ECMAScript

Ja, aus *JavaScript* wurde *ECMAScript*. Weitere Namen und Versionen siehe https://en.wikipedia.org/wiki/ECMAScript.

Und nicht nur die Bezeichnung hat sich geändert: neue Konzepte wurden hinzugefügt. Hier eine kleine Auswahl neuer Sprachelemente, die für das Arbeiten mit PWAs hilfreich sind und beim Lesen von Programmbeispielen für das Verständnis wichtig sind.

PWAs können auch in „älteren" Sprachversionen geschrieben werden; mit der moderneren Syntax werden die Programme leichter lesbar und übersichtlicher.

Vereinbaren von Variablen

Wird eine Variable mit (beispielsweise)

```
i = 3;
```

vereinbart und initialisiert, ist das eine globale Variable. Solange JavaScript-Programme schmucke 10-Zeiler waren, war das kein Problem. Aber: globale Variablen sind ein schlechter Programmierstil. Besser wäre *innerhalb* einer Funktion:

```
var i = 3;
```

Damit bleibt der Gültigkeitsbereich der Variablen auf die umschließende Funktion beschränkt. Mit

```
let i = 3;
```

wird (vereinfacht ausgedrückt) der Gültigkeitsbereich auf die umschließende Funktion oder den umschließenden Block beschränkt. Daher ist `let` für die Vereinbarung der Laufvariablen einer Schleife vorzuziehen.

Schleifen

Nehmen wir an, wir haben in einer Liste die Temperaturmittelwerte von Wien über 12 Monate aus 2018 gespeichert :

```
var t = [4.2, -0.7, 3.5, 15.8,
        18.7, 21.3, 22.7, 23.4,
        17.4, 13.0, 6.6, 3.0];
```

Oder auch mit

```
const t = [4.2, -0.7, 3.5, 15.8,
          18.7, 21.3, 22.7, 23.4,
          17.4, 13.0, 6.6, 3.0];
```

Die Berechnung des Mittelwertes ist mit

```
var mw = 0;
for (let i=0; i<t.length; i++)
   mw += t[i];
mw /= 12;
```

möglich, aber nicht sehr hübsch zu lesen. Nehmen wir an, dass das Jahr 12 Monate hat: ist seit 45 v.Chr. im julianischen und im gregorianischen Kalender üblich. Dann könnten wir statt `t.length` einfach 12 schreiben. (Wenn auch gestandene C-Programmierer nichts dabei finden … aber ich denke an alle, die das gerade einmal lernen müssen.)

Die `for`-Schleife samt Addition kann auch als

```
for (let i in t)
   mw += t[i];
```

geschrieben werden. Schon besser – schließlich weiß das Programm ja, wie groß das *Array* `t` ist.

Mit ECMAScript 6 ist die – meiner Ansicht nach – übersichtlichste Schreibweise

```
for (let v of t)
   mw += v;
```

möglich. Mit `of` wird nicht über die Indizes, sondern über die Werte (genauer: über die Namen der Eigenschaften) iteriert. Das ist es, was wir eigentlich haben wollten: *„Nimm einzeln jeden Wert v von t und addiere ihn zu mw."*)

Werden aber die Temperaturwerte als Objekt

```
const t = {jaenner:4.2, februar:-0.7,
   maerz:3.5, april:15.8, mai:18.7, :
   juni:21.3, juli:22.7, august:23.4,
   september:17.4, oktober:13.0,
   november:6.6, dezember:3.0};
```

gespeichert, ist trotzdem nur mehr die Variante

```
for (let i in t)
   mw += t[i];
```

möglich.

Zeichenketten

Wir wollen den errechneten Mittelwert samt Mittelwert in einem Satz ausgeben und speichern diesen Satz vorerst in einer Zeichenkette `mwtext`:

```
var mw = 12.4;
var mwtext =
   'Die mittlere Temperatur in Wien war 2018 '
   +mw+' Grad Celsius.';
```

Wenn da mehrere Variablenwerte und Texte gemischt werden, ist das keine sehr elegante Schreibweise. *ECMAScript* 6 führt *Template Strings* ein. Damit wird dieses Programmstück leichter zu schreiben und zu lesen:

```
var mw = 12.4;
var mwtext =
  `Die mittlere Temperatur in Wien war 2018 ${mw}
Grad Celsius.`;
```

Der String ist in diesem Fall durch Gravis (auch *„Backticks"*, Unicode *GRAVE ACCENT*U+0060) eingeschlossen. Statt einer Variablen kann in den geschwungenen Klammern auch ein Ausdruck stehen.

arrow functions

Mit ES6 wurde eine neue Syntax für die Vereinbarung von Funktionen zusätzlich eingeführt. Statt

```
var addiere = function(x, y) {
   return x + y;
};
```

ist nun auch

```
const addiere = (x, y) => { return x + y };
```

oder noch kürzer

```
const addiere = (x, y) => x + y;
```

zulässig. Sehr nützlich in anonymen Funktionen!

Anonyme Funktionen

Programme zur Auswertung von Webseiten müssen auf *Ereignisse (events)* reagieren: Maus-Klicks, Dateneingaben, Berührungen am Schirm usw. Daher wird in Programmen festgelegt, welche Funktion nach dem Eintreten eines Ereignisses aufgerufen wird. Diese Funktionen werden *Callback*-Funktionen genannt.

Hier ein einfaches Beispiel (ohne die Verwendung von Ereignissen):

Die Funktion `ersterSchritt` gibt den Text *"Erster Schritt"* aus. Erst wenn diese Aktion erfolgreich war, wird die Funktion `callback` aufgerufen, die ihrerseits über alert den Text *"Zweiter Schritt"* ausgibt.

```
function ersterSchritt(callback) {
   alert("Erster Schritt");
   callback();
}
function zweiterSchritt() {
  alert("Zweiter Schritt");
}
// Aufruf:
ersterSchritt(zweiterSchritt);
```

Es spricht nichts dagegen, benannte Funktionen (hier: `zweiterSchritt`) zu verwenden. Da aber diese Funktionen oft nur einmal verwendet werden, werden Programme mit anonymen Funktionen kürzer…

```
function ersterSchritt(callback) {
   alert("Erster Schritt");
   callback();
}

// Aufruf:
ersterSchritt(function() {
   alert("Zweiter Schritt");
});
```

… aber nicht unbedingt übersichtlicher. Vor allem, wenn Funktionen geschachtelt werden. Mit *arrow-functions* wird das Ganze in *ECMAScript* zu

```
function ersterSchritt(callback) {
   alert("Erster Schritt");
   callback();
}

// Aufruf:
ersterSchritt(() => {
   alert("Zweiter Schritt");
});
```

Objekte

ECMAScript-Objekte enthalten Paare von *Namen* und *Eigenschaften (properties)* oder *Namen* und *Methoden (methods)*, getrennt durch einen Doppelpunkt und eingeschlossen in geschwungenen Klammern.

```
var temperaturen = {
  ort: "Wien",
  jahr: 2018,
  t: [4.2, -0.7, 3.5, 15.8, 18.7, 21.3,
      22.7, 23.4, .4, 13.0, 6.6, 3.0],
  mittelwert: t => (t.reduce((sum,v)=>
        (sum + v)) / t.length)
}
```

Der Name in einem Paar kann auch variabel sein. Beispiel:

```
const ZEIT = "jahr";
var temperaturen = {
   ort: "Wien",
   [ZEIT]: 2018,
...
}
```

`ort`, `jahr` und `t` sind *Eigenschaften*, `mittelwert` ist eine *Methode*. Das Beispiel zeigt auch, dass Objekte verschiedene Datentypen enthalten können.

Wie gesagt – das ist eine kleine Auswahl von interessanten Sprachelementen der neuen *ECMAScript*-Versionen.

Links

https://medium.freecodecamp.org/write-less-do-more-with-javascript-es6-5fd4a8e50ee2