Regular Expressions

reguläre Ausdrücke = spezielle Zeichenmuster, um zu prüfen, ob eine Zeichenkette bestimmte Zeichenkombination enthält

Reguläre Ausdrücke definieren:

1. über Konstruktorfunktion
   für reguläre Ausdrücke, die dynamisch zur Laufzeit aus Usereingabe (String) erzeugt werden

   let regExp = new RegExp('abcde');

2. über Literal-Schreibweise
   wenn regulärer Ausdruck feststeht und sich zur Laufzeit nicht ändert
   arbeitet schneller als Konstruktorfunktion

   let regExp2 = /abcde/;

Methoden von RegExp Objekt:

test()
prüft, ob Zeichenmuster in Zeichenkette vorkommt
Rückgabe: Boolean

exec()
sucht erstes Vorkommen für Zeichenmuster
Rückgabe = Array mit:

1. Zeichenkette, die auf regulären Ausdruck passt
2. IndexPosition des 1. Vorkommens in übergebener Zeichenkette
3. bergebene Zeichenkette (input)

toString()
gibt Zeichenmuster als String zurück

Methoden von String, um reguläre Ausdrücke auszuwerten:

match()
sucht innerhalb einer Zeichenkette Vorkommen, die auf Zeichenmuster zutreffen
gibt diese als Array zurück

replace()
ersetzt Vorkommen innerhalb einer Zeichenkette, die auf entsprechendes Zeichenmuster zutreffen

search()
sucht nach Vorkommen für entsprechendes Zeichenmuster
gibt den Index des ersten Vorkommens zurück (ohne Fund -1)

wichtige Notationen für RegExp

Definition einer Zeichenfolge

Ausdruck	Bedeutung
.	ein beliebiges Zeichen (nicht Zeilenumbruch)
a	das Zeichen »a«
ab	die Zeichenkette »ab«
a|b	die Zeichen »a« oder »b«

Definition einer Zeichenklasse

anders als bei Zeichenfolge - hier Bezug auf einzelnes Zeichen

Bezeichnung	Schreibweise	Bedeutung
einfache Klasse	[xyz]	eines der Zeichen x, y oder z
Negation	[^xyz]	nicht x, y, z, sonst beliebiges Zeichen
Bereich	[a-zA-Z]	ein Zeichen zwischen a und z || A und Z

vordefinierte Zeichenklassen

Schreibweise	Bedeutung
.	irgendein Zeichen
\d	Ziffer [0-9]
\D	keine Ziffer
\s	Whitespace (Leerraum)
\S	kein Whitespace (Leerraum)
\w	Wortzeichen [a-zA-Z0-9_]
\W	kein Wortzeichen

Beschreibung von Wortgrenzen und Anfang / Ende von Zeichenketten:

um nur Zeichenketten anzusprechen, die komplett auf regulären Ausdruck passen (also nicht nur zu Teil eines Strings) muss Anfang und Ende des regulären Ausdrucks mit den Zeichen ^ und $ begrenzt werden

Ausdruck	Bedeutung
^	Anfang einer Zeichenkette (Zirkumflex-Zeichen)
$	Ende einer Zeichenkette
\b	Anfang oder Ende eines Wortes
\B	keine Wortgrenze

Achtung: Bedeutung des Zirkumflex-Zeichens innerhalb von eckigen Klammern nicht für Anfang einer Zeichenkette sondern Negation des Ausdrucks - bsw. /[^0-9]/ steht für Zeichen, das keine Ziffer ist

Quantifizierer

definieren, wie viele Vorkommen von Zeichens oder Zeichenklasse in Zeichenkette vorhanden sein müssen, damit sie durch regulären Ausdruck akzeptiert wird

können für Zeichen oder Zeichenklasse bestimmen:

? hinter Zeichen	- kann optional nur einmal oder gar nicht vorkommen
* hinter Zeichen	- kann beliebig oft (auch keinmal) vorhanden sein
+ hinter Zeichen	- muss mindestens einmal vorhanden sein
{n} hinter Zeichen	- muss genau n-mal vorhanden sein
{n,} hinter Zeichen	- muss mindestens n-mal vorhanden sein
{n,x} hinter Zeichen	- muss mindestens n-mal und maximal m-mal vorhanden sein

Zusätzliche Festlegungen:

Flag	Beschreibung
g	global: alle Vorkommen finden
i	ignore case: keine Unterscheidung zw. Groß- / Kleinschreibung
m	multiline: über mehrere Zeilen anwenden, wenn sich Zeichenkette über mehrere Zeilen erstreckt

Auf einzelne Teile eines Vorkommens zugreifen

über Gruppen möglich, auf bestimmte Teile einer Zeichenkette, die auf regulären Ausdruck passt, zuzugreifen

- werden in regulärem Ausdrucks über runde Klammern definiert
- öffnende Klammer definiert Anfang der Gruppe, schließende Klammer Ende der Gruppe
- Rückgabewert von exec() enthält bei Verwendung von Gruppen ab Index 1 die Zeichen, die in eine der definierten Gruppen fallen

Vorteil der Gruppierung von Zeichen

- Quantifizierer lassen sich so auf Zeichenfolgen anwenden
- regulärer Ausdruck (es)+ steht bsw. für ein oder mehrere Vorkommen der Zeichenfolge es

Destructuring

Prinzip des Destructurings (sogenannte destrukturierende Anweisungen) ermöglicht es, Werte, die in Objekteigenschaften oder Arrays hinterlegt, relativ einfach verschiedenen Variablen zuzuweisen, quasi aus Objekt bzw. Array zu extrahieren

unterscheidet zwischen Array-Destructuring und Objekt-Destructuring.

Werte aus Arrays extrahieren - Array-Destructuring

Werte, die man mit Werten aus entsprechendem Array belegen möchte, hinter let/const/var kommasepariert in eckige Klammern schreiben (in Reihenfolge, in der Werte aus Array extrahiert werden sollen)

Schreibweise umgekehrt zur Konstruktion

wenn Variablen, denen Werte zugeordnet werden sollen, bereits existieren und nicht erst deklariert werden müssen, können let /const/var weggelassen werden

wenn Array weniger Elemente enthält als Variablen angegeben, erhält entsprechende Variable Wert undefined, wenn weniger Variablen als Array-Elemente, werden überzählige Elemente vernachlässigt

gibt Möglichkeit default-Werte (Standartwerte) zuzuweisen - wie bei Funktionsparametern

ist auch möglich, nur bestimmte Werte aus Array abzubilden - dafür an entsprechender Stelle keine Variable vorsehen (entsprechende Stelle leer lassen)

Destructuring funktioniert auch mit mehrdimensionalen Arrays

Werte aus Objekten extrahieren - Objekt-Destructuring

funktioniert analog zu Array-Destructuring - Schreibweise auch hier umgekehrt zur Konstruktion

wenn Variablen und Objekteigenschaften gleiche Bezeichner haben, geht Extrahieren noch einfacher: implizit wird entsprechende Zuweisung von Eigenschaftswert zu Variablen vorgenommen

kann auch Werte aus geschachtelten Objekten extrahieren – Objektaufbau beachten!

falls Variablen bereits deklariert sind, muss Destructuring-Anweisung als Ausdruck definiert werden (runde Klammern setzen), sonst wird es vom Interpreter als Codeblock definiert ({variable} = Objekt);

Objekt-Destructuring lässt sich auch mit Array-Destructuring kombinieren

können aus Array von Objekten in for-of-Schleife leicht Propertys extrahieren

Destructuring kann auch in Kombination mit Funktionen bzw. deren Parametern genutzt werden