

JavaScript()

# JavaScript definition

- Language of the web
- Dynamically typed
- Aside from websites it can be used for **servers**, **mobile applications** or even **desktop software**.

# Adding JavaScript to a project

```
<body>
  <div>
    <p>JavaScript Workshop</p>
  </div>
  <script src="src/index.js"></script>
</body>
```

```
<body>
  <div>
    <p>JavaScript Workshop</p>
  </div>

  <script>
    // JavaScript goes here
  </script>
</body>
```

# Data declaration and reassignment

- `const` – short for constant – it cannot be reassigned;
- `let` – short for `let x be equal y` – it can be reassigned;

```
const name = "Stefan";
```

```
let age = 20;
```

```
const name = "Błażej";
```

```
name = "Błażej";
```

```
let age = 20;
```

```
age = 21;
```

# Scope

- Both `const` and `let` are `block-scoped`
- Block-scope is defined by curly braces `{}`
- Scope works up but not down

```
const name = "Błażej";

function localScope() {
  const surname = "Nowak";
  console.log(name, surname); // Błażej Nowak
}

console.log(surname); // surname is not defined
```

# Data types

- String
- Number
- Function
- Date
- Boolean – true / false
- Null – for no value
- Undefined – for unassigned values
- Object – complex data structures
- Array – list of values

```
const string = "";  
const number = 0;  
const object = {};  
const array = [];  
const boolean = true;  
const noValue = null;  
const date = Date();  
let unassignedValue; // undefined  
  
function myFunction() {}
```

# Array

- List of values
- Represented by 0-based index
- Can contain any other type of data including other arrays
- Each value in the array can be accessed via square bracket notation []

```
const array = ["string", 10, true, ["string", 20], {}, function() {}];  
  
const index = 2  
  
console.log(array[1], array[index]); // 10, true
```

# Object

- Dataset containing contextual information
- Represented as a set of key-value pairs
- Can contain any other type of data including other objects
- You can access each value using a **key** and **dot** or **square bracket** notation

```
const person = {  
  name: "Janusz",  
  age: 20,  
  student: true,  
  siblings: [{ name: "Janina", age: 11, relationship: "sister" }]  
};  
  
const dynamicValue = "student";  
  
console.log(person.name, person["age"], person[dynamicValue]);
```



# Function

- Enables to run a particular block of code on demand
- Takes in arguments, does calculations and returns new value – or not
- Function declaration vs. Function expression
- Arrow function

```
function functionDeclaration(arg1, arg2) {}  
  
const functionExpression = function(arg) {};  
  
const arrowFunction = arg => {};  
  
const implicitReturnArrowFunction = (arg1, arg2) => arg1 + arg2;
```

# Primitive vs. Complex types

- String, number, boolean, null & undefined are primitive
- Array, function, object and date are complex
- Primitive types are passed by value
- Complex types are passed by reference

```
const complex = {  
  age: 0  
};  
let primitive = 0;  
  
function myFunction(prim, comp) {  
  prim = 10;  
  comp.age = 10;  
  console.log("function scope", { prim, comp });  
}  
  
myFunction(primitive, complex);  
console.log("global scope", { primitive, complex });
```

function scope ▼ {prim: 10, comp: {...}} ⓘ

▼ comp:

age: 10

▶ \_\_proto\_\_: Object

prim: 10

▶ \_\_proto\_\_: Object

global scope ▼ {primitive: 0, complex: {...}} ⓘ

▼ complex:

age: 10

▶ \_\_proto\_\_: Object

primitive: 0

▶ \_\_proto\_\_: Object

# Arythmetic operators

```
const add = 2 + 2;  
const subtract = 2 - 2;  
const multiply = 2 * 2;  
const divide = 2 / 2;  
const modulo = 3 % 2;  
const power = Math.pow(2, 3);  
const squareRoot = Math.sqrt(4);
```

# Logic operators

```
console.log(5 == "5"); // true
console.log(5 === "5"); // false
console.log(5 !== "5"); // true
console.log(5 != "5"); // false
console.log(5 > "5"); // false
console.log(5 >= "5"); // true
console.log(5 < "5"); // false
console.log(5 <= "5"); // true
```

# Conditional statements

```
const num1 = 5;
const num2 = -2;

if(num1 === 5 && num2 === 5){
  console.log("Both values are equal to 5")
} else if(num1 !== 5 || num2 !== 5) {
  console.log("At least one value is equal to 5")
} else {
  console.log("none of the values are equal to 5")
}
```

# Array methods

- Array.prototype.filter
- Array.prototype.sort
- Array.prototype.map
- Array.prototype.reduce

```
const array = [1, 2, 5, 6, 10, 40, 111, 44, 1, 3, 4, 51, 44];

const onlyOddNumbers = array.filter(a => a % 2); // [1, 5, 111, 1, 3, 51];

onlyOddNumbers.sort((current, next) => (current > next ? 1 : -1)); // [1, 1, 3, 5, 51, 111];

const squareAllNumbers = onlyOddNumbers.map(current => current * current); // [1, 1, 9, 25, 2601, 12321]

const reduceToSingleValue = squareAllNumbers.reduce(
  (accumulator, currentValue) => {
    return accumulator + currentValue;
  },
  0
); // 14958
```

## Array exercises

<https://github.com/mackankowski/frontend-bootcamp/blob/master/trainings/js/playground/array-exercises.html>



# Ternary operator

```
const height = 160;
```

```
const canRideRollercoaster = height >= 160 ? "YES!" : "NO!";
```

# Switch statement

```
const number = 1;

switch (number) {
  case 1:
    console.log("one");
    break;
  case 2:
    console.log("two");
    break;
  case 5 + 5:
    console.log("expressions work too");
    break;
  case "1":
    console.log("it's a string!");
    break;
  default:
    console.log(number);
    break;
}
```

# Loops - while

```
let i = 0;  
const arr = [];  
while (i < 5) {  
  arr.push(i);  
  i++;  
}
```

# Loops - for

```
const arr = [];  
for (let i = 0; i < 5; i += 1) {  
  arr.push(i);  
}
```

# Accessing DOM elements

```
const div = document.querySelector("div");
const ul = document.querySelector("#list");
const li = document.querySelectorAll(".list-item");

console.log(div, ul, li);
```

```
▶ <div>...</div> ▶ <ul id="list">...</ul> ▶ NodeList(4) [li.list-item, li.list-item, li.list-item, li.list-item]
```

```
<div>
  <ul id="list">
    <li class="list-item"></li>
    <li class="list-item"></li>
    <li class="list-item"></li>
    <li class="list-item"></li>
  </ul>
</div>
```

# Accessing DOM elements

```
const div = document.querySelector("div");  
  
const children = div.childNodes;  
const innerText = div.innerText;  
const innerHTML = div.innerHTML;  
const classList = div.classList;
```

# Changing DOM elements

```
const div = document.querySelector("div");  
  
div.classList.add("newClass", "awesome");  
div.classList.remove("newClass");  
div.innerText = "hello";  
div.style.backgroundColor = "purple";
```

# Adding new DOM elements

```
const div = document.querySelector("div");  
  
const header = document.createElement("header");  
const h1 = document.createElement("h1");  
h1.innerText = "Hello 3LO";  
header.append(h1);  
div.prepend(header);
```

# Hello 3LO

JavaScript Workshop



# DOM Events

```
const div = document.querySelector("div");
const eventHandler = eventObject => {
  console.log(eventObject);
};

div.addEventListener("click", eventHandler);
div.addEventListener("mouseover", eventHandler);
div.addEventListener("mouseenter", eventHandler);
div.addEventListener("mouseleave", eventHandler);
document.addEventListener("keyup", eventHandler);
document.addEventListener("keydown", eventHandler);
// ...and many more
```

# SetTimeout / ClearTimeout

```
const timeoutId = setTimeout(() => {  
  console.log("Hello!");  
}, 2000);  
  
clearTimeout(timeoutId);
```

# SetInterval / clearInterval

```
const intervalId = setInterval(() => {  
  console.log("Hello!");  
}, 1000);  
  
setTimeout(() => {  
  clearInterval(intervalId);  
}, 10000);
```

# Local storage

- Accepts only **primitive** value types
- Browser scoped
- Persists through page reloads
- Key-value pair based

```
localStorage.setItem("3LO", "Pizza?");  
localStorage.getItem("3LO");  
localStorage.removeItem("3LO");  
localStorage.clear();
```

# Local storage – what about complex types though?

```
const person = {  
  name: "Janusz",  
  age: 20,  
  student: true,  
  siblings: [{ name: "Janina", age: 11, relationship: "sister" }]  
};  
  
const makeIntoAPrimitive = JSON.stringify(person);  
localStorage.setItem("3LO", makeIntoAPrimitive);  
const value = localStorage.getItem("3LO"); // it's still a primitive!  
const makeIntoPersonAgain = JSON.parse(value);
```

# Talking with backend and other services

- Fetch API
- Promises
- Async / await

```
async function talkWithExternalService() {  
  const promise = await fetch("https://quotes.rest/qod");  
  const data = await promise.json();  
  console.log(data);  
}  
  
talkWithExternalService();
```

# Workshop #1 random quote generator

- A quote is generated on page load
- User can click a button to generate a new quote
- Style it best you can!

# Workshop #2 stopwatch

- User can play, pause and reset the timer
- Style it best you can!



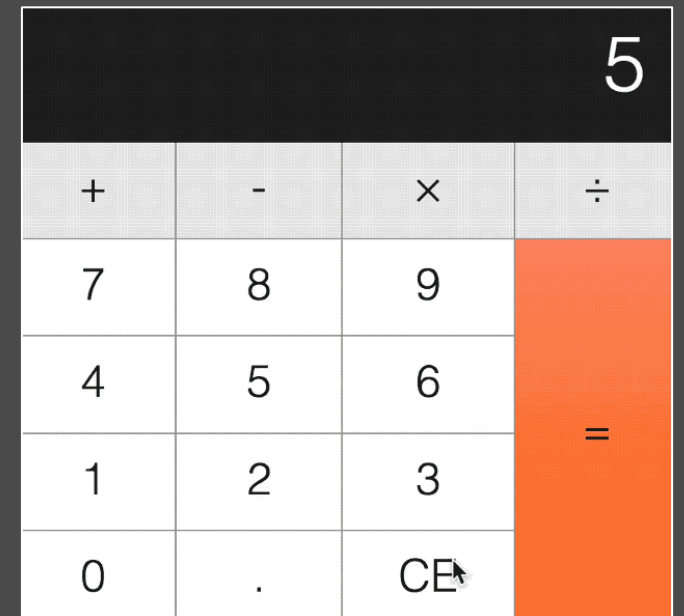
## Workshop #3 todo list

- User can add new todos
- User can edit existing todos
- User can delete a todo
- Todos persist through reload – local storage
- Style it best you can!

# Workshop #4 calculator

Requirements:

- Using **DOM EventListeners**: [https://www.w3schools.com/js/js\\_html\\_dom\\_eventlistener.asp](https://www.w3schools.com/js/js_html_dom_eventlistener.asp)
- Responsive layout (flexbox, grid)



# Workshop #5 hangman \*

## Requirements:

- Vanilla JS (no libraries and frameworks)
- Use predefined dataset: <https://github.com/mackankowski/frontend-bootcamp/blob/master/trainings/js/playground/hangman/data.js>
- Scoreboard in **localStorage**: [https://www.w3schools.com/html/html5\\_webstorage.asp](https://www.w3schools.com/html/html5_webstorage.asp)
- Reponsive layout (flexbox, grid)