[1] Reguläre Ausdrücke

Reguläre Ausdrücke sind bestimmte Muster, um Texte zu überprüfen, zu zählen, zu suchen und zu ersetzen. Mit dem so genannten ***RegExp***-Objekt werden reguläre Sprachausdrücke erzeugt, wobei es sich um String-Objekte handelt, welche nach einer speziellen Syntax aufgebaut sind.

(1) Bildung von regulären Ausdrücke mit Literal-Schreibweise: *var reg = /String/;*
Der Schrägstrich (/) dient als Begrenzung für reguläre Ausdrücke.
(2) Bildung von regulären Ausdrücken mit Konstruktorfunktion: *var reg = new RegExp(String);*
(3) Verwendung von zusätzlichen Steuerzeichen (Flags): *var reg = new RegExp(String, Flags);*
Beispiele für Flags: g = global (d.h. im ganzen String)
i = case-insensitive (d.h. unabhängig von Klein-/Großschrift)
m = multiline (d.h. in allen Textzeilen)

Ein erstes Beispiel: *var str = "In genau diesem Text wird jetzt gesucht.";*

```
(a)  var posi = str.search(/Text/);          // Textsuche, ergibt die Position 16
(b)  var myArray = str.match(/e/gi);         // Textprüfung, ergibt ein array [e,e,e,e,e,e]
                                             //  g = global, i = case-insensitiv
(c)  var count  = str.match(/e/gi).length;   // Textzählung, ergibt die Anzahl 6
(d)  var strneu = str.replace(/e/gi, 'XYZ');  // Textersetzung, ersetzt alle „e“ durch „XYZ“
```

Will man anstelle eines konstanten Strings eine String-Variable „such“ in einem regulären Ausdruck verwenden, dann muss sie mit „new RegExp(such, flags)“ registriert werden. Beispielsweise:

```
var str = "In genau diesem Text wird jetzt gesucht.";
var such =  "e“;

var reg = new RegExp(such, 'gi');            // neues RegExp-Objekt mit den Flags 'gi'
                                             // anstelle von var reg = /e/gi;
var count = str.match(reg).length;           // ergibt die Anzahl 6
```

Als zweites Beispiel für die Anwendung von **regulären Ausdrücken** soll die Trimmfunktion dienen. Darunter versteht man eine Funktion, die aus einem String x alle vorangestellten und auch alle nachgestellten Blanks (Leerzeichen) entfernt.

Diese Aufgabe läßt sich zunächst mit der vordefinierten trim-Methode von Strings lösen: **y = x.trim();**

Eine andere Lösung kann mittels regulärer Ausdrücke erfolgen:

```
function  myTrim(x)  {  return x.replace(/^\s+|\s+$/gm,'')  }
```

Erklärung:
- x = String (ev. mit Zeilenvorschüben \n)
- / . . . / = String-Muster mit Spezialzeichen (/ = Slash als Begrenzung)
- \ . . . = Der Backslash als Kenn-Markierung für reguläre Zeichen
- gm = Steuerzeichen für global (g) und multiline (m)
- ' ' = Nullstring

Bedeutung: Ersetze im String **x** jedes am Zeilenanfang (^) stehende Blank (**\s+**) oder (|) jedes am Zeilende (**$**) stehende Blank (**\s+**) im ganzen String (**g**) in allen Zeilen (**m**) durch den Nullstring (**' '**).

Das Spezialzeichen + nimmt Bezug auf das davor liegende Zeichen (**\s**) und bewirkt die Gültigkeit für alle diese Zeichen im jeweiligen Kontext, d.h. hier für alle Zeilenanfänge und alle Zeilenenden.

Die vollständige Liste von Spezialzeichen für reguläre Ausdrücke findet man im nächsten Kapitel [2].

• REPL.html, Suchen und Ersetzen mit regulären Ausdrücken

```
<!DOCTYPE html>
<html>
<head>
   <title> repl.html </title>
   <meta charset="ISO-8859-1">
   <meta name="author" content="Herbert Paukert">
</head>

<body>
<p>
<b>REPL.HTML, Ersetzen von Stringteilen </b><br>
(1) Zählen eines Zeichens "b" im String s:  count = s.match(/b/gi).length;<br>
(2) Ersetzen von "abc" durch "XYZ" im String s: s = s.replace(/abc/gi,XYZ);
</p>
<script>

  var s = ' abcde abc abc ';
  var t = 'abc';
  var u = 'XYZ'
  s = prompt('Stringeingabe',s);

  var info1 = 'SUCH-String in [' + s + ']';
  t = prompt(info1,t);
  var reg = new RegExp(t,'gi');
  count = s.match(reg).length;
  alert(t + '  kommt in [' + s + '] genau ' + count + '-Mal vor!');

  var info2 = 'ERSATZ-String in [' + s + ']';
  u = prompt(info2,u);
  v = s.replace(reg,u);
  alert(t + '  wurde in [' + s + '] durch ' + u + ' ersetzt:\n [' + v + ']');

</script>
</body>
</html>
```

• TRIM.html, Trimmen mit regulären Ausdrücken

```
<!DOCTYPE html>
<html>
<head>
   <title> trim.html </title>
   <meta charset="ISO-8859-1">
   <meta name="author" content="Herbert Paukert">
</head>

<body>
<p>
<b>TRIM,HTML, Trimmen eines Strings s </b> (d.h. Entfernen aller randständigen Blanks)<br>
(1) Ohne regulärem Ausdruck mit der Stringfunktion:  s.trim();<br>
(2) Mittels regulärem Ausdruck:  function myTrim(s) { return s.replace(/^\s+|\s+$/gm,'') }<br>
</p>
<script>

  function myTrim(x) { return x.replace(/^\s+|\s+$/gm,''); }

  var s = '          abcde abc abc          ';
  var t = 'abc';
  var u = 'XYZ'

  s = prompt('Stringeingabe mit randständigen Blanks',s);
  var ungetrimmt = s;
  var getrimmt = s.trim();
  var info1 = '1: Trimmen eines Strings ohne regulärem Ausdruck: ';
  alert(info1 + '\n\n ungetrimmt: ' + ungetrimmt + '\n     getrimmt: ' + getrimmt + '\n ');

  var ungetrimmt = s;
  var getrimmt = myTrim(s);
  var info2 = '2: Trimmen eines Strings mittels regulärem Ausdruck: ';
  alert(info2 + '\n\n ungetrimmt: ' + ungetrimmt + '\n     getrimmt: ' + getrimmt + '\n ');

</script>
</body>
</html>
```

[2] Liste der Spezialzeichen für reguläre Ausdrücke

Reguläre Ausdrücke (Regular Expressions oder kurz ***RegEx***) werden in JavaScript verwendet, um Zeichenfolgen (Strings) zu durchsuchen, zu überprüfen, zu vergleichen, zu extrahieren oder zu ersetzen. Zur Erstellung von regulären Ausdrücken gibt es in JavaScript genau zwei Möglichkeiten:

Variante 1: Mit Schrägstrichen (const regex = /hallo/;)
Variante 2: Mit dem RegExp-Konstruktor (const regex = new RegExp("hallo");)

1. Wichtige RegEx-Zeichen

Die folgenden Zeichen haben eine besondere Bedeutung:
(Zeichen, Bedeutung, Beispiel)

. Beliebiges Zeichen (außer Zeilenumbruch), (/h.llo/)
\d Eine Ziffer (0–9), (/\d/)
\D Keine Ziffer, (/\D/)
\w Buchstabe, Ziffer oder _, (/\w/)
\W Kein Wortzeichen, (/\W/)
\b Wortgrenze, (/\b/)
\B Keine Wortgrenze, (/\B/)
\s Leerzeichen bzw. Whitespace, (/\s/)
\S Kein Whitespace, (/\S/)
^ Anfang eines Strings, (/^Hallo/)
$ Ende eines Strings, (/Welt$/)
* Null oder mehr Wiederholungen, (/abc*/)
+ Eine oder mehr Wiederholungen, (/abc+/)
? Null oder eine Wiederholung, (/abc?/)
{n} Genau n Wiederholungen, (/\d{4}/)
[abc] Eines der angegebenen Zeichen, (/[abc]/)
[^abc] Keines der angegebenen Zeichen, (/[^abc]/)
() Gruppierung, (/(abc)+/)
\| Alternative (ODER), (/rot\|blau/)

2. Wichtige RegEx-Steuerzeichen

Die folgenden Zeichen dienen der Prozess-Steuerung:
(Zeichen, Bedeutung, Beispiel)

\t Horizontaler Tabulator (Tab), (/\t/)
\n Zeilenumbruch (Line Feed, LF), (/\n/)
\r Wagenrücklauf (Carriage Return, CR)
\v Vertikaler Tabulator
\f Seitenvorschub (Form Feed)
\b Backspace innerhalb einer Zeichenklasse [...]
\0 Nullzeichen (NUL), sofern keine Dezimalziffer folgt

\cA bis \cZ Steuerzeichen (Control Characters)
\xHH Zeichen mit hexadezimalen Code, (ESC-Zeichen: /\x1B/)

3. Wichtige RegEx-Funktionen

(Methode, Bedeutung, Beispiel))

test()	Prüft, ob ein Muster vorhanden ist
exec()	Liefert den gefundenen Treffer
match()	Liefert passende Treffer (als Array-Elemente)
matchAll()	Liefert alle Treffer inklusive Gruppen
replace()	Ersetzt gefundene Zeichenfolgen
search()	Liefert die Position des ersten Treffers
split()	Zerlegt einen String anhand eines Musters (Trennzeichen, Separator) in ein Array

Beispiel: test()

```
const text = "Hallo Welt!";
const regex = /Hallo/;
console.log(regex.test(text));
// true
```

Beispiel: match()

```
const text = "Ich lerne JavaScript.";
const ergebnis = text.match(/JavaScript/);
console.log(ergebnis[0]);
// JavaScript
```

Beispiel: replace()

```
const text = "Hallo Welt!";
const neu = text.replace(/Welt/, "JavaScript");
console.log(neu);
// Hallo JavaScript!
```

4. RegEx-Flags (Modifikatoren)

Flags verändern das Suchverhalten.
(Flag, Bedeutung)

g	Globale Suche (alle Treffer)
i	Groß-/Kleinschreibung ignorieren
m	Mehrzeilenmodus
s	Punkt umfasst auch Zeilenumbrüche
u	Unicode-Modus
y	Suche ab der aktuellen Position

Beispiel:

```
const text = "Hallo hallo HALLO";
const regex = /hallo/gi;
console.log(text.match(regex));
// ["Hallo", "hallo", "HALLO"]
```

5. Praktische Beispiele

Beispiel A: Prüfen, ob ein String nur Zahlen enthält.

```
const regex = /^\d+$/;
console.log(regex.test("12345")); // true
console.log(regex.test("123abc")); // false
```

Beispiel B: E-Mail-Adresse überprüfen

```
const regex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
console.log(regex.test("test@example.com"));
// true
```

Das ist eine einfache Plausibilitätsprüfung,
keine vollständige Prüfung aller gültigen E-Mail-Adressen.

Beispiel C: Telefonnummer überprüfen.

```
const regex = /^\+?[0-9\s-]{6,20}$/;
console.log(regex.test("+43 660 1234567"));
// true
```

Beispiel D: Alle Zahlen aus einem Text extrahieren.

```
const text = "Ich habe 25 Äpfel und 10 Birnen.";
const zahlen = text.match(/\d+/g);
console.log(zahlen);
// ["25", "10"]
```

Beispiel E: Wörter ersetzen

```
const text = "Java ist toll. Java ist beliebt.";
const neu = text.replace(/Java/g, "JavaScript");
console.log(neu);
// JavaScript ist toll. JavaScript ist beliebt.
```

6. Capture Groups (Gruppen)

Mit runden Klammern können bestimmte Teile eines Treffers
separat erfasst werden.

```
const text = "15.11.2026";
const regex = /(\d{2})\.(\d{2})\.(\d{4})/;
const result = text.match(regex);
console.log(result[0]); // 15.11.2026
console.log(result[1]); // 15
console.log(result[2]); // 11
console.log(result[3]); // 2026
```

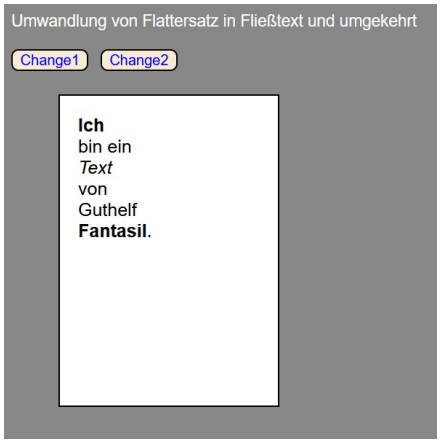
Damit lassen sich Datum, Monat und Jahr getrennt auslesen.

Hinweis:

Für JavaScript-Anwendungen sind besonders test(), match(), replace(), \d, \w, ^, $
und die Flags g und i wichtig. Damit kann man sehr viele praktische Aufgaben lösen.

[3] Das Demo-Programm „*chText.html*“

Das nachfolgende Programm zeigt die Mächtigkeit und Schönheit der Programmiersprache JavaScript. Dabei wird ein vorgegebener Text im Flattersatz umgewandelt in das Fließtext-Format und umgekehrt. In der ersten Programmversion (***chText.html***) ist der Code in Kurzform geschrieben. In der zweiten Programmversion (***chText_.html***) ist der Code in einer Langform mit mit ausführlichen Kommentaren versehen. Das Programm verwendet reguläre Ausdrücke als wichtige Codestrukturen.





Erste Programmversion

```
<!DOCTYPE html>
<html lang="de">
<head>
<title>chText.html</title>
<meta charset="ISO-8859-1">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<meta name="description" content="Flattersatz <-> Fließtext">
<meta name="keywords" content="Flattersatz,Fließtext">
<meta name="author" content="Herbert Paukert">

<style>
body {
   padding-left: 20px;
   background-color: #888888;
   color: white;
   font-family: 'Arial';
   font-size: 18px;
}
#txt {
   position: absolute;
   left: 80px;
   top: 120px;
   width: 200px;
   height: 300px;
   background-color: white;
   border: 2px solid black;
   padding: 20px;
   color: black;
   font-family: 'Arial';
   font-size: 20px;
   text-align: left;
   editable: true;
}
.btn {
   margin-right: 8px;
   border: 2px solid black;
   border-radius: 8px;
   background-color: antiquewhite;
   color: blue;
   font-family: 'Arial';
   font-size: 16px;
}
</style>
```

```
<script>

// Ein Demo für die Mächtigkeit und Schönheit von JavaScript
// var text = "<b>Ich</b><br>bin ein<br><i>Text</i><br>von<br>Guthelf<br><b>Fantasil</b>.<br>";
var okay = false;

function inFliessText(tex) {
// Umwandlung von Flattersatz in Fließtext (Kurzform)
   return tex
          .split(/\n\s*\n/)
          .map(absatz =>
                absatz
                   .replace(/-\s*\n\s*/g, "")
                   .replace(/\n/g, " ")
                   .replace(/\s+/g, " ")
                   .trim()
           )
           .join("\n\n");
}

function outFliessText() {
// Ausgabe von Fließtext (Kurzform)
   let text = document.getElementById("txt").innerHTML;
   text = text.replaceAll("<br>","\n");
   text = inFliessText(text);
   document.getElementById("txt").innerHTML = text;
   okay = true;
}

function inFlatterSatz(tex, maxZei = 100) {
// Umwandlung von Fließtext in Flattersatz (Kurzform)
   const woerter = tex.replace(/\s+/g, ' ').trim().split(' ');
   let zeilen = [];
   let aZeile = '';
   for (const wort of woerter) {
     if ((aZeile + wort).length > maxZei) {
       zeilen.push(aZeile.trim());
       aZeile = wort + ' ';
     } else { aZeile += wort + ' '; }
   }
   if (aZeile) { zeilen.push(aZeile.trim()); }
   return zeilen.join('\n');
}

function outFlatterSatz() {
// Ausgabe von Flattersatz (Kurzform)
   if (!okay) { alert('Zuerst Schalter <Change1> anklicken !'); return; }
   let text = document.getElementById("txt").innerHTML;
   text = inFlatterSatz(text,10);
   text = text.replaceAll("\n","<br>");
   document.getElementById("txt").innerHTML = text.trim();
   okay = false;
}

</script>
</head>

<body>
<br>Umwandlung von Flattersatz in Fließtext und umgekehrt.<br>
<div id="txt">
<b>Ich</b><br>bin ein<br><i>Text</i><br>von<br>Guthelf<br><b>Fantasil</b>.<br>
</div>
<br><br>
<input type="button" class="btn" value=" Change1 " onclick="outFliessText()">
<input type="button" class="btn" value=" Change2 " onclick="outFlatterSatz()">
<br><br>
</body>
</html>
```

Zweite Programmversion

```
<!DOCTYPE html>
<html lang="de">
<head>
<title>chText_.html</title>
<meta charset="ISO-8859-1">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<meta name="description" content="Flattersatz <-> Fließtext">
<meta name="author" content="Herbert Paukert">

<style>
body {
   padding-left: 20px;
   background-color: #888888;
   color: white;
   font-family: 'Arial';
   font-size: 18px;
}
#txt {
   position: absolute;
   left: 80px;
   top: 120px;
   width: 200px;
   height: 300px;
   background-color: white;
   border: 2px solid black;
   padding: 20px;
   color: black;
   font-family: 'Arial';
   font-size: 20px;
   text-align: left;
   editable: true;
}
.btn {
   margin-right: 8px;
   border: 2px solid black;
   border-radius: 8px;
   background-color: antiquewhite;
   color: blue;
   font-family: 'Arial';
   font-size: 16px;
}
</style>

<script>
// var text = "<b>Ich</b><br>bin ein<br><i>Text</i><br>von<br>Guthelf<br><b>Fantasil</b>.<br>";
var okay = false;

function inFliessText(tex) {
// Umwandlung von Flattersatz in Fließtext (Langform)

      function myMap(value) {
         // Führt mit jedem Element eines String-Arrays diese Funktion aus.
         value = value.replace(/-\s*\n\s*/g, "");
         // Regulärer Ausdruck /-\s*\n\s*/
         // g ... global, "" ... leerer String
         // 1.Methode: Entfernt im Objekt "value" alle Bindestriche "-",
         //            welche direkt vor einem Zeilenumbruch stehen,
         //            mit Leerraumzeichen (Blank,Tabulator,...) davor und dahinter.
         value = value.replace(/\n/g, " ");
         // 2.Methode: Ersetzt jeden Zeilenumbruch durch jeweils ein Blank.
         value = value.replace(/\s+/g, " ");
         // 3.Methode: Ersetzt alle mehrfachen Leerraumzeichen durch einfache Blanks.
         value = value.trim();
         // 4.Methode: Entfernt alle randständigen Blanks.
         return value; // Liefert das so modifizierte Array als Return-Wert.
      }

      let absatz = tex.split(/\n\s*\n/);
      // Regulärer Ausdruck: /\n\s*\n/
      // Sucht einen Zeilenumbruch, gefolgt von beliebig vielen Leerraumzeichen
      // und einem Zeilenumbruch als Separator für ein Array von Absätzen
      // Dadurch wird der Text in ein Array von Strings umgewandelt.
      absatz = absatz.map(myMap);
      // Führt mit jedem Array-Element die Funktion "myMap" aus
      tex = absatz.join("\n\n");
      // Wandelt das modifizierte Absatz-Array in einen Text um,
      // wo der Array-Separator durch einen doppelten Zeilenumdrch ersetzt wird.
    return tex; // Liefert den so modifizierten Text als Return-Wert.
}
```

```
function outFliessText() {
// Ausgabe von Fließtext (Langform)
   let text = document.getElementById("txt").innerHTML;
   // Übernimmt den div-Inhalt mit allen Textauszeichnungen.
   text = text.replaceAll("<br>","\n");
   // Ersetzt alle physischen Zeilenmbrüche durch "\n".
   text = inFliessText(text);
   // Umwandlung von Flattersatz in Fließtext.
   document.getElementById("txt").innerHTML = text;
   // Übergibt den modifizierten div-Inhalt.
   okay = true; // Steuervariable für Rückumwandlung von Fließtext in Flattersatz.
}

function inFlatterSatz(tex, maxZei) {
// Umwandlung von Fließtext in Flattersatz (Langform)
// und Entfernung von mehrfachen Leerzeichen
   if (!maxZei) { maxZei = 100; }
   // Wenn "maxZeichen" nicht angegeben ist, dann ist der Standardwert 100.
   let woerter,wort,wl;
   // Definition von drei lokalen Variablen
   woerter = tex.replace(/\s+/g, ' ');
   // Ersetzt ein oder aufeinanderfolgende Leerraumzeichen global durch Blanks.
   woerter = woerter.trim();
   // Entfernt alle randständigen Blanks
   woerter = woerter.split(' ');
   // Wandelt den String "woerter" in ein Array um mit Blanks als Separator.
   let zeilen = [];
   let aZeile = '';
   // Definiert das leere Array "zeilen" und die aktelle Zeilen-Variable "aZeile".
   // Dann wird nachfolgende Schleife ausgeführt (mit "wort" als Element aus "woerter").

   for (wort of woerter) {
     wl = (aZeile + wort).length;
     // Ermittelt fortlaufend die Länge wl der aktuellen Zeile.
     if (wl > maxZei) {
       // Wenn wl > maxZei ist, dann wird die getrimmte aktuelle Zeile "aZeile"
       // im Zeilen-Array "zeilen" aufgestapelt und die neue, aktuelle Zeile
       // "aZeile" mit dem aktuellen Wort begonnen.
       zeilen.push(aZeile.trim());
       aZeile = wort + ' ';
     }
     else {
       aZeile = aZeile + wort + ' ';
       // Andernfalls beginnt eine neue Zeile mit dem aktuellen Wort.
     }
   }
   if (aZeile) {
     zeilen.push(aZeile.trim());
     // Nach dem Schleifenende wird eine nicht leere letzte Zeile noch aufgestapelt.
   }
   zeilen = zeilen.join('\n');
   // Wandelt das Array "zeilen" mit "\n" als Trennzeichen in einen String um.
   return zeilen; // Liefert den String "zeilen" als Return-Wert.
}

function outFlatterSatz() {
// Ausgabe von Flattersatz (Langform)
   if (!okay) { alert('Zuerst Schalter <Change1> anklicken !'); return; }
   let text = document.getElementById("txt").innerHTML;
   // Übernimmt den div-Inhalt mit allen Textauszeichnungen.
   text = inFlatterSatz(text,10);
   // Umwandlung von Fließtext in Flattersatz mit Zeilenlänge = 10 Zeichen.
   text = text.replaceAll("\n","<br>");
   // Ersetzt alle Zeichen "\n" durch physische Zeilenumbrüche.
   document.getElementById("txt").innerHTML = text.trim();
   // Übergibt den modifizierten div-Inhalt.
   okay = false;
}
</script>
</head>

<body>
<br>Umwandlung von Flattersatz in Fließtext und umgekehrt<br>
<div id="txt">
<b>Ich</b><br>bin ein<br><i>Text</i><br>von<br>Guthelf<br><b>Fantasil</b><br>.
</div>
<br><br>
<input type="button" class="btn" value=" Change1 " onclick="outFliessText()">
<input type="button" class="btn" value=" Change2 " onclick="outFlatterSatz()">
<br><br>
</body>
</html>
```