INP (4)

Prof. Dr.-Ing. S. Gössner

University of Applied Sciences Lippe & Höxter

Inhalt

Datentyp String

Ein *String* ist allgemein eine *Zeichenfolge*. In Javascript ist ein String eine Folge von "*16 bit Unicode Zeichen*".

Literale	Syntax	Beispiele
String	`/(".*")|('.*')/`	`"abc", 'de f', "x\"", ''`

- Ein String wird durch Anführungszeichen `"` oder Hochkommata `'` begrenzt.
- Ein String kann nicht über mehrere Zeilen hinweg definiert werden.
- Um Anführungszeichen oder Hochkommata innerhalb eines Strings zu verwenden, müssen diese durch einen vorangestellten Backslash `\` maskiert werden.
- Der Datentyp eines einzelnen Zeichens ist nicht gesondert definiert *(char)*. Stattdessen wird ein String mit einem einzelnen Zeichen verwendet.
- Ein String ohne Inhalt (`""` oder `''`) wird *Leerstring* genannt.

Sonderzeichen

Sonderzeichen	Bedeutung
`\n`	Line feed (newline)
`\t`	Horizontal tab (Ctrl-I)
`\'`	Single quotation mark
`\"`	Double quotation mark
`\\`	Backslash

Beispiele

```
// ok
"Ausgabe über\nzwei Zeilen"
'\tAusrichtung\tmit\tTabulator'
"Klaus' Spruch: \"Hallo!\""
'c:\\ord\\ner'
// nicht ok
"text'
"zwei
 Zeilen"
'c:\programme'
```

String operator +

Der binäre Operator + verbindet zwei Strings zu Einem.

```
var s = "wie " + 'geht\'s';  // wie geht's
```

Wenn wenigstens einer der beiden Operanden ein String ist, dann ist auch das Ergebnis ein String.

+	Number/Boolean	String
Number/Boolean	Number	String
String	String	String

Beispiele:

```
var zweizeiler = "zeile eins"+
                 "zeile zwei";
"level" + 4 + 2  // level42
3+1+"2"          // 42
```

String/Number Konvertierung

Javascript stellt drei Funktionen zur Konvertierung zwischen den Datentypen *Number* und *String* zur Verfügung.

Funktion	Bedeutung
num.toString(basis)	Konvertierung einer Zahl *num* zur Basis *basis* und Rückgabe als String.
parseInt(str, basis)	Konvertierung einer Zahl in *str* angegeben bzgl. der Basis *basis* in eine Dezimalzahl.
parseFloat(str)	Konvertierung einer Zahl in *str* in eine dezimale Gleitkommazahl.

Beispiele:

```
var hex = 0x2A;
var bin = hex.toString(2); // "101010"
parseInt(bin, 2)           // 42
parseInt(hex)              // 42
(42).toString(16)          // "2a"
parseInt("123Express")     // 123
parseInt("3.14")           // 3
parseFloat("0.314e1")      // 3.14
```

String als Objekt

- Ein String ist nicht veränderbar.
- Einer Variablen vom Datentyp *String* kann ein anderer String zugewiesen werden.
- Ein String ist gleichzeitig ein Objekt und besitzt *Eigenschaften* und *Methoden*.

Eigenschaft/Methode	Bedeutung
str.length	Liefert die Anzahl der Zeichen des Strings *str*.
str.charAt(idx)	Liefert das Zeichen innerhalb des Strings *str* an der Stelle *idx*. *(Beachte: Das erste Zeichen eine Strings hat den Index '0')*.
str.charCodeAt(idx)	Liefert den Unicode-Wert des Zeichens innerhalb des Strings *str* an der Stelle *idx*.
str.indexOf(substr[,idx])	Sucht innerhalb eines Strings *str* den String *substr* beginnend an der Stelle *idx*. Bei fehlendem *idx* wird von Beginn an gesucht.
str.split(separator)	Trennt einen String *str* an jeder Stelle an der ein Zeichen aus *sep* vorkommt auf und liefert die entstehenden Strings als *Array* zurück.
str.substr(idx[,length])	Liefert einen Teilstring von *str* beginnend an der Stelle *idx* mit der Länge *length*. Fehlt *length*, dann besitzt der Teilstring alle weiteren Zeichen bis zum Ende von *str*.
str.toLowerCase()	Liefert eine Kopie von *str* zurück, in der alle Buchstaben in Kleinbuchstaben gewandelt sind.
str.toUpperCase()	Liefert eine Kopie von *str* zurück, in der alle Buchstaben in Grossbuchstaben gewandelt sind.

Datentyp Boolean

Der Datentyp _Boolean repräsentiert einen Wahrheitswert.

Literale	Syntax	Beispiele
Boolean	`/(true|false)/`	-

Beispiel

```
var go = true;

while (go) {
  go = false;
}
```

Konvertierung

Für die Konvertierung anderer Datentypen in einen Bool'schen Wert gilt die Regel:

> Die Werte `0, "", NaN, null, undefined` liefern `false`, alle anderen Werte `true`.

- Die Konvertierung eines *Boolean* in *Number* wird folgendermassen durchgeführt:
  - `true` => 1
  - `false` => 0
- Die Konvertierung eines *Boolean* in *String* ergibt:
  - `true` => `"true"`
  - `false` => `"false"`

Beispiele:

```
true + true       // 2
"alles " + true   // alles true
```

Vergleichs- und logische Operatoren

Operator	Bedeutung	Beispiele
==	Gleichheit der Werte ggfs. nach Typanpassung	`"42" == 42 // true`
!=	Ungleichheit der Werte auch nach Typanpassung	`"42" != 42 // false`
===	Gleichheit der Werte und Datentypen	`"42" === 42 // false`
!==	Ungleichheit der Werte oder Datentypen	`"42" !== 42 // true`
>	Grösser als	`42 > 43 // false`
<	Kleiner als	`42 < 43 // true`
>=	Grösser oder gleich	`42 >= 43 // false`
<=	Kleiner oder gleich	`42 <= 43 // true`
&&	logisches Und	`2 < 3 && 3 >= 4 // false`
||	logisches Oder	`2 < 3 || 3 >= 4 // true`
!	Negation	`2 < 3 && !(3 >= 4) // true`

- Das logische *Und* besitzt eine höhere Prorität als das logische *Oder*.
- Die doppelte Negation liefert den äquivalenten Wahrheitswert zu jedem belieben Wert

```
!!"" == true // false
```

Operatoren Übersicht

Operatoren führen eine Operation mit einem oder mehreren Operanden durch.

- unärer Operator: `@X`
- binärer Operator: `X @ Y`
- ternärer Operator `X ? Y : Z`

Operator type	Operators
member	. []
call / create instance	() new
negation/increment	! ~ - + ++ - - typeof void delete
multiply/divide	* / %
addition/subtraction	+ -
bitwise shift	<< >> >>>
relational	< <= > >= in instanceof
equality	== != === !==
bitwise-and	&
bitwise-xor	^
bitwise-or	|
logical-and	&&
logical-or	||
conditional	?:
assignment	= += -= *= /= %= <<= >>= >>>= &= ^= |=
comma	,

Die Operatoren in der Tabelle besitzen **abnehmende** Priorität, d.h. Operatoren in der ersten Zeile haben die *höchste* und in der letzten Zeile die *niedrigste* Priorität. Operatoren in derselben Zeile besitzen die gleiche Priorität.

[Quelle: MozillaDeveloper Center]

Ausdrücke

Ein *Ausdruck (expression)* …

- ist Teil einer *Anweisung*.
- ist eine gültige Sequenz von Literalen, Variablen, Operatoren und/oder Ausdrücken.
- resultiert in einem Wert mit definierten Datentyp *number, string, boolean* oder *object*.
  `( 8*5 + 2 )`
- kann einer Variablen einen Wert zuweisen.
  `( x = (90 - 6)/2 )`
- heisst je nach Datentyp des Wertes
  - arithmetischer Ausdruck
  - Stringausdruck
  - logischer Ausdruck
  - Objektausdruck
- Ausdrücke werden grundsätzlich von links nach rechts – unter Berücksichtigung der Prioritätenregeln – ausgewertet.

Anweisungen

Eine Anweisung *(statement)* ...

- ist die notwendigerweise minimale Einheit zur Programmausführung.
- hat selbst keinen Wert bzw. Datentyp.
- zur Programmverzweigung ist eine *Bedingungsanweisung*.
- für eine Programmschleife ist eine *Wiederholungsanweisung*.

```
var s = "hallo", vocals=0;

for (var i=0; i<s.length; i++) {
   switch (s.charAt(i)) {
      case "a": case"A":
      case "e": case"E":
      case "i": case"I":
      case "o": case"O":
      case "u": case"U": vocals++; break;
      default: break;
   }
}

window.alert(vocals + " Vokale gefunden!");
```

Kommentare

Ein Kommentar *(comment)* …

- wird vom Interpreter nicht verarbeitet (überlesen).
- dient dem aktuellen oder einem nachfolgenden Programmierer zum besseren Verständnis von Programmteilen.
- wird gerne benutzt, um temporär einen Programmteil auszublenden.
- kann einzeilig oder mehrzeilig kodiert werden.
- in mehrzeiliger Form kann er nicht mit einem weiteren mehrzeiligen geschachtelt werden.

```
// Dies ist ein *einzeiliger* Kommentar
var x = 42;
/* Und nun
   ein mehr-
   zeiliger
   Kommentar */
```

Bedingungsanweisungen

Eine Bedingungsanweisung *(conditional expression)* …

- bewirkt eine Verzweigung im Programmablauf in Abhängigkeit von einem Booleschen Ausdruck.
- kann in zweierlei Form auftreten.

if … else

```
if (bedingung) {
   anweisung*
}
else {
   anweisung*
}
```

switch

```
switch(ausdruck) {
   case literal1:
      anweisung*
      break;
   ...
   case literalN:
      anweisung*
      break;
   default:
      anweisung*
      break;
}
```

Wiederholungsanweisungen

Eine Wiederholungsanweisung *(loop)* …

- erlaubt die mehrfach, periodische Abarbeitung von Programmteilen.
- kann in vier unterschiedlichen Schreibweisen verwendet werden.

do … while

```
do {
   anweisung*
}
while (bedingung);
```

for

```
for (initialisierung; bedingung; inkrementierung) {
   anweisung*
}
```

for … in

```
for (eigenschaft in object) {
   anweisung*
}
```

while

```
while (bedingung) {
   anweisung*
}
```