Einige Programmier-Basics

Am Beispiel von JavaScript

(als Ergänzung zu weiterem Lernmaterial zu nutzen)

INHALT

1. Programmiersprachen - im Allgemeinen

Computer können im Grunde nur einige sehr grundlegende Befehle ausführen: „Speichere eine Zahl an einem bestimmten Speicherplatz!“, „Addiere den Inhalt eines bestimmten Speicherplatzes zum Inhalt eines anderen Speicherplatzes!“ oder „Lasse einen bestimmten Lichtpunkt am Display leuchten!“

Da es schnell unübersichtlich wird, mit solchen Befehlen komplexere Programme zu erstellen, wurden höhere Programmiersprachen entwickelt, die für Menschen leichter zu verstehen sind.
Bevor der Computer ein Programm abarbeitet bzw. ausführt, wird es automatisch in eine für den Computer interpretierbare Sprache übersetzt. Die Befehle auf dieser "untersten" Ebene, also auf der Ebene, die der Maschine am nächsten und dem Menschen am fernsten ist, bestehen letztlich nur aus Kombinationen von Nullen und Einsen. Auf dieser Ebene kann der Computer die Befehle sofort ausführen.

Inzwischen gibt es sehr viele verschiedene Programmiersprachen mit verschiedenen Stärken und Einsatzzwecken. Die grundlegenden Konzepte der Programmierung finden sich jedoch in allen Sprachen wieder, also Konzepte wie insbesondere Variablen, Funktionen, bedingte Anweisungen, Schleifen usw.

2. JavaScript im Kontext von HTML und CSS

HTML ist keine Programmiersprache sondern eine Auszeichnungssprache (Markup Language) zur Strukturierung von Hypertext-Dokumenten. Diese HTML-Dokumente sind die Grundlage des World Wide Web und können von Webbrowsern dargestellt werden. HTML dient dazu, einen Text semantisch zu strukturieren, nicht aber ihn visuell zu gestalten. Zur visuellen Gestaltung und Formatierung wird CSS benutzt.

HTML und CSS sind **statisch**, d.h. sie bieten keine Möglichkeit, HTML- oder CSS-Code zu verändern, nachdem die Seite geladen wurde. JavaScript ermöglicht es jedoch, eine **dynamische** Webseite zu gestalten, die Interaktionsmöglichkeiten bietet und Seiteninhalte dynamisch (nach)laden und verändern kann. Mit JavaScript können HTML- und CSS-Elemente einer Webseite verändert werden.

JavaScript ist eine Skriptsprache, die von Browsern interpretiert und ausgeführt werden kann. Damit hat JavaScript den Vorteil, unabhängig vom jeweiligen Betriebssystem zu sein.

3. Konzepte von Programmiersprachen am Beispiel von JavaScript

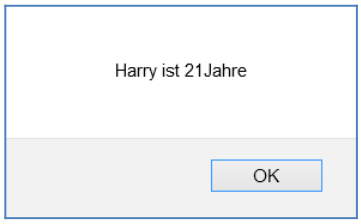
Variablen

In der Programmierung ist eine Variable ein Behälter für einen Wert. Solche Werte können Zahlen sein (number) oder Zeichenketten (string), aber auch Objekte oder Felder und andere Datentypen. Wie bei Schließfächern oder Postfächern können die Inhalte (Werte) der Variablen sich im Laufe der Abarbeitung eines Programms ändern. Über den Namen der Variablen kann auf ihren Inhalt zugegriffen werden. Wir können also Variablen nutzen, um darin einen Wert abzulegen, zu verändern und später wieder zu benutzen.

In JavaScript wird eine Variable in der Regel mit dem Schlüsselwort **`let`** definiert.
(Das Schlüsselwort **`var`** wurde 2015 durch die beiden Schlüsselwörter **`let`** und **`const`** abgelöst. Schlüsselwörter gehören zum vordefinierten Vokabular einer Programmiersprache.)

Beispiele:

```
//Definition von Variablen:
let alter;
let name;
// Danach kann einer Variablen ein Wert zugewiesen werden:
alter = 21;
name = "Harry";
//... und so kann die Variable z.B. benutzt werden:
alert (name + " ist " + alter + "Jahre");
```



Mehrere Variablen können auch zusammen definiert werden:

```
let alter, name;
```

Definition und Wertzuweisung können auch in einem Befehl geschrieben werden:

```
let alter = 21;
let name = "Nadine";
```

Kommentare in JavaScript:

```
//  Mit 2 Schrägstrichen wird erreicht,
```

dass der Text, der dahinter steht, bis zum Ende der Zeile nicht vom Computer interpretiert wird. Es ist ein Kommentar für menschliche Leser:innen, mit dem insbesondere vermerkt wird, was mit dem entsprechenden Code-Abschnitt erreicht wird.
Will man nicht nur eine einzelne Zeile "auskommentieren", sondern einen ganzen Bereich, nutzt man /* und */, siehe:

```
/*
Dies ist der Kommentar, der auch mehrere
Zeilen lang sein kann.
*/
```

Types – Datentypen

Variablen können verschiedene Datentypen beinhalten. Diese Datentypen sind in allen Programmiersprachen ähnlich, z.B. Zeichenkette (string), logischer Wert (Boolean), numerischer Datentyp (number), Objekt, Array usw.

In JavaScript muss der Datentyp nicht - wie in vielen anderen Programmiersprachen - vorab deklariert werden, sondern ergibt sich dynamisch aus dem Zusammenhang. Bei Bedarf kann der Datentyp sogar während der Ausführung des Programms automatisch konvertiert werden.

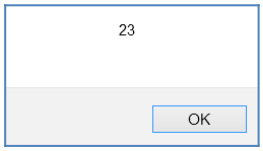
```
let alter = 16;        // Zahlen werden einfach hinter das „=“ geschrieben.
let lastName = "Johnson";
// Strings werden in einfache oder doppelte Hochkommas eingebettet.
```

Setzt man jedoch eine Zahl in Hochkommas, wird die Zahl als String gespeichert.

Mit einem String kann nicht gerechnet werden, deshalb werden im folgenden Code-Beispiel die zwei Werte beim Interpretieren des Programms einfach hintereinander geschrieben. Das "+" wird nicht als Addition sondern als Konkatenation (Verknüpfung) interpretiert:

```
let x = '2';
alert(x + 3);
```

Es wird 23 ausgegeben:



String

Ein String ist eine Zeichenkette, die aus 0 oder mehr Zeichen (character) besteht und von einfachen oder doppelten Hochkommas umgeben wird.

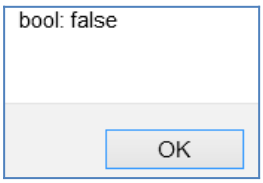
Number

Number ist ein numerischer Datentyp. Nummern werden in anderen Programmiersprachen oft noch weiter unterteilt, insbesondere in ganze Zahlen (integer) und Fließkommazahlen (float; diese enthalten einen Dezimalpunkt, kein Komma!).

Boolean

Ein Logischer Wert (Boolean) kann nur einen von zwei Werten annehmen: *true* oder *false*. Dies sind keine Zeichenketten. Sie werden also nicht in Anführungszeichen gesetzt.

```
let a=3;
let b=5;
let bool=(a==b);
alert("bool: "+bool);
```



Logische Operatoren in JavaScript:	
größer: >	kleiner: <
größer oder gleich: >=	kleiner oder gleich: <=
gleich: ==	ungleich: !=

Objekte

Zusammengesetzte Datentypen (auch als *komplexe Datentypen* bezeichnet) können mehr als einen Wert speichern. Sie sind Sammelbehälter für Daten der unterschiedlichsten Art. Zu den zusammengesetzten Datentypen zählen Felder (arrays) und Objekte (objects).

Ein Objekt beginnt mit `{` und endet mit `}`. Die in einem Objekt enthaltenen Daten werden durch Kommas getrennt und in Form von Schlüssel-Wertepaaren eingetragen. Schlüssel und Wert sind wiederum getrennt durch einen Doppelpunkt, also in der Form `Schlüssel : Wert`.

Der Schlüssel ist eine Zeichenkette und muss innerhalb eines Objekts eindeutig sein. Dadurch spielt es keine Rolle, in welcher Reihenfolge die Daten in einem Objekt aufgelistet sind. Dem Schlüssel wird ein Wert von beliebigem Datentyp zugeordnet.

Beispiele:

```
let kreis= {
        x: 80,
        y: 60,
        r: 50,
        fillColor: "yellow",
        stroke: "grey",
        strokeWidth: 10,
};
```

```
let bike= {
          type: "citybike",
          model: "27T",
          weight: "18kg",
          color: "red",
          verfuegbar: true
};
```

Es gibt auch leere Objekte: `let` leeresObjekt= {};

Darüberhinaus stellt auch der Client-Browser Objekte zur Verfügung, z.B. Window und Document (wie in document. getElementById("myId").innerHTML
(siehe dazu auch das Document Object Model (DOM)).

Punktnotation

Auf die Objekteigenschaften kann über deren Namen mit der **Punkt-Notation** zugegriffen werden: objektname.eigenschaftsname , z.B.

```
kreis.fillColor
kreis.x
bike.verfuegbar
```

Entsprechend kann ein Wert auch neu gesetzt werden, z.B.

```
kreis.r=30;
```

Objektmethoden

Aktionen, die mit einem Objekt oder seinen Eigenschaften gemacht werden können, nennt man Methoden. Methoden werden als Funktionsdefinition in der Liste der Objekteigenschaften gespeichert, also z.B.

```
let kreis= {
     x: 80,
     y: 60,
     r: 50,
     fillColor: "yellow",
     stroke: "grey",
     strokeWidth: 10,
     resize: function(radiusneu) {
              kreis.r += radiusneu;
             }
};
```

```
let bike= {
  type: "citybike",
  model: "27T",
  weight: "18kg",
  color: "red",
  verfuegbar: true,
  vermieten: function() {
              bike.verfuegbar = false;
             }
```

Auch auf Methoden von Objekten kann über deren Namen mit der **Punkt-Notation** zugegriffen werden: objektname.methodenname() , z.B.

```
bike.vermieten();  //bike.verfuegbar wird auf "false" gesetzt
kreis.resize(-10); //kreis.r wird um 10 verringert
meinMarker.bindPopup(); //an den Marker wird ein Popup gebunden
meinMarker.addTo(mymap);  //der Marker wird auf der Karte angezeigt
```

oder auch hintereinander:

```
meinMarker.bindPopup().addTo(mymap);
```

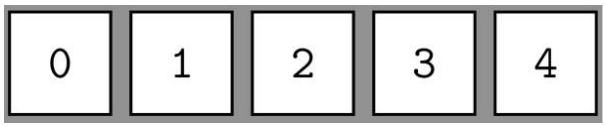
Arrays

Ein Feld (Array) enthält eine durch Kommas getrennte, geordnete Liste von Elementen gleichen oder verschiedenen Typs. Es beginnt mit `[` und endet mit `]`, z.B.

`let` wochentage = ["Montag", "Dienstag", "Mittwoch", "Donnerstag", "Freitag", "Samstag", "Sonntag"];

Ein Array kann auch leer sein: **`let`** leeresFeld = [];

Die Adresse bzw. der Index des ersten Feld-Elementes ist 0, des zweiten 1 usw.



Um auf ein Element des arrays zuzugreifen, adressiert man es über den in eckigen Klammern stehenden Index, wie z.B. `wochentage[0]` , womit man auf das erste Element zugreift.

```
// Der Variablen day wird der Wert "Sonntag" zugewiesen:
let day = wochentage[6];
```

Andere Beispiele:

```
let geoposition = [53.54, 8.56];
let latitude = geoposition[0];
let longitude= geoposition[1];

let teammitglieder=["Nadine", "Pascal", "Leonie"];
```

Mit der Methode

```
teammitglieder.push("Meret");
```

kann man ein neues Element am Ende des Array einfügen, und mit

```
let lastElement = teammitglieder.pop();
```

wird das letzte Element aus dem Array entfernt, und - in diesem Fall - einer Variablen zugewiesen.

Die Länge eines Feldes wird mit der vordefinierten Eigenschaft "length" ermittelt oder gesetzt:

```
let laenge=wochentage.length;   //Wert von laenge ist 7.
```

Auf das letzte Element des Feldes kann also so zugegriffen werden:

```
let day=wochentage[wochentage.length-1];
//Wert von day ist "Sonntag".
```

Hier wird auch die **Punktnotation** verwendet:
Vor dem Punkt steht der Name des arrays, hinter dem Punkt die Eigenschaft.

Die in einem Feld enthaltenen Daten können wiederum auch Felder sein, z.B.

```
// Liste mit Geopositionen:
let liste = [
    [53.5405, 8.58356],
    [53.53949, 8.58271],
    [53.53988, 8.58376],
];
```

Funktionen

Eine Funktion ist ein Block von Code, mit dem eine bestimmte Aufgabe ausgeführt wird. Dieser Code wird, wie sonst auch, von oben nach unten ausgeführt.

Jede Programmiersprache verfügt über zahlreiche vorgefertigte Funktionen, in JavaScript gibt es z.B. die Funktionen **alert**(), **prompt**(), **replace**(), **length**(), **toString**(), **toLowerCase**() und viele mehr.

Eine Funktion wird mit dem Schlüsselwort **`function`** definiert, gefolgt von einem Namen für die Funktion, die aufgerufen wird, gefolgt von runden Klammern. In den Klammern können ein oder mehrere durch Kommas getrennte Parameternamen stehen, falls an die Funktion etwas übergeben werden soll. Dahinter folgt der von geschweiften Klammern umgebene Code, der nach dem Aufruf der Funktion ausgeführt wird, z.B.

```
function  multipliziere(x, y) {
   return x * y;
}
```

Funktion in OSM/Leaflet:

```
function zoomfaktorAnzeigen() {
      let z = mymap.getZoom();
      alert("Zoomfaktor: " + z);
    }
```

Die Definition der Funktion wird von sich aus nicht ausgeführt. Damit eine Funktion ausgeführt wird, muss sie aufgerufen werden. Ein Funktionsaufruf besteht immer aus dem Namen der Funktion gefolgt von runden Klammern, also z.B.

```
 /* Nach dem Schlüsselwort folgt der Name der Funktion und in runden
Klammern Argumente, d.h. Werte für die Parameter, falls an die Funktion
etwas übergeben werden soll:*/

multipliziere(23, 37);
```

Funktionen können mit dem Schlüsselwort `return` einen Wert an den "Aufrufer" zurückgeben, z.B.

```
// x wird der return-Wert der Funktion multipliziere zugewiesen, also 12:
let x = multipliziere (4, 3);
```

Eine Funktion kann auch über ein Ereignis aufgerufen werden, also z.B. wenn jemand auf einen Button klickt:

```
<button onclick='zoomfaktorAnzeigen()'>Zoomfaktor anzeigen</button>
```

4. Algorithmische Kontrollstrukturen

Bedingte Anweisungen - if-Anweisungen

Um auszudrücken, dass ein Code-Block nur ausgeführt werden soll, wenn eine bestimmte Bedingung zutrifft, benutzt man bedingte Anweisungen, die in der Regel mit dem Schlüsselwort **`if`** und einer Bedingung eingeleitet werden. Will man auch eine Alternative ausdrücken benutzt man **`else`** `{...}`

In JavaScript benutzt man die folgende Syntax für diese Anweisungen:

```
if (condition)  {
     ... //Code-Block, der ausgeführt werden soll
}
```

also z.B.

```
if (a==0) {
     alert ("Division durch 0 ist nicht erlaubt.");
}
else {
    let ergebnis = 100/a;
    alert ("Das Ergebnis der Division ist " + ergebnis);
}
```

Schleifen

Um zu erreichen, dass Anweisungen in einem Programm wiederholt durchgeführt werden, können Schleifen verwendet werden. Die zwei wichtigsten Varianten sind **`while`**-Schleifen und **`for`**-Schleifen (Zählschleifen). Jede Schleife hat eine Durchlauf-Bedingung, damit sie nicht endlos läuft. D.h. vor einem Durchlauf wird eine Bedingung geprüft. Ist die Bedingung erfüllt, werden die Befehle innerhalb der Schleife ausgeführt. Ist die Bedingung nicht (mehr) erfüllt, wird die Schleife verlassen.

while-Schleifen

`while`-Schleifen wiederholen etwas solange, wie eine Bedingung zutrifft. Die Bedingung wird vor Eintritt in die Schleife geprüft. Wenn diese true ist, dann wird die Schleife durchlaufen, wenn false, dann nicht. Die **`while`**-Schleife wird so lange durchlaufen, wie die Bedingung erfüllt ist.

Beispiel in JavaScript:
allgemeine Syntax:

```
while (condition)  {
     ... // Schleifenrumpf mit Code, der ausgeführt werden soll
}
```

Beispiel:

```
let i = 100;
while (i > 10) {
  document.write("<br>Der Wert von i ist " + i);
  i=i/2;
}
```

for-Schleifen (Zählschleifen)

Zählschleifen werden benutzt, wenn von vorneherein feststeht, wie oft oder bis zu welchem Maximalwert eine Schleife durchlaufen werden soll.

Die Schreibweise "i++" wird oft als Verkürzung für "i=i+1;" benutzt. Ebenso ist i+=3 eine verkürzte Schreibweise für "i=i+3;"

Beispiel in JavaScript:

```
for (i = 0; i < 5; i++) {
  document.write("Die Zahl ist ", i , "<br>")
}
```

Ergebnis:

Die Zahl ist 0
Die Zahl ist 1
Die Zahl ist 2
Die Zahl ist 3
Die Zahl ist 4

Eine **for**-Schleife hat die folgende Syntax

```
for (Initialisierung einer Zählvariablen; Bedingung; Verändern der
Zählvariablen) {
      ... //Schleifenrumpf mit Code, der ausgeführt werden soll
}
```

Die Zählvariable wird vor dem ersten Schleifendurchlauf initialisiert. Mit der Bedingung wird entschieden, ob die Schleife überhaupt durchlaufen wird. Das Verändern der Zählvariablen geschieht nach der Ausführung des Codes im Schleifenrumpf.

5. Software-Bibliotheken

Für einige Themen und Aufgabenstellungen, die immer wieder bei der Programmierung vorkommen, werden sogenannte Software-Bibliotheken entwickelt. Diese Bibliotheken sind eine Sammlung von vorgefertigten Programmbausteinen (insbesondere Funktionen und - je nach Programmiersprache - Klassen), die im eigenen Programm genutzt werden können. Dazu muss man die Bibliothek, also die Datei(en), in der sie gespeichert ist, einbinden (importieren).

Eine JavaScript-Bibliothek bindet man z.B. mit folgendem HTML-Code auf einer Webseite ein:

```
<script src="https://....js"></script>
```

Siehe unten das Beispiel zum Einbinden der Leaflet-Bibliothek.
Die Bausteine der Bibliothek, also insbesondere Funktionen, können dann im eigenen Programm aufgerufen und genutzt werden.

Leaflet - Bibliothek

Leaflet ist eine Open-Source-JavaScript-Bibliothek, mit der interaktive Karten gestaltet werden, siehe https://leafletjs.com/ .
Das Einbinden der Leaflet-Bibliothek erreicht man z.B. durch folgenden HTML-Code:

```
<script type="text/javascript"
src="https://informatik.hs-bremerhaven.de/leaflet/leaflet.js"></script>
```

Auf das zugehörige Stylesheet wird durch das link-Tag referenziert:

```
<link rel="stylesheet" type="text/css" href="https://informatik.hs-
bremerhaven.de/leaflet/leaflet.css" />
```

Bootstrap - Framework

Bootstrap ist ein Open-Source-JavaScript-Framework zur Gestaltung von Webseiten auf der Basis von HTML, CSS und JavaScript, siehe https://getbootstrap.com/.

Während man bei der Nutzung einer Bibliothek Softwarebausteine aus einer Sammlung an passenden Stellen im eigenen Programm wiederverwendet, liefert ein Framework ein ganzes Programmiergerüst. Die Bootstrap-Gestaltungsvorlagen geben also quasi die Architektur der zu gestaltenden Website vor und bestimmen darüber, wie und wo Softwarebausteine im Rahmen dieser Architektur eingesetzt werden können.