***Reintroducing JavaSript***

***Datentypen:***

* Numbers
  + Es existiert ein Objekt **Math** für die Anwendung von mathematischen Funktionen und Konstanten, welches in der Aufzählung oben noch nicht genannt wurde:

**Math.sin(3.5);**

var area = Math.PI \* r \* r;

* + **parseInt(string, Basic)** wandelt einen String in einen Integer um. Gibt man die Basis nicht an, könnte dies in älteren Browsern zu überraschenden Ergebnissen führen, da man früher das Zahlensystem mit einem Präfix markierte. "0" stand für eine Oktalzahl, "0x" für eine Hexadezimalzahl.

Auf die gleiche Weise lassen sich auch Gleitkommazahlen mit Hilfe der Funktion [**parseFloat()**](https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Global_Objects/parseFloat) konvertieren, welche - anders als die verwandte Funktion [parseInt()](https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Global_Objects/parseInt) - immer mit der Basis 10 arbeitet.

Falls die Zahl keine Nummer ist, dann wird NaN zurückgegeben.

* + **isNaN(NaN);** // true überprüft ob es eine Zahl ist oder nicht. Gibt boolean zurück.
  + **isFinite(NaN);** // false überprüft ob die Zahl endlich ist. false wenn der Wert positiv [Infinity](https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Global_Objects/Infinity), negativ [Infinity](https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Global_Objects/Infinity) or [NaN](https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Global_Objects/NaN) ist, andernfalls true.
* String (Zeichenkette)
  + Es gibt bei JavaScript keinen speziellen Datentyp für einzelne Zeichen, stattdessen sollte einfach ein String mit einer Länge von 1 verwendet werden.
  + **"hello".length; // 5** Um die Anzahl der Zeichen eines Strings herauszufinden, greift man auf die Eigenschaft [length](https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Global_Objects/Array/length) zu.
  + **"hello".charAt(0); // "h"**

**"hello, world".replace("hello", "goodbye"); // "goodbye, world"**

**"hello".toUpperCase(); // "HELLO"**

* Weitere Datentypen und Werte
  + **Null** representiert einen „nicht Wert“
  + **Undefined** ist ein Wert vom Typ ‚undefined‘, welcher für einen nicht initialisierten Wert steht – also anzeigt das kein Wert zugewiesen ist. Bei JavaScript ist es möglich, eine Variable zu deklarieren, ohne sie zu initialisieren, ihr also einen Wert zuzuweisen - genau dann ist ihr Typ undefined (undefiniert).
  + **Boolean()** Jeder Wert kann nach den folgenden Regeln zu einem booleschen Wert konvertiert werden:
    - false , 0, leere Strings (""), NaN , null , und undefined werden zu false
    - Alle anderen Werte werden true

Aus diesem Grund ist oft von "true Werten" und "false Werten" die Rede, womit Werte gemeint sind, die zu true oder false werden, nachdem sie zu booleschen Werten konvertiert wurden.

* Variablen
  + Neue Variablen werden in JavaScript mit dem Schlüsselwort „var“ deklariert.

Deklariert man eine Variable, ohne ihr einen Wert zuzuweisen, ist ihr Typ undefined.

Ein sehr wichtiger Unterschied zu anderen Programmiersprachen besteht darin, dass bei JavaScript Blöcke keine Sichtbarkeitsbereiche festlegen. Wird z.B. eine Variable mit dem var-Schlüsselwort in einer Kontrollstruktur (z.B. if-Anweisung) definiert, so ist diese Variable innerhalb der ganzen Funktion sichtbar - und nicht nur innerhalb des if-Blocks. Mit der ECMAScript Edition 6 ist es mithilfe der neuen Schlüsselwörter [let](https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Statements/let) und [const](https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Statements/const) möglich, Variablen zu erstellen, die an den jeweiligen Block gebunden sind.

* Operatoren
  + Beim Vergleichsoperator „==“ werden die Typen nicht beachtet und es kommen dabei oft komische Ergebnisse heraus: 123==“123“ //true

Damit die Typen beachtet werden muss „===“ verwendet werden:

123===“123“ //false

Das gleiche gilt bei „!=“ und „!==“

***Kontrollstrukturen:***

* + **If:**

var name = "kittens";

if (name == "puppies") {

name += "!";

} else if (name == "kittens") {

name += "!!";

} else {

name = "!" + name;

}

* + **While, do-While Schleife:**

while (true) {

// Endlosschleife!

}

var input;

do {

input = get\_input();

} while (inputIsNotValid(input))

* + **For-Schleife:**

for (var i = 0; i < 5; i++) {

// Wird 5-Mal ausgeführt

}

* + **Switch-Case:**

switch(action) {

case 'draw':

drawit();

break;

case 'eat':

eatit();

break;

default:

doNothing();

}

***Objekte:***

* Sind wie Hashtabellen bei C und C++
* Der "Name"-Teil ist ein JavaScript-String, während als Wert ein beliebiger für JavaScript gültiger Wert gesetzt werden kann - andere Objekte eingeschlossen. Dies ermöglicht die die Zusammenstellung von beliebig komplexen Datenstrukturen.
* **var obj = new Object();** // Initialisierung eines Objekts
* **var obj = {};** // Initialisierung eines Objekts

**var obj = {**

**name: "Carrot",**

**"for": "Max",**

**details: {**

**color: "orange",**

**size: 12**

**}**

**}**

**obj.details.color; // orange**

**obj["details"]["size"]; // 12**

* **obj.for = "Simon"; // Syntaxfehler, da 'for' ein reserviertes Codewort ist**

**obj["for"] = "Simon"; // funktioniert**

***Arrays:***

* alte Syntax für die Erstellung:

**var a = new Array();**

**a[0] = "dog";**

**a[1] = "cat";**

**a[2] = "hen";**

**a.length; // 3**

* **var a = ["dog", "cat", "hen"];**

**a.length; // 3**

* **var a = ["dog", "cat", "hen"];**

**a[100] = "fox";**

**a.length; // 101**

* **typeof a[90]; // undefined**
* **["dog", "cat", "hen"].forEach(function(currentValue, index, array) {**

**// Verarbeite currentValue oder array[index]**

});

* Das Hinzufügen eines Elements zu einem Array funktioniert folgendermaßen:

**a.push(item);**

| **Methodenname** | **Beschreibung** |
| --- | --- |
| * a.toString() | * Gibt einen String zurück, der aus dem Ergebnis von toString() eines jeden Elements, mit Komma getrennt, besteht. |
| * a.toLocaleString() | * Wie toString() , nur wird anstelle von toString() toLocaleString() aufgerufen. |
| * a.concat(item1[, item2[, ...[, itemN]]]) | * Erstellt eine Kopie von a, fügt dieser die übergebenen   Elemente hinzu und gibt diese Kopie zurück. |
| * a.join(sep) | * Wandelt das Array in einen String um. Die Elemente   werden durch sep getrennt. |
| * a.pop() | * Entfernt das letzte Element und gibt es zurück. |
| * a.push(item1, ..., itemN) | * Push fügt eines oder mehrere Elemente am Ende hinzu. |
| * a.reverse() | * Kehrt die Reihenfolge des Arrays um. |
| * a.shift() | * Entfernt das erste Element und gibt es zurück. |
| * a.slice(start, end) | * Gibt einen Teil eines Arrays zurück. |
| * a.sort([cmpfn]) | * Sortiert das Array mithilfe der übergebenen   Sortierfunktion cmpfn. |
| * a.splice(start, delcount[, item1[, ...[, itemN]]]) | * Modifiziert ein Array, indem ein Teil gelöscht und   durch mehrere Elemente ersetzt wird. |
| * a.unshift(item1[, item2[, ...[, itemN]]]) | * Fügt Elemente am Anfang hinzu. |

***Funktionen:***

* Aufbau einer Funktion:

**function add(x, y) {**

**var total = x + y;**

**return total;**

**}**

* Es wird mit return ein Wert zurückgegeben ansonsten gibt JavaScript undefined zurück
* Wenn man das obige **add()** ohne Parameter aufruft sind x und y undefined
* Es können auch mehr Parameter als erwartet übergeben werden: **add(1, 2, 3, 4)**
* Die Funktion hat ein Arrayähnliches Objekt namens arguments wo alle Übergebenen Argumente enthalten sind
* JavaScript Funktionen sind Objekte
* Es gibt auch sogenannte anonyme Funktionen:

**var avg = function() {**

**var sum = 0;**

**for (var i = 0, j = arguments.length; i < j; i++) {**

**sum += arguments[i];**

**}**

**return sum / arguments.length;**

**}**

* Rekursive Aufrufe sind wie in C# möglich
* Um anonyme Funktionen aufrufen zu können kann man ihnen einen Namen geben. Dies nennt man "named IIFEs" ( Immediately Invoked Function Expressions).

**var charsInBody = (function counter(elm) {**

**if (elm.nodeType == 3) { // TEXT\_NODE**

**return elm.nodeValue.length;**

**}**

**var count = 0;**

**for (var i = 0, child; child = elm.childNodes[i]; i++) {**

**count += counter(child);**

**}**

**return count;**

**})(document.body);**

Beachte, dass JavaScript-Funktionen ebenfalls Objete sind - wie alles andere in JavaScript - und man kann Attribute ändern und hinzufügen, wie wir es im Objekte-Abschnitt gesehen haben.