Contents

JS

Data Types

```
var x = 1; // works inside function scope
           // doesn't matter where declared it will exist inside the script
let y = 2; // works only inside defined block (e.g. function)
const z = 3; // works only inside defined block & not changeable
document.writeln(x); // write to document body
```

Operators

- arithmetic: `+`, `-`, `*`, `/`, `%`, `++`, `--`, . . .
- assign: `=`, `+=`, `-=`, `*=`, `/=`, . . .
- compare: `==`, `<`, `<=`, `>=`, `>`, `!=`, . . .
- compare with datatype check: `===`
- bitwise: `&`, `|`, `^`, `<<`, `>>`, . . .

Control Flow

```
if {}
else {}

for (let i = 0; i < 5; i++) {
    continue; // <-- skips the rest of the current iteration
}

for (let key in obj) {}

for (let n of numbers) {}

numbers.forEach((number) => { console.log(number); }) // read only

while (condition) {}

do {} while (condition);

switch (expression) {
    case val1:
        // Code
        break;
    case val2:
        // Code
        break;
    default:
        // Code
}
```

Arrays

- are dynamic

```
let numbers = [1, 2, 3, 4]; // index starts with 0, type: object
let persons = ["Petra", "Maier", 23, true]; // datatypes don't matter
let length = numbers.length;
person.push("another person"); // add el
person.sort(); // sort elements
let [first, second, third, fourth] = persons; // deconstruct

let arr2 = [1,2,3,4,5];
arr2.splice(1, 3); // splice on index 1 and remove 3 elements
// [1, 5]
```

Strings

```
let str = "Hello World";
str = "Hello " + "World"; // concat
length = str.length;
let sub = str.substring(0, 2); // without index 2
let new_str = str.replace("Hello", "Hi"); // returns new str
str.replaceAll("l", "M");
let char = str.charAt(2);
char = str[2]; // strings start with 0
let arr = str.split(" "); // [Hello, World]`
// Multi Line
let i = 1;
`Var: ${i}`
```

Objects

```
let person = {}; // empty obj

person = {
    firstName: "James",
    age: function(age) {
        this.age = age;
    },
    animal(animal) { // new property
        this.animal = animal;
    },
    toString: function() { // override toString()
        return this.firstName + " " + this.lastName;
    }
}
```

Operations

```
person.lastName = "Bug" // create new property
person.move = function(location) { // create new function + property
    this.location = location; // location is a new property!
};
person.move("Germany");
person.age(21);
person.animal("cat");
```

Get Keys

```
const obj = { 0: "a", 1: "b", 2: "c" };
let objkeys = Object.keys(obj); // ['0', '1', '2'] type Array<Object>
```

```
let objvalues = Object.values(obj); // ['0', '1', '2'] type Array<Object>
let letter = obj["0"]; // use string to access property
```

Functions

- Functions are objects

```
function f1(var1, var2 = 2) { } // no return types for functions or parameters
// with default value
let f2 = function(var1, var2) { } // anonymous function
let f3 = (var1, f4) => { // arrow function
    f4(var1);
}
f3(4, function(variable) { console.log(variable) });
```

Filtering

```
let filter_array = [];
numbers.forEach((number) => {
    if(number > 0) { filter_array.push(number); }
})

filter_array = numbers.filter((number) => number > 5);
filter_array = numbers.filter((number) => {
    if(number > 5) { return number }
})
```

Custom Sort Criteria

```
let persons2 = [
    { name: "Petra",
      age: 21 },
    { name: "Maier",
      age: 20 }
]
persons2.sort((p1, p2) => {
    return p1.name.localeCompare(p2.name); // sort by name, locale based sorting
    // return p1.age - p2.age; // sort by age
})
```

Map

```
const map1 = new Map();
map1.set('a', 1);
map1.set('b', 2);
map1.get("b"); // 2
```

```
let keys = map1.keys(); // {a, b}
let values = map1.values(); // {1, 2}
```

Classes

```
class Person {
    nonPrivateField = 1;
    #privateField;

    constructor(name, age) {
        this.nonPrivateField = 2;
        this.newField = 3; // new field (doesn't exist outside ctor)
        this.#privateField = 1;

        // _ is an old convention for private fields
        // control private fields with getter & setter
        this._name = name; // new + private field
    }

    // getter & setter are mandatory when working with private attributes
    get privateField() { return this.#privateField }
    set privateField(value) { this.#privateField = value }

    set name(name) { this._name = name } // access with Person.name = "Name"
    get name() { return this._name } // access with Person.name

    toString() { return "Person class" }
}

let p1 = new Person("Tom", 21);
p1.privateField; // 1, getter
p1.privateField = 2; // setter

p1.nonPrivateField; // 2
p1.nonPrivateField = 3; // 3

class Worker extends Person {
    constructor(name, age, nr) {
        super(name, age);
        this._nr = nr;
    }
    toString() { return super.toString() + "Worker class" }
}
```

Error Handling

```
if(smt == 0) { throw "new error" }

try {
    const z = 0;
    z = 1; // throws error
} catch (e) {
    console.log(e);
}
```

I/O

```
console.log(x, y, z);
console.error(x, y, z);
console.log(`print var ${x}`);
console.log("print var" + x + "with concat");
console.log("print var" + (x == 0 ? y : z) + "with condition");
```

JSON

- Object description, used for web communication
- allowed attribute types: string, number, bool, null, object, array
- functions not allowed

```
let person = {
    "name": "Tom Hengst", // attributes in quotes
    "age": 25, // Access person.age
}

// convert JSON object to string -> "{ "name" = "Tom Hengst", .....}"
let personAsString = JSON.stringify(person);
// convert string to JSON object -> equals person again
let personAsJSON = JSON.parse(personAsString);
```

DOM (document object model)

Find Elements

```
let elements = document.getElementsByClassName("class"); // returns array by css class name
let element = document.getElementsByClassName("class")[0]; // return first found element by
let elements2 = document.getElementsByName("name"); // returns array by name attribute
let elements3 = document.getElementsByTagName("div"); // returns array by tag
let element1 = document.getElementById("id"); // id attribute
let element3 = document.querySelector("div > #id");

const element4 = document.querySelector("div.user-panel.main input[name='login']");
```

```
const elements4 = document.querySelectorAll("div input[name='login']");

const form = document.forms.formName; // name of form attribute
```

Get Related Nodes

```
let parent = node.parentNode;
let children = node.children; // array & skip text nodes
let children2 = node.childNodes; // array
let next = node.nextElementSibling; // skip text nodes (document.createTextNode(text);)
let next2 = node.nextSibling;
let prev = node.previousSibling;
// & node.firstChild, node.lastChild
// document.firstChild returns <html>
```

Manipulation

```
let el = document.createElement("div");
el.classList.add("class"); // add a class to class attribute automatic concatenation
el.classList.remove("class");
el.classList.contains("class");
el.classList; // return classnames as array
el.className = "class"; // completely override class attribute value
el.className // return classname
el.setAttribute("name", "value"); // add name="value"
el.getAttribute("name");

el.style.backgroundColor = "red"; // change css attributes
el.style.backgroundColor; // return the color value
el.style = "background-color: red"

el.innerHTML = "<div> some text </div>";
el.appendChild(node); // add child at the end inside parent
el.removeChild(node);
el.insertBefore(node, reference); // insert child before a specific node
```

Event Listeners

```
element.addEventListener("click", () => {
    // other event listeners: "submit", "mouse+over/out/up/down"
    // "keyup" keyrelease, "keydown" keypress, "input" for chars & numbers only
    // "onfocus", "resize", "mousemove", "onload"

    // html based version works with <div onclick="mouseclick(this)"></div>
    // function mouseclick(element) { ... }
});
```

Fetch API

```
async function getFriends() {
    let uri = chatServer + chatCollectionId + "/friend";
    let response = await fetch(uri, {
        method: "GET",
        body: JSON.stringify(data),
        headers: {
            'Authorization': "Bearer " + chatToken
        }
    });

    if(response.ok) {
        let result = await response.json();
        console.log(result);
        return result;
    }
    else {
        console.error('error ' + response.status);
        return  null;
    }
}

// Alternative (Promise based):
function getFriends() {
    fetch(uri, {
        method: "GET",
        body: JSON.stringify(data),
        headers: {
            'Authorization': "Bearer " + chatToken
        }
    })
    .then(response => { // server responds
        if(response.ok) {
            return response.json();
        }
        else {
            console.error('error ' + response.status);
            return null;
        }
    })
    .then(data => {
        console.log(data);
    })
    .catch(error => { // server error
        console.error('error ' + error);
```

```
    });
}
```

AJAX

```
function getFriends() {
    let uri = chatServer + chatCollectionId + "/friend";
    let xmlhttp = new XMLHttpRequest();
    xmlhttp.onreadystatechange = function () {
        if (this.readyState == 4 && this.status == 200) {
            let data = JSON.parse(xmlhttp.responseText);
            document.getElementById("txtHint").innerHTML = data;
    }};
    xmlhttp.open("GET", uri, true); // true = asynchronous
    xmlhttp.setRequestHeader('Authorization', 'Bearer ' + chatToken);
    // xmlhttp.setRequestHeader('Content-type', 'application/json'); for post, put etc.
    xmlhttp.send();
    // xmlhttp.send(JSON.stringify(data)); for post, put etc.
}
```