DANIEL MIKESCH

JAVASCRIPT 101

ARRAYS

- sind Variablen die mehrere Werte speichern
- jedes Element hat einen eindeutigen Index (Beginnend bei 0!)
- ein Array-Element kann ein weiteres Array beinhalten (mehrdimensionale Arrays)

```
var myArrayA = ["rot", "gruen", "gelb", "blau", “schwarz"];
var myArrayB = new Array("Merkur", "Venus", “Erde");

var multiDimensionalA = ["X", "Y", 1, 4, ["Z", 3], “Z"];
var multiDimensionalB = new Array(“X", 4, myArrayB, “Z");
```

ARRAYS

- Zugriff auf Elemente mit [index]

```
var myArray = ["A", "B", 1, true];
var multiDimensional = ["X", "Y", 1, 4, ["Z", 3], "Z"];

console.log(myArray[0]); // A

console.log(myArray[2]); // 1
myArray[2] = 42;
console.log(myArray[2]); // 42

console.log(myArray[myArray.length - 1]); // true

console.log(multiDimensional[1]); //Y
console.log(multiDimensional[4][0]); //Z
```

ARRAY LENGTH – ITERATING TROUGH AN ARRAY

- die aktuelle Länge eines Arrays kann aus dem Attribut length ausgelesen werden myArray.length
- dies kann in kann in Kombination mit einer for-Schleife zum durchgehen aller Elemente benutzt werden

```
var myArray = ["A", "B", 1, true];
for(var i = 0; i < myArray.length; i++) {
   console.log(i + ':' + myArray[i]);
}
//0: A
//1: B
//2: 1
//3: true
```

ARRAY METHODEN

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array

- myArray.indexOf(vergleichsElement[, offset])
- myArray.push(newElement[, newElement, ...])
- myArray.pop( )
- myArray.shift( )
- myArray.unshift(newElement[, newElement, ...])
- myArray.reverse( )
- myArray.sort([vergleichsfunktion])
- myArray.join(trenner)

myArray.indexOf(vergleichsElement[, offset])

Ähnlich wie myString.indexOf - Durchsucht das Array und returnt die Position des ersten Treffers oder -1 wenn kein gleiches Element gefunden werden kann.
Mit den zweiten Argument kann optional ein Offset angegeben werden wo zu suchen begonnen werden soll. Die Element die einen kleineren Index haben werden dann ignoriert.

```
var myArray = ["red", "green", "blue"];

console.log( myArray.indexOf("red") ); // 0
console.log( myArray.indexOf("red", 1) ); // -1
console.log( myArray.indexOf("blue") ); // 2
console.log( myArray.indexOf("yellow") ); // -1
```

myArray.push(newElement[, newElement, ...])

Hängt ein oder mehrere Elemente an ein Array an

```
var myArray = ["red", "green", "blue"];

console.log(myArray); // ["red","green","blue"]
myArray.push("yellow");
console.log(myArray); // ["red","green","blue","yellow"]
```

myArray.pop( )

Entfernt und returnt das letzte Element des Arrays

```
var myArray = ["red", "green", “blue"];

console.log(myArray); // ["red","green","blue"]
var tmp = myArray.pop();
console.log(tmp); // "blue"
console.log(myArray); // ["red","green"]
```

myArray.shift( )

Entfernt und returnt das erste Element des Arrays

```
var myArray = ["red", "green", "blue"];

console.log(myArray); // ["red","green","blue"]
var tmp = myArray.shift();
console.log(tmp); // "red"
console.log(myArray); // ["green","blue"]
```

myArray.unshift(newElement[, newElement, ...])

Fügt Vorne ein oder mehrere neue Element hinzu

```
var myArray = ["red", "green", "blue"];

console.log(myArray); // ["red","green","blue"]
myArray.unshift("yellow");
console.log(myArray); // ["yellow","red","green","blue"]
```

myArray.reverse( )

Invertiert die Reihenfolge der Elemente

```
var myArray = ["red", "green", "blue"];

console.log(myArray); // ["red","green","blue"]
myArray.reverse();
console.log(myArray); // ["blue","green","red"]
```

myArray.sort( )

Sortiert ein Array

```
var myArray = ["red", "green", "yellow", "blue"];

console.log(myArray); // [“red”,”green","yellow","blue"]
myArray.sort();
console.log(myArray); // [“blue”,”green","red","yellow"]
```

myArray.sort([vergleichsfunktion])

Sortiert ein Array mit Hilfe einer Vergleichsfunktion

```
var myArray = [1, 0, 4, 2, 9];

console.log(myArray); // [1,0,4,2,9]
myArray.sort(function(a, b){
    if(a > b) {
        return 1; //b vor a
    } else if (a < b) {
        return -1; //a vor b
    }
    return 0; //kein platztausch notwendig
});
console.log(myArray); // [0,1,2,4,9]
```

myArray.join(trenner)

Erzeugt einen String in dem die Elemente aneinander gehängt werden

```
var myArray = ["red", "green", "blue"];

console.log(myArray); // ["red","green","blue"]
var str = myArray.join(" - ");
console.log(str); // "red - green - blue"
```

NOCH MEHR ARRAY METHODEN

- myArray.filter(filterFunktion)
- myArray.slice(start [, ende])
- myArray.splice(start, löschAnzahl [, item, …])
- myArray.concat(value [, value, …])
- myArray.fill(value[, start [, end]])

Details und weiter Methoden können in der JavaScript Referenz gefunden werden

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array

FUNKTIONEN

- Funktionen erlauben es, Code zu definieren der erst beim Aufruf der Funktion ausgeführt wird.
- Deklaration mit dem Schlüsselwort function. Aufrufen mit ().

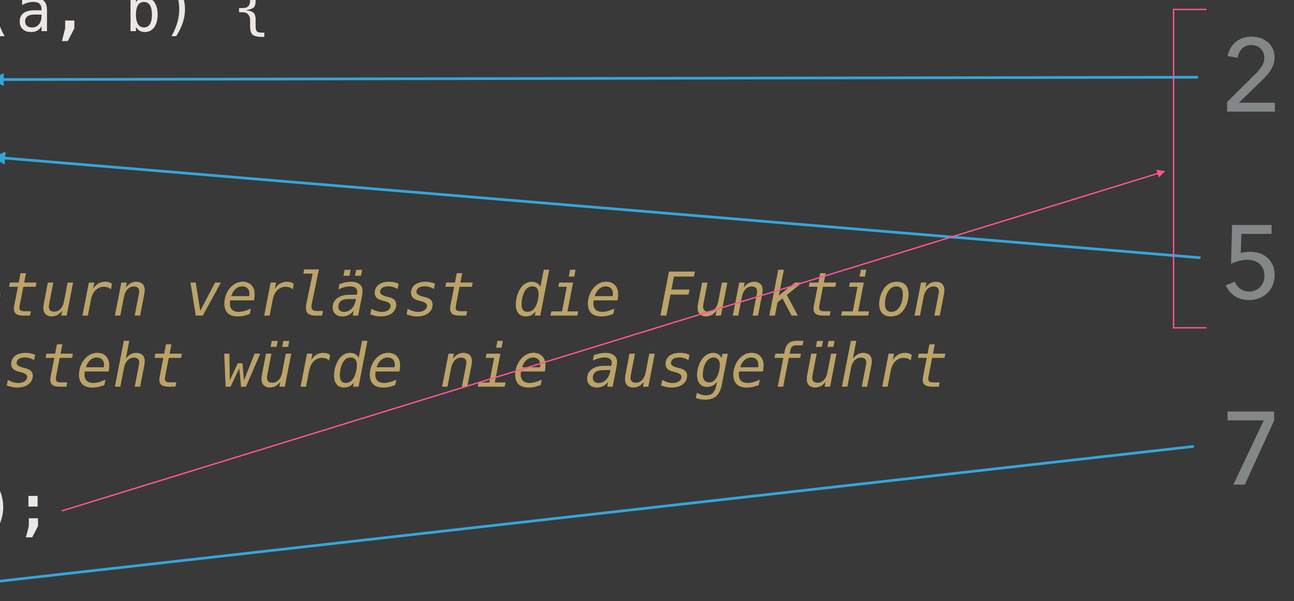
```
function greetUser(_name) {
   console.log("Hi " + _name);
}
greetUser("Tom"); //Hi Tom
greetUser("Sara"); //Hi Sara
```

PARAMETER UND RETURN

```
var foo = function(a, b) {
   console.log(a);
   console.log(b);
   var sum = a + b;
   return sum; //return verlässt die Funktion
   //Code der hier steht würde nie ausgeführt
};
var ret = foo(2, 5);
console.log(ret);
```

PARAMETER UND RETURN

```
var foo = function(a, b) {
   console.log(a);
   console.log(b);
   var sum = a + b;
   return sum; //return verlässt die Funktion
   //Code der hier steht würde nie ausgeführt
};
var ret = foo(2, 5);
console.log(ret);
```

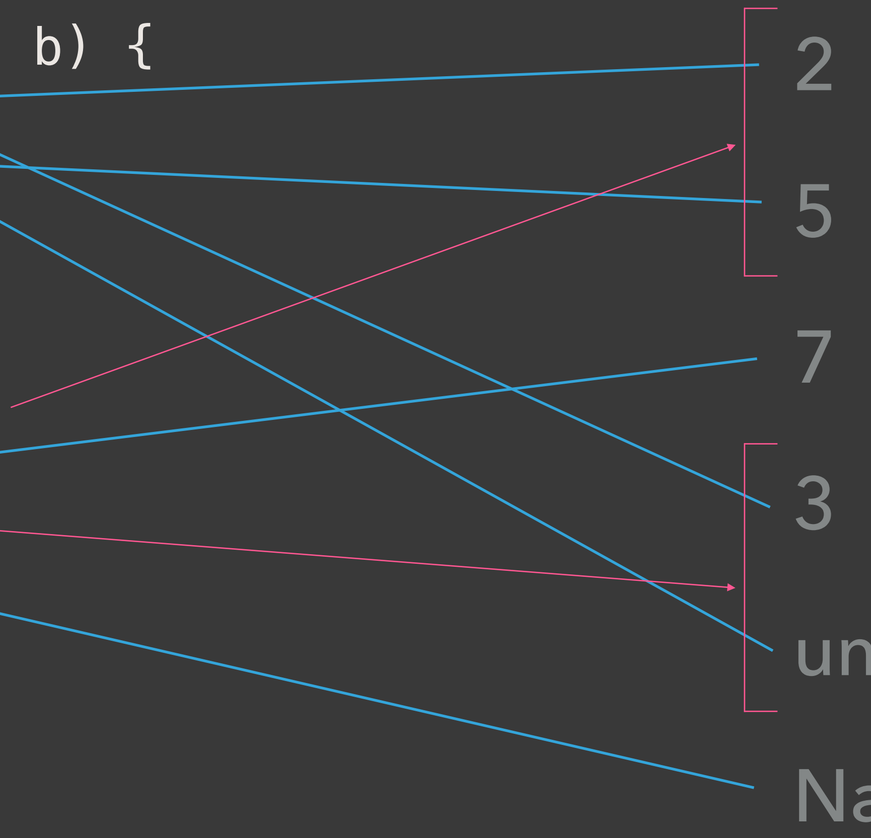


PARAMETER UND RETURN

```
var foo = function(a, b) {
    console.log(a);
    console.log(b);
    var sum = a + b;
    return sum;
};
var ret1 = foo(2, 5);
console.log(ret1);
var ret2 = foo(3);
console.log(ret2);
```

PARAMETER UND RETURN

```
var foo = function(a, b) {
    console.log(a);
    console.log(b);
    var sum = a + b;
    return sum;
};
var ret1 = foo(2, 5);
console.log(ret1);
var ret2 = foo(3);
console.log(ret2);
```



NAMED AND ANONYMOUS FUNCTION DECLARATION

```
//Benannte Funktion
function add(a, b) {
    return a + b;
}
add(3,5);

//Anonyme Funktion gespeichert in einer Variable
var add = function(a, b) {
    return a + b;
};
add(3,5);

//Anonyme Funktion direkt aufgerufen
(function(a, b) {
    return a + b;
})(3, 5);
```

NAMED AND ANONYMOUS FUNCTION DECLARATION

```
//Benannte Funktion
function doSomething(_event) {
    console.log(_event);
}
var elm = document.querySelector("#myButton");
elm.addEventListener("click", doSomething);

//Anonyme Funktion gespeichert in einer Variable
var doSomething = function(_event) {
    console.log(_event);
}
var elm = document.querySelector("#myButton");
elm.addEventListener("click", doSomething);

//Anonyme Funktion direkt als Callback
var elm = document.querySelector("#myButton");
elm.addEventListener("click", function(_event) {
    console.log(_event);
});
```

SCOPE

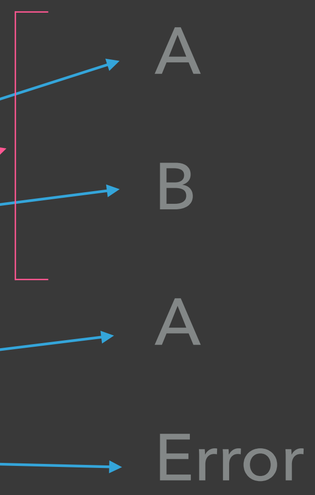
- Jede Funktion hat ihren eigenen lokalen Scope
- Der Code innerhalb einer Funktion kann die Variablen außerhalb sehen.
- Der Code außerhalb einer Funktion kann die Variablen innerhalb nicht sehen!

```
var a = "A";
var foo = function(){
    var b = "B";
    console.log(a);
    console.log(b);
};
foo();
console.log(a);
console.log(b);
```

SCOPE

- Jede Funktion hat ihren eigenen lokalen Scope
- Der Code innerhalb einer Funktion kann die Variablen außerhalb sehen.
- Der Code außerhalb einer Funktion kann die Variablen innerhalb nicht sehen!

```
var a = "A";
var foo = function(){
    var b = "B";
    console.log(a);
    console.log(b);
};
foo();
console.log(a);
console.log(b);
```



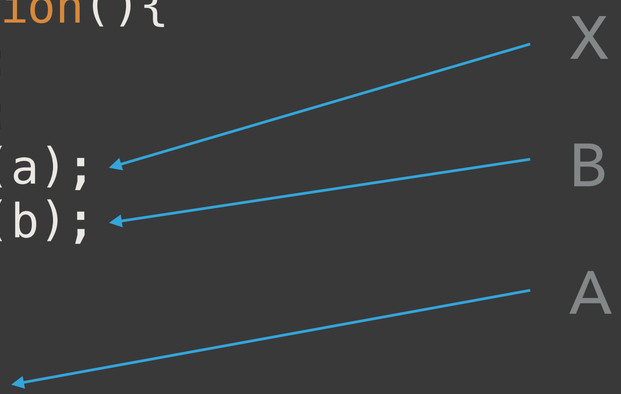
SCOPE

```
var a = "A";
var foo = function(){
    var b = "B";
    var a = "X";
    console.log(a);
    console.log(b);
};
foo();
console.log(a);
```

```
var a = "A";
var foo = function(){
    var b = "B";
    console.log(a);
    a = "X";
    console.log(a);
    console.log(b);
};
foo();
console.log(a);
console.log(b);
```

SCOPE

```
var a = "A";
var foo = function(){
    var b = "B";
    var a = "X";
    console.log(a);
    console.log(b);
};
foo();
console.log(a);
```




```
var a = "A";
var foo = function(){
    var b = "B";
    console.log(a);
    a = "X";
    console.log(a);
    console.log(b);
};
foo();
console.log(a);
console.log(b);
```

SCOPE

```
var a = "A";
var foo = function(){
    var b = "B";
    var a = "X";
    console.log(a);
    console.log(b);
};
foo();
console.log(a);
```

X

B

A

```
var a = "A";
var foo = function(){
    var b = "B";
    console.log(a);
    a = "X";
    console.log(a);
    console.log(b);
};
foo();
console.log(a);
console.log(b);
```

A

X

B

X

Error

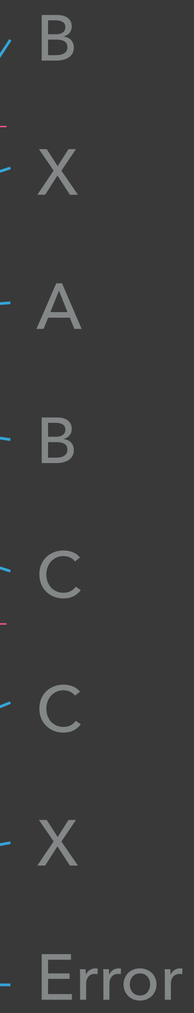
SCOPE

```
var a = "A";
var b = "B";
var foo = function(_x){
    var c = "C";
    console.log(_x);
    console.log(a);
    console.log(b);
    console.log(c);
    b = _x;
    return c;
    b = "Y";
};
console.log(b);
var tmp = foo("X");
console.log(tmp);
console.log(b);
console.log(c);
```

```
var a = "A";
var d = null;
var foo = function(_x) {
    var b = "B";
    var bar = function(_y) {
        var c = "C";
        var yay = function(_z) {
            var d = _z;
            console.log(d);
            var a = ";(";
        };
        a = ":)";
        yay(_y);
        console.log(d);
        console.log(b);
    };
    console.log(a);
    bar(_x);
};
console.log(a);
foo(":P");
console.log(a);
console.log(b);
```

SCOPE

```
var a = "A";
var b = "B";
var foo = function(_x){
    var c = "C";
    console.log(_x);
    console.log(a);
    console.log(b);
    console.log(c);
    b = _x;
    return c;
    b = "Y";
};
console.log(b);
var tmp = foo("X");
console.log(tmp);
console.log(b);
console.log(c);
```




```
var a = "A";
var d = null;
var foo = function(_x) {
    var b = "B";
    var bar = function(_y) {
        var c = "C";
        var yay = function(_z) {
            var d = _z;
            console.log(d);
            var a = ";(";
        };
        a = ":)";
        yay(_y);
        console.log(d);
        console.log(b);
    };
    console.log(a);
    bar(_x);
};
console.log(a);
foo(":P");
console.log(a);
console.log(b);
```

SCOPE

```
var a = "A";
var b = "B";
var foo = function(_x){
    var c = "C";
    console.log(_x);
    console.log(a);
    console.log(b);
    console.log(c);
    b = _x;
    return c;
    b = "Y";
};
console.log(b);
var tmp = foo("X");
console.log(tmp);
console.log(b);
console.log(c);
```

B

X

A

B

C

C

X

Error

```
var a = "A";
var d = null;
var foo = function(_x) {
    var b = "B";
    var bar = function(_y) {
        var c = "C";
        var yay = function(_z) {
            var d = _z;
            console.log(d);
            var a = ";(";
        };
        a = ":)";
        yay(_y);
        console.log(d);
        console.log(b);
    };
    console.log(a);
    bar(_x);
};
console.log(a);
foo(":P");
console.log(a);
console.log(b);
```

A

A

:P

null

B

:)

Error

DEFAULTPARAMETER

In ES6 gibt es nun auch die Möglichkeit Defaultwerte für Parameter zu vergeben

```
var foo = function(a, b){
  a = (typeof a !== 'undefined') ?  a : "DA";
  b = (typeof b !== 'undefined') ?  b : "DB";
  console.log(a + "" + b);
};

foo("a","b");
foo("a");
foo(null, undefined);
```

ab

aDB

nullDB

```
var bar = function(a = "DA", b = "DB"){
  console.log(a + "" + b);
};

bar("a","b");
bar("a");
bar();
bar(undefined,"b");
```

ab

aDB

DADB

DAb

“ARROW” FUNKTION

Arrow-Function mit 3 Parametern und 2 Statements

```
var myFunction = (a,b,c) => { a++; return a + b + c; };
```

Kurzschreibweise mit einem Parameter und 2 Statements

```
a => { a++; console.log(a); }
```

Kurzschreibweise mit einem Parameter und einem Statement das “returned” wird

```
a => a++
```

Kurzschreibweise mit einem Parameter und einem Object-Literal das “returned” wird

```
a => ({A:a})
```

Arrow-Funktionen haben kein this-binding. Es gilt also das this der umgebenden Funktion.

LET CONST – BLOCKSCOPE FÜR JS

var a;	Variable a mit Functionscope
let b;	Variable b mit Blockscope
const c;	“Konstante” c mit Blockscope

```
var i = 0;
if (true) {
  var i = 1;
}
console.log(i);

let x = 0;
if (true) {
  let x = 1;
   console.log(x);
}
console.log(x);
```

```
const j = 0;
j = 1;
console.log(y);

if (true) {
   const y = 0;
}
console.log(y);
```

```
const z = [3,1,2];
console.log(z);

z[0] = 4;
console.log(z);

z.push(8);
z = z.sort();
console.log(z);
```

LET CONST – BLOCKSCOPE FÜR JS

var a;	Variable a mit Functionscope
let b;	Variable b mit Blockscope
const c;	“Konstante” c mit Blockscope

```
var i = 0;
if (true) {
  var i = 1;
}
console.log(i);
```

1

```
let x = 0;
if (true) {
  let x = 1;
  console.log(x);
}
console.log(x);
```

1

0

```
const j = 0;
j = 1;
console.log(y);

if (true) {
  const y = 0;
}
console.log(y);
```

```
const z = [3,1,2];
console.log(z);

z[0] = 4;
console.log(z);

z.push(8);
z = z.sort();
console.log(z);
```

LET CONST – BLOCKSCOPE FÜR JS

var a; Variable a mit Functionscope
let b; Variable b mit Blockscope
const c; “Konstante” c mit Blockscope

```
var i = 0;
if (true) {
  var i = 1;
}
console.log(i);
```

1

```
let x = 0;
if (true) {
  let x = 1;
  console.log(x);
}
console.log(x);
```

1

0

```
const j = 0;
j = 1;
console.log(y);

if (true) {
  const y = 0;
}
console.log(y);
```

Type ERROR

0

Referenz ERROR

```
const z = [3,1,2];
console.log(z);

z[0] = 4;
console.log(z);

z.push(8);
z = z.sort();
console.log(z);
```

LET CONST – BLOCKSCOPE FÜR JS

var a; Variable a mit Functionscope
let b; Variable b mit Blockscope
const c; “Konstante” c mit Blockscope

```
var i = 0;
if (true) {
  var i = 1;
}
console.log(i);
```

1

```
let x = 0;
if (true) {
  let x = 1;
  console.log(x);
}
console.log(x);
```

1

0

```
const j = 0;
j = 1;
console.log(y);

if (true) {
  const y = 0;
}
console.log(y);
```

Type ERROR

0

Referenz ERROR

```
const z = [3,1,2];
console.log(z);

z[0] = 4;
console.log(z);

z.push(8);
z = z.sort();
console.log(z);
```

[3,1,2]

[4,1,2]

Type Error

[4,1,2,8]