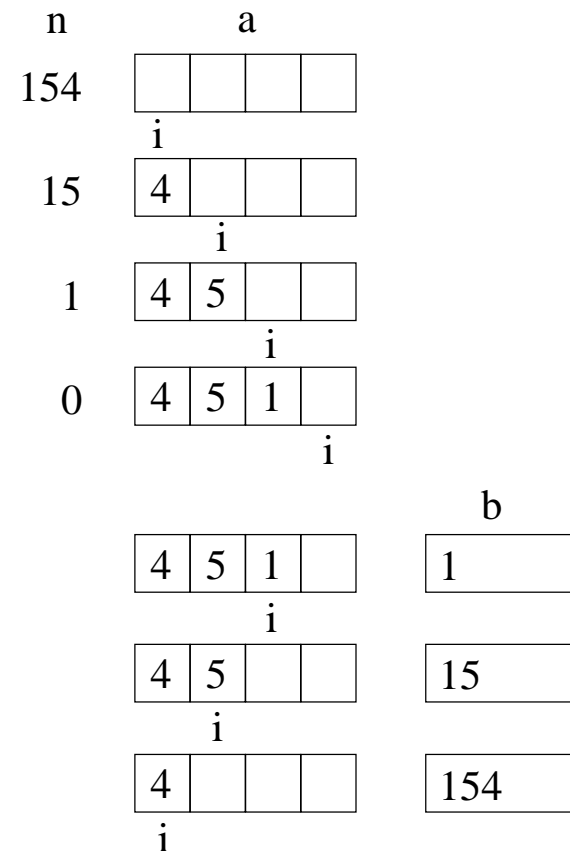
// int  $\Rightarrow$  String

// double  $\Rightarrow$  String

# Beispiel: Zahl in String konvertieren (ohne Convert)

Idee: Ziffern mit  $n \% 10$  abspalten und in *char*-Array sammeln

```
int n = 154; //Umzuwandelnde Zahl
char[] a = new char[10];
int i = 0;
do {
    a[i] = (char) (n % 10 + '0');
    n = n / 10;
    i++;
} while (n > 0);
string b = "";
do {
    i--;
    b = b + a[i];
} while (i > 0);
Console.WriteLine("String: " + b);
```



Funktioniert auch ohne Zwischenspeicherung im Array → Versuche es!