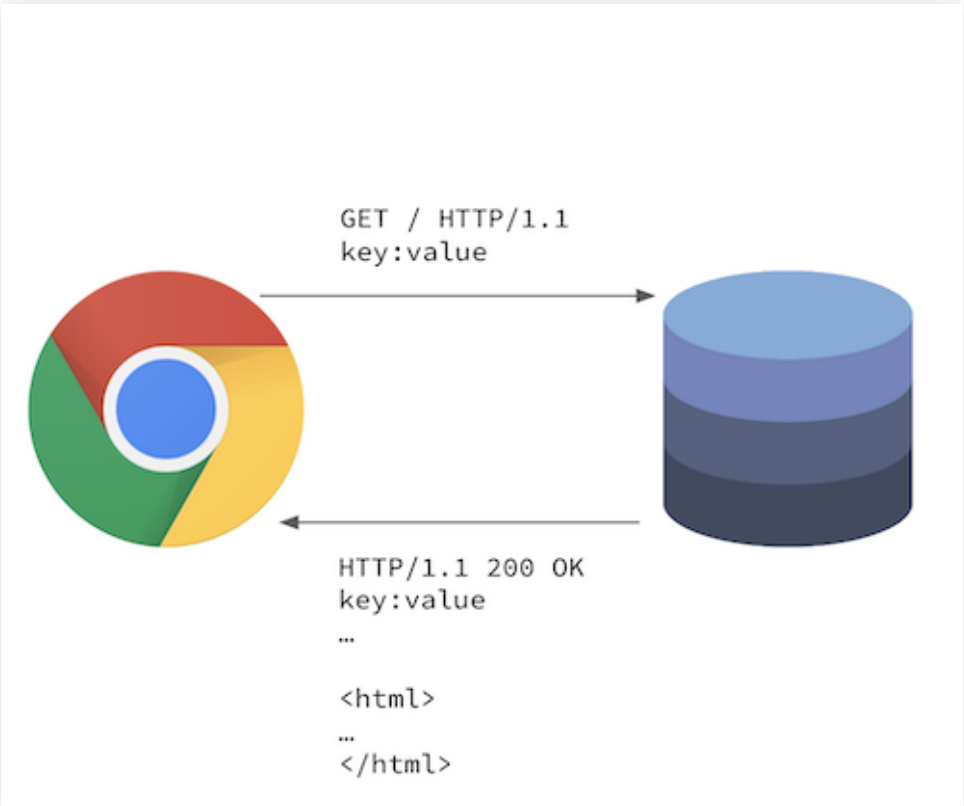
JAVASCRIPT

JAVASCRIPT

ist eine interpretierte Sprache, die von Browsern intepretiert wird.

1. Chrome fragt Adresse an
2. Server liefert HTML Dokument mit Javascript zurück
3. Chrome intepretiert HTML, CSS, Javascript



```
<html>
  <head>
    <script type=”text/javascript”>
      function onHello(event) {
        event.target.innerHTML = "World";
      }

      function outHello(event) {
        event.target.innerHTML = "Hello";
      }

      var element = document.getElementById("hello");
      element.addEventListener("mouseover", onHello);
      element.addEventListener("mouseleave", outHello);
    </script>
  </head>
  <body>
    <p>Hallo</p>
  </body>
```

VARIABLEN, BEDINGUNGEN UND SCHLEIFEN

VARIABLEN IN JAVASCRIPT

Variablen werden mit `var` eingeleitet.

```
var a = 2;
var b = 'hello world';
var c = 3.5;
var d = true;
```

BEDINGUNGEN IN JAVASCRIPT

Javascript hat zwei Gleichheitsoperatoren `==` und `===` Link.

- `==` prüft auf "lose Gleichheit"
- `===` prüft auf "strikte Gleichheit"

```
var a = 2;
var b = '2';
if (a == b) {
  console.log('works!');
} else {
  console.log('jumps not in here');
}

if (a === b) {
  console.log('doesnt work!');
} else {
  console.log('jumps here');
}
```

Ansonsten gelten die allgemeinen Vergleichsoperatoren:

`>`, `<`, `<=`, `>=`

SCHLEIFEN IN JAVASCRIPT

Link

FOR-SCHLEIFE

```
for (var variable = anfangswert; abbruchbedingung; inkrement) {
  ...
}
```

Beispiel

```
var sum = 0;

for (var i = 0; i <= 100; i++) {
  sum = sum + 1;
}

console.log(sum);
```

DO-WHILE-SCHLEIFE

```
var confirmed = false;

do {
  confirmed = confirm(
    'Wollen Sie diese Seite wirklich sehen?'
  );
} while(!confirmed)
```

WHILE-SCHLEIFE

```
var confirmed = false;

while(!confirmed) {
  confirmed = confirm(
    'Wollen Sie diese Seite wirklich sehen?'
  );
}
```

Eine `Do-While-Schleife` lässt sich immer in eine `While-Schleife` übertragen.

Wie?

```
var confirmed = false;

while(!confirmed) {
  confirmed = confirm(
    'Wollen Sie diese Seite wirklich sehen?'
  );
}
```

JAVASCRIPT ARRAYS

MOTIVATION

Werte gemeinsam in eine Variable speichern

statt

```
var a1 = 2;
var a2 = 4;
var a3 = 6;
```

so

```
var a = [2, 4, 6];
```

AUF WERTE ZUGREIFEN

Der Index muss eine integer Zahl sein.

```
var a = [2, 4, 6];
console.log(a[0]); // 2
console.log(a[1]); // 4
console.log(a[2]); // 6
```

ITERIEREN

```
var a = [2, 4, 6];
for (var i = 0; i < a.length(); i++) {
  console.log(a[i]);
}
```

WERT HINZUFÜGEN

Link

```
var a = [2, 4, 6];
a.push(8);
console.log(a); // [2, 4, 6, 8]
```

WERT LÖSCHEN

Link

```
var a = [2, 4, 6];
a.splice(1, 1);
console.log(a); // [2, 6]
```

JAVASCRIPT OBJECTS

JAVASCRIPT OBJECTS (OBJEKTE)

sind das wichtigste Konzept in Javascript.

MOTIVATION

statt

```
var firstName = 'Michael';
var lastName = 'Bykovski';
var age = 25;
```

so

```
var human = {
  firstName: 'Michael',
  lastName: 'Bykovski',
  age: 25,
}
```

AUF WERTE ZUGREIFEN

1. Mit Punkt und dem Indexnamen
2. Wie bei einem Array über den Indexnamen

```
var human = {
  firstName: 'Michael',
  lastName: 'Bykovski',
  age: 25,
}

console.log(human.firstName); // Michael
console.log(human['lastName']); // Bykovski
```

WERTE HINZUFÜGEN

Werte hinzufügen, wie beim Zugriff.

```
var human = {
  firstName: 'Michael',
  lastName: 'Bykovski',
  age: 25,
}

human['city'] = 'Wiesbaden';
human.city = 'Wiesbaden';
```

JAVSCRIPT FUNKTIONEN

Funktionen kapseln `Code-Anweisungen innerhalb eines Namensraum` ab. Alle Variablen, die innerhalb der Funktion definiert wurden, werden beim `Abschluss der Funktion gelöscht`.

SYNTAX

```
function name(parameter1, parameter2, paramenterN) {
  // code to be executed
}

function name(parameter1, parameter2, paramenterN) {
  // code to be executed
  return x;
}
```

BEISPIEL

```
var human = {
  firstName: 'Michael',
  lastName: 'Bykovski',
  age: 25,
}

function fullName(humanObject) {
  var fullNameVar = humanObject.firstName + ' ' + humanObject.lastName;
  return fullNameVar;
}

console.log(fullName(human)); // Michael Bykovski
console.log(fullNameVar); // error: fullNameVar is not defined
```

WEITERES BEISPIEL

```
var human = {
  firstName: 'Michael',
  lastName: 'Bykovski',
  age: 25,
}

function setFullName(humanObject) {
  humanObject.fullName = humanObject.firstName + ' ' + humanObject.lastName;
}

console.log(setFullName(human)); // undefined
console.log(human.fullName); // Michael Bykovski
```

VEREINIGUNG UND `this` KEYWORD

```
var human = {
  firstName: 'Michael',
  lastName: 'Bykovski',
  age: 25,
  fullName: function() {
    return this.firstName + ' ' + this.lastName;
  }
}

console.log(human.fullName()); // Michael Bykovski
```

DAS `window` OBJECT

Link

Das `window` object enthält Objekte, Eigenschaften und Funktionen, die der Browser und das aktuelle Fenster (in dem das Javascript ausgeführt wird) bereitstellen.

Auf das `window` object kann jederzeit und überall zugegriffen werden, da es so elementar ist. Es ist "global" verfügbar.

```
console.log(window.innerWidth); // 700
var result = prompt('Bitte geben Sie eine Zahl ein: ');
```

DAS `document` OBJECT

enthält Funktionen, mit denen sich auf DOM Elemente zugreifen, manipulieren, hinzufügen und löschen lassen.

Link

DOM = Document Object Model ist eine Repräsentierung aller HTML Elemente im HTML Dokument als Javascript Objekte.

Beispiel

```
<p id="hello" class="hello">Der Text</p>
<p id="keinHello" class="hello">Ein weiterer Text</p>
```

```
var helloElement = document.getElementById('hello');
var helloElements = document.getElementsByClassName('hello');

var helloElement = document.querySelector('.hello');
var helloElements = document.querySelectorAll('.hello');
```

DAS OBJEKT ELEMENT

spiegelt in Javascript ein HTML-Element wieder.

```
<p id="hello" class="hello">Der Text</p>
```

```
var helloElement = document.getElementById('hello');
console.log(helloElement.classList); // DOMTokenList ["hello", value: "hello"]
```

DOMTokenList

EREIGNISSE

Damit Ereignisse, wie beispielsweise ein `Klick` von Javascript verwendet werden können, müssen diese `"registriert"` werden. Bei der Registrierung wird eine Funktion `mitgegeben`, die ausgeführt wird, wenn das Ereignis `eintrifft`.

Liste aller Events

Event Object

```
<p id="clickme">Klick auf mich</p>
```

```
var element = document.getElementById('clickme');
var onClick = function(event) {
  alert('Geklickt!');
};
element.addEventListener('click', onClick)
```

```
document.getElementById('clickme').addEventListener(
  'click',
  function(event) {
    alert('Geklickt!');
  },
);
```

Resultat