

The logo consists of a solid yellow square centered on a dark blue background. Inside the yellow square, the letters 'JS' are written in a large, bold, dark blue sans-serif font.

JS

JAVASCRIPT

HTML UND JAVASCRIPT

JavaScript kann über das Script-Element in HTML Dateien eingebunden werden.



```
<script>  
    console.log("Hello World")  
</script>  
<script src="script.js"></script>
```

Die Einbindung einer externen Datei erfolgt im `<head>` oder vor dem `</body>`-Element.

LADEN DER DATEIEN - *BODY*

*Das Setzen des **script-Tags** am Ende des Body hat zur Folge, dass zuerst die HTML parse und dann die js ausgeführt wird. Sollte grundlegend benutzt werden, um auch ältere Browser zu unterstützen.*

LADEN DER DATEIEN - *DEFER*

*Das Setzen des Attributs **defer** im **script-tag** hat zur Folge, dass das **html** Dokument zuerst parse und dann die **js** in der Reihenfolge ausgeführt wird, wie sie im Dokument erscheint.*

LADEN DER DATEIEN - ASYNC

*Das Setzen des Attributs **async** im **script-tag** hat zur Folge, dass die js während des parse des **html** Dokuments runterlädt. Wenn diese runtergeladen ist, wird das parse des **html** Dokuments gestoppt und die js direkt ausgeführt. Scripte mit hoher Priorität und unabhängig von restlichen Dateien wie Google Analytics können mit **async** eingebunden werden.*

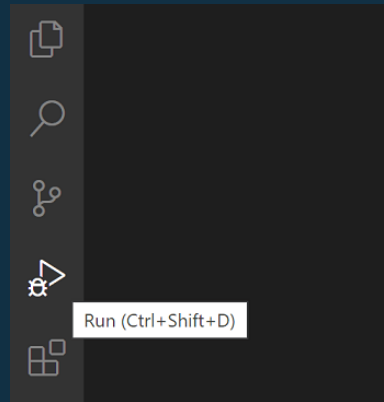
LADEN DER DATEIEN

async vs defer



KONSOLE ÜBER VSCODE

Konsolen Ausgaben lassen sich auch über die Debugging Funktion in VSCode anzeigen.



<https://code.visualstudio.com/docs/nodejs/browser-debugging>

VARIABLEN

JavaScript kennt verschiedene Arten von Variablen.

```
var x = 5;  
  
let y = 10;  
  
const name = "Ada";
```


VARIABLEN

Scopes

	var	let	const
Global Scope	✓	✓	✓
Function Scope	✓	✓	✓
Block Scope	✗	✓	✓
Can be reassigned?	✓	✓	✗
Can be redeclared?	✓	✗	✗

VARIABLEN SCOPE

```
let firstname = "Ada"; // Global Scope

function sayName(name) { // Function Scope
  console.log(name);
}

if (firstname === "Ada") {
  // Block Scope {}
  console.log("Hello %s", firstname)
}
```

VARIABLEN SCOPE

```
var x = "outside";  
function foo() {  
  var x = "inside";  
  console.log(x);  
}  
foo();  
console.log(x);
```

Was wird ausgegeben?

VARIABLEN SCOPE

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

Was wird ausgegeben?

DATENTYPEN

Boolean

Zwei Werte: *true* und *false*

Null

Einen Wert: *null*

Undefined

Eine Variable der noch kein Wert zugewiesen wurde.

Number

Zahlen; Ganzzahlig und Gleitkommazahl

String

Zeichenketten; stehen in Anführungszeichen

ARRAYS UND OBJEKTE

Zur *Speicherung* mehrere Daten in einer Variable kennt JavaScript Arrays und Objekte.

```
// Array
var color = ["blue", "red", "yellow"] // Zugriff über Index (0...)

// Object
var person = {lastname: "Lovelace", firstname: "Ada",
  name: "value"}
```

Auch Arrays sind Objekte.

OBJECT REFERENCE

Variablen für ein Objekt speichern nicht den Inhalt sondern lediglich eine Referenz (Zeiger) auf das Objekt.
Dadurch lässt sich ein Objekt durch eine Zuweisung **nicht** kopieren.

```
const array1 = [1, 2];  
const array2 = array1;  
// Änderungen an einem der Arrays wirken sich auf beide Arrays aus  
array2.pop(); // Löschen des letzten Elements  
console.log(array1);  
// Ausgabe: [1]
```

KOPIEREN EINES OBJECTS / ARRAYS

zwei Methoden

```
// Array Methode  
const clone = original.slice();
```

```
// Object  
const clone = Object.assign({}, original);
```


AUFGABE (1.1)

Schreibe ein Programm, dass den Benutzer nach einem Alter in Jahren fragt. Anschließend soll das Programm das Alter jeweils in Monaten, Tagen und Stunden ausgeben.

```
prompt("Bitte gebe ein Alter in Jahren ein!");
```



BEDINGUNGEN

Über Bedingungen lassen sich Fallentscheidungen treffen. In JavaScript werden hierfür die Schlüsselwörter **if** und **else**, sowie eine Kombination verwendet.

```
if (alter >= 18) {  
    console.log("Du darfst Auto fahren!");  
} else if (alter >= 17) {  
    console.log("Du darfst mit Begleitung fahren.");  
} else {  
    console.log("Du darfst noch kein Auto fahren");  
}
```

VERGLEICHSOPERATOREN

Operator	Beschreibung
==	Equal – true, wenn Operanden gleich.
===	Strict Equal - true, wenn Operanden und Typ gleich.
!=	Not equal – true, wenn Operanden ungleich sind
!==	Strict not equal – true, wenn Operanden und Typ ungleich.
>	Greater than – true, wenn der linke Operand größer dem rechten Operanden ist.
>=	Greater than or equal – true, wenn der linke Operand größer als, oder gleich dem rechten Operanden ist.
<	Less than – true, wenn der linke Operand kleiner dem rechten Operanden ist.
<=	Less than or equal – true, wenn der linke Operand kleiner als, oder gleich dem rechten Operanden ist.

BEDINGUNGEN

Bedingung mit dem logischen && Operator

```
// Ergebnis vor dem && ist wahr:  
true && console.log("Hier ist eine Ausgabe")  
// Ergebnis vor dem && ist falsch:  
false && console.log("Hier kommt keine Ausgabe")
```

Der &&-Operator gibt den ersten falschen (False) Wert aus.
Abbildung von *else* nicht möglich!

BEDINGUNGEN

Bedingung mit Conditional Operator

```
// Condition ? if true : if false  
true ? console.log("Ausgabe") : console.log("Keine Ausgabe")
```

AUFGABE (1.2)

Wenn die Eingabe des Alters negativ ist soll eine Fehlermeldung ausgegeben werden.



Löse die Aufgabe mit *if* sowie dem Conditional Operator.

SCHLEIFEN

JavaScript kennt verschiedene Schleifentypen.

- for Schleife
- while Schleife
- do...while Schleife

FOR-SCHLEIFE

```
for (var i = 0; i < 9; i++) {  
  console.log(i);  
  // more statements  
}  
// Inkrement Operator (++), addiert eine eins und  
// gibt den Wert zurück.
```

Die For-Schleife läuft so lange wie die Bedingung *true* ist.

ITERABLE OBJECTS

Mit *for* können iterable Objekte durchlaufen werden.

```
let arr = [3, 5, 7]; // Array

for (let i in arr) {
  console.log(i); // logs "0", "1", "2"
}

for (let i of arr) {
  console.log(i); // logs "3", "5", "7"
}
```

Iterator i und Iterable arr

WHILE SCHLEIFE

```
var n = 0;  
  
while (n < 3) {  
    n++; //  
}
```

Die While-Schleife läuft so lange wie die Bedingung war (*true*) ist.

DO...WHILE SCHLEIFE

```
var i = 0;  
do {  
  i += 1;  
  console.log(i);  
} while (i < 5);
```

Die do...while-Schleife läuft so lange wie die Bedingung *true* ist, jedoch mindestens einmal (Fußgesteuerte Schleife).

AUFGABE (2.1)

Schreibe ein Programm, dass aus einem Zahlen-Array die größte Zahl ermittelt und ausgibt.
Nutze hierfür die for...of Schleife.

```
const zahlen = [12, 34, 29, 120, 55];  
...  
// Ausgabe: "Die größte Zahl ist 120"
```

1 2
3 4

Aufgabenstellung

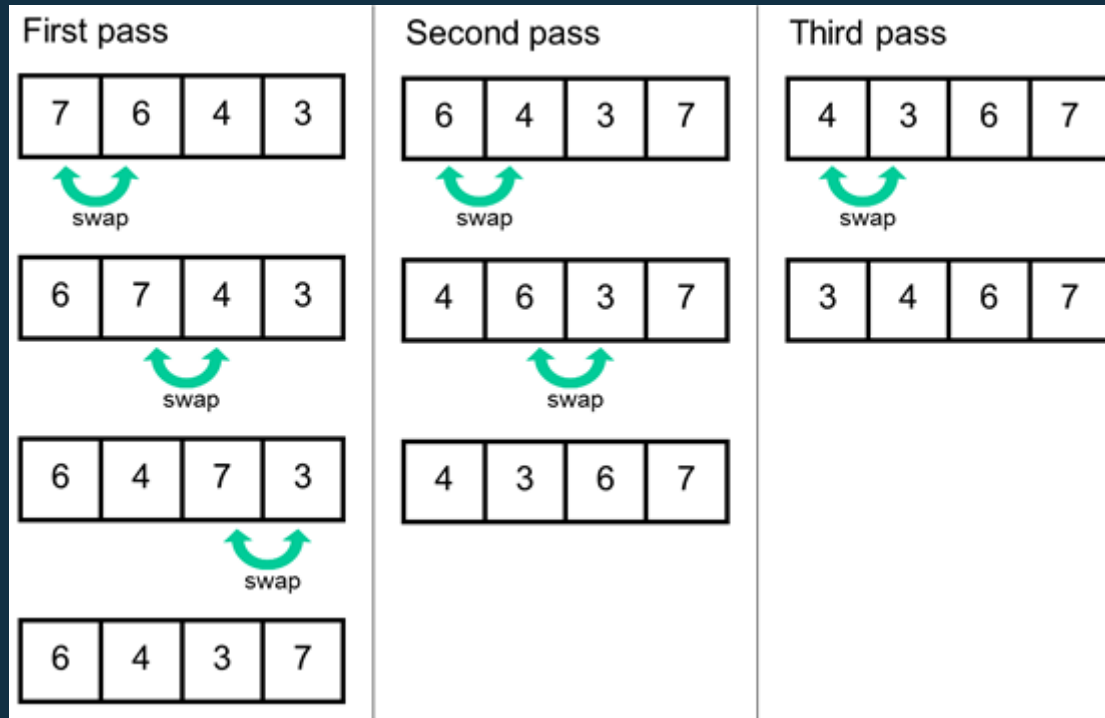
SORTIER ALGORITHMEN

Mit Hilfe der Grundstrukturen einer Programmiersprache lassen sich verschiedene Sortier-Algorithmen abbilden.

- Bubble Sort
- Selection Sort
- Insertion Sort
- ...

<https://de.wikipedia.org/wiki/Sortierverfahren>

BUBBLE SORT



Zahlen werden durchlaufen und mit dem rechten Nachbarn verglichen. Bei Regelverletzung werden die Zahlen getauscht.

BUBBLE SORT

Bubble Sort mit JavaScript

```
let arr = [12 , 5, 99, 34];

for (let n = arr.length; n>1; n--) {
  for(let i = 0; i < n-1; i++) {
    if(arr[i] > arr[i+1]) {
      let arr_old = arr[i];
      arr[i] = arr[i+1];
      arr[i+1] = arr_old;
    }
  }
}
```



FUNKTIONEN

Eine Funktion beschreibt eine Reihe von Anweisungen, um eine Aufgabe auszuführen.

```
function square(number) {  
    return number * number;  
}  
  
const quadrat = square(2);
```

Häufig haben Funktionen Übergabeparameter () sowie einen oder mehrere Rückgabewerte (*return*).

ANONYMOUS FUNCTIONS

In JavaScript lassen sich Funktionen ohne Namen erzeugen.

```
let show = function () {  
  console.log('Anonymous function');  
};  
show();
```

Zum späteren Aufruf können anonyme Funktionen in eine Variable gespeichert werden. In einigen Fällen ist eine Speicherung mit Namen jedoch nicht erforderlich.

AUFGABE (2.2)

Implementiere die Rückgabe des höchsten Wertes in eine Funktion.

Bei Funktionsaufruf soll ein Array übergeben werden und der höchste Wert zurückgegeben werden.

1	2
3	4

Aufgabenstellung

INTERVAL UND TIMEOUT

Über *setInterval* und *setTimeout* lassen sich Funktionsaufrufe zeitlich steuern.

```
// Alle drei Sekunden wird Hello ausgegeben.  
setInterval(function(){ console.log("Hello"); }, 3000);  
// Nach drei Sekunden wird Hello ausgegeben.  
setTimeout(function(){ console.log("Hello"); }, 3000);
```

Die Angabe der Zeit erfolgt in Millisekunden.

ARRAY METHODEN


Arrays haben viele vordefinierte Methoden um die Daten zu verarbeiten.







```
// Entfernen des letzten Elements und Rückgabe  
const letztesElement = zahlen.pop();  
// Sortieren  
zahlen.sort();
```

Überblick der Methoden

Ähnliche Methoden gibt es auch für Strings (siehe Übungen).

ARRAY METHODEN






.map(=>) => 

.filter() => 






.every() => false

.some() => true

.fill(1, ) => 

.findIndex(el => el===) => 3

.find() => 

.reduce((acc, cur)=>acc+cur)=> 

AUFGABE (2.3)

Erweitere die Zahlen-Array Aufgabe um die Funktion `sort()`.
Gebe die Zahlen in umgekehrter Reihenfolge (Groß nach Klein) aus.

1	2
3	4

Aufgabenstellung

ARRAY METHODE MAP()

Wendet eine Funktion auf jedes Array Element an und gibt ein neues Array zurück.

```
const numbers = [4, 9, 16, 25];  
const newArr = numbers.map(verdoppeln)  
  
function verdoppeln(zahl) {  
    return zahl * 2;  
}
```

ARRAY METHODE FILTER()

Prüft jedes Element gegen eine Funktion deren Rückgabe *true* oder *false* ist.

```
const ages = [32, 33, 16, 40];  
ages.filter(checkAdult)    // Returns [32, 33, 40]  
  
function checkAdult(age) {  
  return age >= 18;  
}
```


ARRAY METHODE CONCAT()

Zusammenführen von zwei Arrays (merge).

```
const hege = ["Cecilie", "Lone"];  
const stale = ["Emil", "Tobias", "Linus"];  
const children = hege.concat(stale);  
// Ausgabe: ['Cecilie', 'Lone', 'Emil', 'Tobias', 'Linus']
```

AUFGABE



(2.4)

Schreibe ein Programm, dass die Zahlen in einem Array filtert, so dass nur gerade Zahlen ausgegeben werden.

1	2
3	4

Aufgabenstellung

TRY...CATCH

Block von Anweisungen bei dem im Fehlerfall eine gewünschte Reaktion ausgeführt wird.

```
try {  
    // Block of code to try  
}  
catch(err) {  
    // Block of code to handle errors  
}
```

Der Fehlerfall kann sowohl unvorhergesehen, als auch gewollt sein.