**Date() - Datum und Uhrzeit**

arbeitet mit GMT-Zeit, Millisekunden seit dem 1.1.1970  
kann auf Datum und Zeit zugreifen  
wenn Aufruf ohne Parameter – Rückgabe aktueller Wert

*Syntax:* neues Datumsobjekt erzeugen  
Objektname = new Date();

*setzbar in 4 Varianten*

* new Date()
* new Date(Jahr, Monat, Tag, Stunden, Minuten, Sekunden, Millisekunden)
* new Date(Millisekunden)
* new Date(Zeichenkette\*)

\* Zeichenkette die von Date.parse() verarbeitet werden kann siehe https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Global\_Objects/Date/parse

let heute = new Date();

console.log(`Heute ist der ${heute.getDate()}.${heute.getMonth() + 1}.${heute.getFullYear()}!`);

console.log(heute.getDate()); // Gibt den aktuellen Tag im Monat (1-31) zurück  
console.log(heute.getMonth()); // Gibt den aktuellen Monat (0-11) zurück  
console.log(heute.getFullYear()); // Gibt das Jahr im Format YYYY zurück  
console.log(heute.getDay()); // Gibt den Wochentag zurück (startet Sonntag mit 0)  
console.log(heute.getHours()); // Gibt die aktuelle Stunde am Tag (0-23) zurück  
console.log(heute.getTime()); // Gibt die Millisekunden seit dem 1.1.1970 zurück

wichtigste Methoden von Date im Buch S.304f

**gebräuchliche Lese-Methoden**

* getDate() (Monatstag ermitteln)
* getDay() (Wochentag ermitteln, 0-6, 0 entspricht Sonntag)
* getMonth() (Monat ermitteln **0-11**)
* getFullYear() (volles Jahr ermitteln)
* getHours() (Stundenteil der Uhrzeit ermitteln)
* getMinutes() (Minutenteil der Uhrzeit ermitteln)
* getSeconds() (Sekundenteil der Uhrzeit ermitteln)
* getMilliseconds() (Millisekundenteil der Uhrzeit ermitteln)
* getTime() (Millisekunden seit dem 01. Januar 1970, 00:00:00 UTC)

**gebräuchliche Schreib-Methoden**

* setDate() (Monatstag setzen, 0-6, 0 entspricht Sonntag)
* setMonth() (Monat setzen, **0-11**)
* setFullYear() (volles Jahr setzen)
* setHours() (Stunden der Uhrzeit setzen)
* setMinutes() (Minuten der Uhrzeit setzen)
* setSeconds() (Sekunden der Uhrzeit setzen)

**Date.now()**gibt die Anzahl der Millisekunden, die seit dem 01.01.1970 00:00:00 UTC vergangen sind zurück  
(keine Date-Objekt sondern Number) statische Methode, kann aufgerufen werden ohne erst ein Date-Objekt zu kreieren

console.log(Date.now());

//entspricht

let jetzt = new Date();

console.log(jetzt.getMilliseconds());

console.log(heute.getDay());

**Konvertierung**

toLocaleString() (Zeitpunkt in lokales Format konvertieren)

mit Übergabe von Parametern Erzeugung einer Instanz von Date, die zur Berechnung genutzt werden kann

let gestern = new Date(); // heute

gestern.setDate(gestern.getDate()-1); // gestern

alert(gestern.toLocaleString());

**JSON**

JSON ist Abkürzung für JavaScript Object Notation - also Notation, die JavaScript-Objekte verwendet

JSON-Support ist für praktisch alle modernen Programmiersprachen verfügbar

JSON ist mittlerweile das am meisten verbreitete sprachunabhängige Datenaustauschformat im Web

In JSON erlaubt sind:

* Object
* Array
* Number
* Boolean
* String
* null

d.h. Basisdatentypen Number, Boolean und String können mit Objekten und Arrays strukturiert werden… Objekte und Arrays lassen sich beliebig schachteln... Schlüsselwort null für leeren Datensatz

wichtige Unterschiede zu „normalem“ JavaScript:

* in JSON sind Anführungszeichen zwingend (dürfen nicht weggelassen werden, wenn nur ein Object-Key vorhanden)
* In JavaScript Strings in einfachen oder doppelten Anführungsszeichen - in JSON nur doppelte erlaubt

auf json.org ist Syntax von JSON genau beschrieben

**JS-Funktionen für Im- und Export von JSON-Daten**

JSON.stringify() wandelt JS-Daten in JSON-String um

JSON.parse() liest JSON-String ein, parst ihn und gibt passende JavaScript-Daten zurück

auch andere Programmiersprachen können mit JSON-Strings umgehen  
darum JSON = tolles Austauschformat zwischen Sprachen und Plattformen

**Cookies**

kleine Textdateien, die auf Client-Seite gespeichert, übermittelt und ausgelesen werden können  
enthalten bsw. Zustimmung zu Datenschutz, Artikel im Einkaufswagen, Nutzer-Wiedererkennung

Cookie-Dateien sind reine Textdateien und enthalten u.a. folgende Infos:

* **Name** (case-insensitive) und **Wert** (nur strings erlaubt) des Cookies
* **Domain** auf Server und **Pfad** auf Server, an den Cookie geschickt werden soll  
  stellt sicher, dass Cookie nur bei Anfragen von diesem Server oder dem speziellen Verzeichnis gelesen wird (optional – wenn nicht angegeben, werden Standardwerte gesetzt)  
  domain=meinedomain.de
* **Ablaufdatum** der Cookie-Gültigkeit – danach verfällt Cookie und wird gelöscht  
  (optional – wenn nicht angegeben, wird er bei Beenden der Browsersession gelöscht)
* **Sicherheitsflag** – optional – enthält Angabe, ob Cookie nur bei SSL-Verbindungen gesendet werden soll (unterbindet Senden über ungesicherte Verbindungen) - kann nur ü+ber SSL-Verbindungen gesetzt werden

Verwaltung von Cookies über Eigenschaft cookie des document-Objekts   
evtl. bereits bestehende Werte werden angefügt, nicht überschrieben

document.cookie = "name=meincookie";  
document.cookie = "datenschutz=set; expires=Sa, 02 Nov 2019 23:59 UTC; path=/";

document.cookie = "username=jessica\_jones; max-age=60\*60\*24\*30; path=/";  
document.cookie = "username= jessica\_jones; max-age=86400; path=/"; // 1 Tag

function alertCookie() {  
 alert(document.cookie);  
}

alertCookie();

Cookies sammeln sich im Cookie-String   
beim Auslesen eines bestimmten Cookies muss Name des Cookies ausgefiltert werden

zweites Cookie mit gleichem Namen, Pfad und Domain überschreibt erstes Cookie   
unterscheiden sich Pfad oder Domain, existieren zwei Cookies mit selbem Namen

**Nachteile von Cookies**

* geringe erlaubte Speichergröße: 4kb
* werden mit jeder Anfrage mitgesendet – erhöhtes Datenvolumen
* werden ohne SSL unverschlüsselt übertragen
* viele Nutzer erlauben keine Cookies oder löschen sie am Sitzungsende