Basic Introduction to JavaScript

David Pinezich

Organization

About me

- Education

    - Informatiker Applikationsentwicklung EFZ (BMS / Passerelle)

    - BSc & MSc of Informatics at UZH

    - *Lehrdiplom für Maturitätsschulen (LfM)*

- Work Experience

    - Past

        - Paul Scherrer Institut (PSI), Architonic

        - ti&m

        - Helsana (Lead Webengineering)

    - Apigenio GmbH – Consulting, Teaching, Architecture-Audits

    - Kantonsschule Baden

- Programming Experience

- david.pinezich@apigenio.ch / david.pinezich@uzh.ch

Organization

PLEASE ASK   BREAKS   CERTIFICATE

Course Overview

- Learning Objectives

    - Apply basics of JavaScript syntax

    - Students can describe the function and the main elements of DOM and are able to select and manipulate element with the DOM.

- Learning activities

    - JavaScript fundamentals: Syntax, Data types, variables, functions, conditionals, loops, arrays and objects

    - Objects basics

    - DOM manipulation

    - UI events

Vanilla JavaScript

In the age of many JavaScript frameworks such as jQuery, Angular, React - **vanilla JavaScript** refers to the actual, **pure JavaScript**.

Short History

- JavaScript was invented by **Brendan Eich** in 1995 and became an ECMA standard in 1997.

- **ECMAScript** is the official name of the language.

- ECMAScript versions have been abbreviated to ES1, ES2, ES3, ES5, and ES6.

- Since 2016 new versions are named by year (ECMAScript 2016 / 2017 / 2018 .. 2022).

- **ECMAScript 1 - 6 is fully supported in all modern browsers.**

Short History (2)

Browser Support for ES6 (2015)

Browser	Version	Date
Chrome	51	May 2016
Firefox	52	Mar 2017
Edge	14	Aug 2016
Safari	10	Sep 2016
Opera	38	Jun 2016

Short History (3)

What is ECMAScript?

- **ECMAScript** is the name of the international standard that defines JavaScript

- Developed by Technical Committee 39 (**TC-39**) of Ecma International

- Issued as a **Ecma-262** and ISO/IEC 16262

- Not part of W3C

| Google V8 | Mozilla SpiderMonkey | Microsoft Chakra | Webkit JSCore |

JavaScript Implementations

This course…

- …is based on **ES5/ES6** (JavaScript 2009/2015)

- But I will talk also about newer features (ECMAScript 2015+) :

    - Added let and const

    - Added default parameter values

    - Added Array.find()

    - Added Array.findIndex()

    - And more

- BUT: It will be easy for you to understand the newer version with your base knowledge after this course

- See: JavaScript Versions (w3schools.com)

- Can I Use: https://caniuse.com/sr_es13 (es 2022)

Learn by DOING

Source: https://patti13strick.files.wordpress.com/2015/12/learn-by-doing.jpg?w=540

Material

- Most of the material is linked to web examples
- For the Hands-on-Examples an IDE (Integrated Development Environment) is recommended like Visual Studio Code or Webstorm

- The Material will be shared via Teams or via Email ☺
  - https://github.com/dpinezich/ajs_23/archive/refs/heads/main.zip

Agenda

What are we going to look at:

- Intro

- Output

- Syntax

- Where to

- Variables

- Functions

- Objects

- Conditional Statements

- Arrays

- Loops

- HTML DOM

Intro

JavaScript can

… change HTML content

… change HTML attribute values

… change HTML styles (CSS)

… hide HTML elements

… and much more ☺

… change HTML content

One of many JavaScript HTML methods is getElementById().

The example below tries to "find" an HTML element (with id="demo")

```
document.getElementById("demo").innerHTML="Hello JavaScript";
```

TRY IT >>

… change HTML attribute values

In this example JavaScript changes the value of the src (source) attribute of an <img> tag:

```
<button onclick="document.getElementById('myImage').src='pic_bulbon.gif'">
    Turn on the light
</button>

<img id="myImage" src="pic_bulboff.gif" style="width:100px">

<button onclick="document.getElementById('myImage').src='pic_bulboff.gif'">
    Turn off the light
</button>
```

TRY IT >>

… change HTML styles (CSS)

Changing the style of an HTML element, is a variant of changing an HTML attribute:

```
document.getElementById("demo").style.fontSize = "35px";
```

TRY IT >>

… hide and show HTML elements

Hiding / showing HTML elements can be done by changing the display style:

```
document.getElementById("demo").style.display="none";
```

```
document.getElementById("demo").style.display="block";
```

Hint: The Online-Editor is also mutable, try to add the "show" button ☺
➔ no harm, and no persistence is going to happen

TRY IT >>

JavaScript display possibilities

JavaScript can "display" data in various ways:

- Writing into an HTML element, using innerHTML

- Writing into the HTML output using document.write()

- Writing into an alert box, using window.alert()

- Writing into the browser console, using console.log()

Using innerHTML

To access an HTML element, JavaScript can use the document.getElementById(id) function.

- The id attribute defines the HTML element

- The innerHTML property defines the HTML content

```
<h1>My First Web Page</h1>
<p>My First Paragraph</p>
<p id="demo"></p>

<script>
   document.getElementById("demo").innerHTML = 5 + 6;
</script>
```

TRY IT >>

Using document.write()

For **testing purposes**, it is convenient to use document.write():

```html
<h1>My First Web Page</h1>
<p>My First Paragraph.</p>

<script>
   document.write(5 + 6);
</script>
```

TRY IT >>

Using window.alert()

You can also use an alert box to display data

- careful! It can be very annoying!

```html
<!DOCTYPE html>
<html>
<body>

<h2>My First Web Page</h2>
<p>My First Paragraph.</p>

<script>
    window.alert(5 + 6);
</script>

</body>
</html>
```

TRY IT >>

Using console.log()

For debugging purposes, you can call the console.log() method in the browser to display data.

```html
<!DOCTYPE html>
<html><body>

<h2>Activate Debugging</h2>

<p>F12 on your keyboard will activate debugging.</p>
<p>Then select "Console" in the debugger menu.</p>
<p>Then click Run again.</p>

<script>
   console.log(5 + 6);
</script>

</body></html>
```

TRY IT >>

Syntax

Syntax

JavaScript syntax is the set of rules, **how** JavaScript programs are constructed:

```
var x, y, z;          // Declare Variables, we will talk about let/const later
x = 5; y = 6;         // Assign Values
z = x + y;            // Compute Values
```

Values

The JavaScript syntax defines two types of values:

- Fixed values: called **Literals**
- Variable values: called **Variables**

Literals

The two most important syntax rules for fixed values are:

1. **Numbers** are written with or without decimals:

```
10.50
1001
```

2. **Strings** are text, written withing double or single quotes:

```
"John Doe"

'John Doe'
```

Variables

In most programming languages, **variables** are used to **store** data values.

- JavaScript uses the var keyword to declare variables
- An **equal sign** is used to **assign values** to variables

In this example x is defined as a variable. After, the integer 6 is assigned to the variable

```
var x;

x = 6;
```

Variables - Let

With ES6 it is also possible to declare variables with let. The difference between let and var is:

var can be reassigned and redeclared (multiple times)

let can be reassigned but not be redeclared

The good point: You cannot "accidentally" redeclare a variable

```
let x = "John Doe";
let x = 0;
// SyntaxError: 'x' has already been declared

x = "John Petterson"; // This works
```

Variables - Const

With ES6 it is also possible to declare variables with let. The difference between const and var is:

var can be reassigned and redeclared (multiple times)

const can not be reassigned and not be redeclared

You cannot accidentally reassign or redeclare a variable

```
const PI = 3.141592653589793;
PI = 3.14;     // This will give an error
PI = PI + 10;  // This will also give an error
```

Variables Overview

var

Assignenment:	var x = "John Doe";	✔
Redeclaration:	var x = 0;	✔
Reassignement :	x = 0;	✔

let

Assignenment:	let x = "John Doe";	✔
Redeclaration:	let x = 0;	✖
Reassignement :	x = 0;	✔

const

Assignenment:	const x = "John Doe";	✔
Redeclaration:	const x = 0;	✖
Reassignement :	x = 0;	✖

It is possible to mix and match all three but recommended to use let & const. The only exception is if you must support older browsers

Operators

JavaScript uses arithmetic operators (+ - * /) to compute values:

(5 + 6) * 10

JavaScript uses an assignment operator (=) to assign values to variables:

var x, y;
x = 5;
y = 6;

Expressions

An expression is a combination of values, variables and operators, which computes to a value.

The computation is called an evaluation

Example: 5 * 3 evaluates to 15:

```
5 * 3
```

Expressions (cont.)

Expressions are also able to contain variable values:

```
x * 3
```

The values can be of various types, such as numbers and strings.

Example, "John" + " " + "Doe", evaluates to "John Doe":

```
"John" + " " + "Doe"
```

Keywords

JavaScript keywords are used to identify "actions" to be performed

The var keyword tells the browser to create one or more variables:

```
var x, y;
x = 5 + 6;
y = x * 6;
```

Keywords (cont.)

abstract	arguments	await*	boolean
break	byte	case	catch
char	class*	const	continue
debugger	default	delete	do
double	else	enum*	eval
export*	extends*	false	final
finally	float	for	function
goto	if	implements	import*
in	instanceof	int	interface
let*	long	native	new
null	package	private	protected
public	return	short	static
super*	switch	synchronized	this
throw	throws	transient	true
try	typeof	var	void
volatile	while	with	yield

Words marked with* are new in ECMAScript 5 and 6.

Comments

Not all JavaScript statements are "executed".

Code after double slashes *//* or between */** and **/* is treated as a comment.

Comments are ignored, and will never be executed or evaluated.

```
var x = 5;              // I will not be executed

// z = x + y;           I will not be executed
```

Case sensitive

All JavaScript identifiers are **case sensitive**.

The variables lastname and lastName, are two different variables:

```
var lastname, lastName;
lastname = "Doe";
lastName = "Peterson";
```

JavaScript does not interpret **VAR** or **Var** as the keyword var.

JavaScript and CamelCase

Historically, programmers have many ways of joining multiple words into variable names:

- **Hyphens**: first-name, last-name, master-card, inter-city
- **Underscore**: first_name, last_name, master_card, inter_city
- **Upper CamelCase**: FirstName, LastName, MasterCard, InterCity
- **Lower camelCase**: firstName, lastName, masterCard, interCity

*JavaScript programmers tend to use **Lower camelCase**.*

But the most important rule is to be **consistent** in your choice.

Where to

The HTML DOM
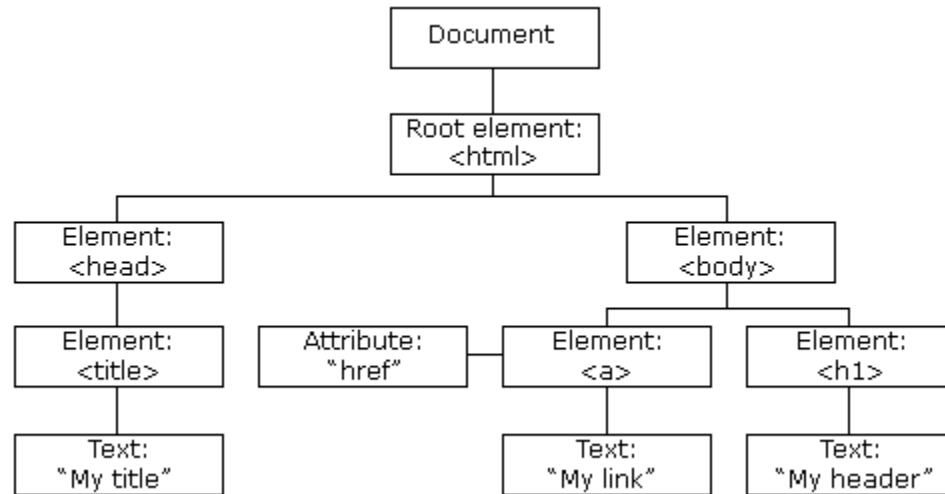
DOM = **D**ocument **O**bject **M**odel

The <script> Tag

In HTML, JavaScript code is inserted between <script> and </script> tags.

```
<script>
    document.getElementById("demo").innerHTML = "My first JavaScript";
</script>
```

TRY IT >>

In \<head\>, \<body\> and \<footer\>

You can place any number of scripts in an HTML document.

Scripts can be placed in the \<body\>, the \<head\> or the \<footer\> section of an HTML page. Or in all of them.

- If your script needs to be loaded before the HTML page, place it in the \<head\> section.

- If your script can be loaded with the HTML page, place it in the \<body\> section.

- If it is ok that the script is loaded after the HTML page, place it in the \<footer\> section (best choice for performance).

TRY IT >>

External JavaScript

External scripts are practical when the same code is used in many different web pages.

JavaScript files have the file extension **.js**. The script will behave as if it was located exactly where the <script> tag is located.

To use an external script, put the name of the script in the src (source) attribute of a <script> tag:

```
<script src="myScript.js"></script>
<script src="myScript2.js"></script>
```

Variables

Variables

JavaScript variables are containers for storing data values.

In this example, x, y, and z are variables declared with the var keyword:

```
var x = 5;
var y = 6;
var z = x + y;
```

Algebra?

In this example, price1, price2, and total are variables:

```
var price1 = 5;
var price2 = 6;
var total = price1 + price2;
```

In programming, just like in algebra, we use variables (like price1) to hold values.

In programming, just like in algebra, we use variables in expressions (total = price1 + price2).

From the example above, you can calculate the total to be **11**.

Identifiers

All JavaScript **variables** must be **identified** with **unique names**.

These unique names are called **identifiers**.

Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume).

The general rule for constructing names for variables (unique identifiers) are:

- Names can contain letters, digits, underscores and dollar signs

- Names must begin with a letter

- Names can also begin with $ and _ (rarely used)

- Names are case sensitive (y and Y are different variables)

- Reserved words (keywords) cannot be used as names

Assignment operator

In JavaScript, the equal sign (=) is an assignment operator, not an "equal to" operator. This is different from algebra!

The following does not make sense in algebra:

```
x = x + 5
```

In JavaScript, however, it makes perfect sense:

It assigns the value of x + 5 to x.

It calculates the value of x + 5 and "puts" the result into x. The value of x is incremented by 5.

Hint: Equal to is == in JavaScript

Data types

JavaScript **variables** can hold numbers like 100 and text values like "John Doe".

In programming, text values are called **text strings**.

JavaScript can handle many types of data, but for now just think of numbers and strings.

- **Strings** are written inside **double** or **single quotes**.

- **Numbers** are written **without quotes**.

If you put a number in quotes, it will be treated as a text string.

```
var pi = 3.14;
var person = "John Doe";
var person = 'John Doe';
```

Declaring (creating) JavaScript Variables

Creating a variable in JavaScript is called "declaring" a variable.

You declare a JavaScript variable with the var keyword:

```
var carName;
carName = "Volvo";
```

After the declaration, the variable has no value (technically it has the value of **undefined**).

But, you can also assign a value to the variable when you declare it:

```
var carName = "Volvo";
```

Arithmetic

As with algebra, you can do arithmetic with JavaScript variables using operators like = and +;

```
var x = 5 + 2 + 3;
```

You can also use strings, but they will be concatenated:

```
var x = "John" + " " + "Doe";
```

Arithmetic Operators

Arithmetic operators perform arithmetic on numbers (literals or variables).

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation
/	Division
%	Modulus (Remainder)
++	Increment
--	Decrement

Exercise 1

- Create a variable called carName, assign the value Volvo to it.

- Create a variable called x, assign the value 50 to it.

- Display the sum of 5 + 10 in a paragraph called "demo", using two variables x and y.

- Create a variable called z, assign x + y to it, and display the result in an alert box.

- On one single line, declare three variables with the following names and values:

  - firstName = "John" / lastName = "Doe" / age = 35

JS CheatSheet

https://htmlcheatsheet.com/js/

And HTML:

https://htmlcheatsheet.com/

Basics ▶

On page script
```
<script type="text/javascript">  ...
</script>
```

Include external JS file
```
<script src="filename.js"></script>
```

Delay - 1 second timeout
```
setTimeout(function () {

}, 1000);
```

Functions
```
function addNumbers(a, b) {
return a + b; ;
}
x = addNumbers(1, 2);
```

Edit DOM element
```
document.getElementById("elementID").innerHTML = "Hello World!";
```

Output
```
console.log(a);            // write to the browser console
document.write(a);         // write to the HTML
alert(a);                  // output in an alert box
confirm("Really?");        // yes/no dialog, returns true/false depe
prompt("Your age?","0");   // input dialog. Second argument is the i
```

Comments
```
/* Multi line
comment */
// One line
```

Data Types ℜ

```
var age = 18;                           // number
var name = "Jane";                      // string
var name = {first:"Jane", last:"Doe"};  // object
var truth = false;                      // boolean
var sheets = ["HTML","CSS","JS"];       // array
var a; typeof a;                        // undefined
var a = null;                           // value null
```
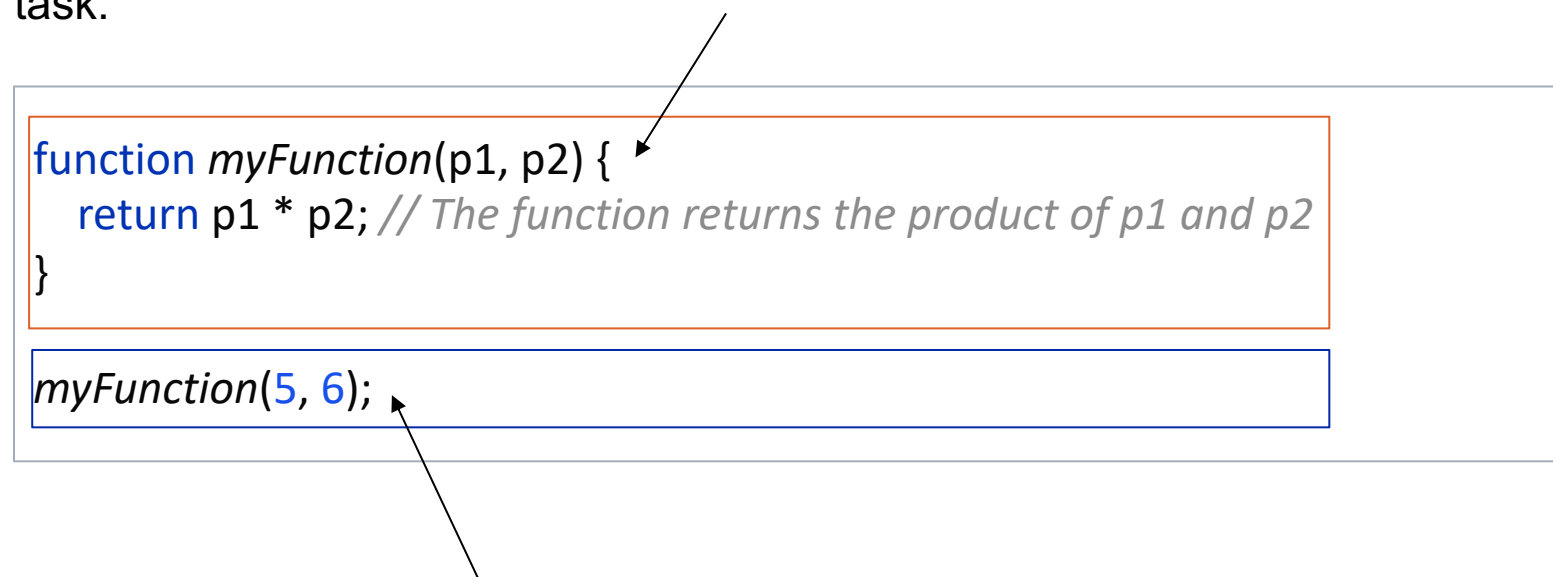
Objects
```
var student = {                    // object name
firstName:"Jane",                  // list of properties and values
lastName:"Doe",
age:18,
height:170,
fullName : function() {            // object function
    return this.firstName + " " + this.lastName;
}
};
student.age = 19;                  // setting value
student[age]++;                    // incrementing
name = student.fullName();         // call object function
```

Loops ∩

For Loop
```
for (var i = 0; i < 10; i++) {
document.write(i + ": " + i*3 + "<br />");
}
var sum = 0;
for (var i = 0; i < a.length; i++) {
sum += a[i];
}               // parsing an array
html = "";
for (var i of custOrder) {
html += "<li>" + i + "</li>";
}
```

While Loop
```
var i = 1;                     // initialize
while (i < 100) {              // enters the cycle if statement is t
i *= 2;                        // increment to avoid infinite loop
document.write(i + ", ");      // output
}
```

Do While Loop
```
var i = 1;                     // initialize
do {                           // enters cycle at least once
i *= 2;                        // increment to avoid infinite loop
document.write(i + ", ");      // output
} while (i < 100)              // repeats cycle if statement is true
```

Break
```
for (var i = 0; i < 10; i++) {
if (i == 5) { break; }         // stops and exits the cycle
document.write(i + ", ");      // last output number is 4
}
```

Continue
```
for (var i = 0; i < 10; i++) {
if (i == 5) { continue; }      // skips the rest of the cycle
document.write(i + ", ");      // skips 5
}
```

Strings ⊗

```
var abc = "abcdefghijklmnopqrstuvwxyz";
var esc = 'I don\'t \n know';   // \n new line
var len = abc.length;           // string length
abc.indexOf("lmno");            // find substring, -1 if doesn't cont
abc.lastIndexOf("lmno");        // last occurance
abc.slice(3, 6);                // cuts out "def", negative values co
abc.replace("abc","123");       // find and replace, takes regular ex
abc.toUpperCase();              // convert to upper case
abc.toLowerCase();              // convert to lower case
abc.concat(" ", str2);          // abc + " " + str2
abc.charAt(2);                  // character at index: "c"
abc[2];                         // unsafe, abc[2] = "C" doesn't work
abc.charCodeAt(2);              // character code at index: "c" -> 99
abc.split(",");                 // splitting a string on commas gives
abc.split("");                  // splitting on characters
128.toString(16);               // number to hex(16), octal (8) or bi
```

Functions

Functions

A JavaScript function is a **block** of code designed to perform a particular task.

```
function myFunction(p1, p2) {
    return p1 * p2; // The function returns the product of p1 and p2
}

myFunction(5, 6);
```

A JavaScript function is executed when "something" **invokes** (calls) it.

TRY IT >>

Function Syntax

A JavaScript function is defined with the function keyword, followed by a **name**, followed by **parentheses ()**.

Function names can contain **letters**, **digits**, **underscores**, and **dollar signs** (same as with variables)

The parentheses may include parameter names **separated by commas**: (parameter1, parameter2, …)

The code to be executed, by the function, is placed inside curly brackets: **{}**

```
function name(parameter1, parameter2, parameter3) {
    // Code to be executed
}
```

Function Syntax (cont.)

Function **parameters** are listed inside the **parentheses ()** in the function definition.

Function **arguments** are the values received by the function when it is invoked.

Inside the function, the **arguments** behave as local variables.

A function is much the same as a procedure or a subroutine in other programming languages.

Parameter(s)

```
function name(parameter1, parameter2, parameter3) {
    window.alert(parameter1);
}
```

argument
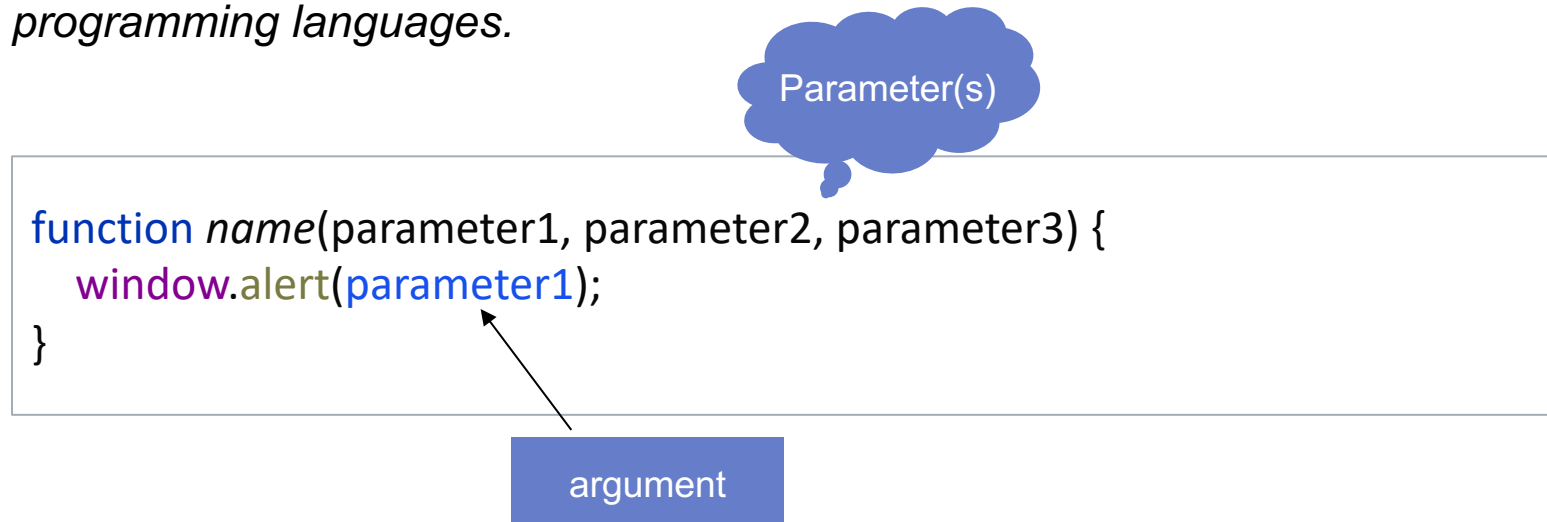
Function Invocation

The code inside the function will execute when "something" **invokes** (calls) the function:

- When an **event** occurs (when a user clicks a button)
- When it is **invoked** (called) from JavaScript code
- **Automatically** (self invoked)

Function Return

When JavaScript reaches a **return statement**, the function will stop **executing** (processing).

If the function was invoked from a **statement**, JavaScript will "return" to execute the code after the invoking statement.

Functions often compute a return value. The return value is "**returned**" back to the "**caller**" (jump back).

```
<script>
   let x = myFunction(4, 3); // Function is called, return value will end up in x
   console.log(x)

   function myFunction(a, b) {
      return a * b;
   }
</script>
```

TRY IT >>

Why Functions?

You can **reuse** code: Define the code once, and use it many times.

You can use the same code many times with different (input) arguments, to produce different results.

Exercise 2

- Convert Fahrenheit to Celsius:

- C = (5/9) * (F – 32)

- Create a function called "toCelcius" with the Fahrenheit values as parameter
- Try it in an Editor (Visual Studio Code / Webstorm) or [Online](#)

Exercise Solution

Convert Fahrenheit to Celsius:

```
<script>

  function toCelsius(fahrenheit) {
    return (5/9) * (fahrenheit - 32);
  }

  document.getElementById("demo").innerHTML = toCelsius(88);

</script>
```

The () operator invokes the function

Using the former example, **toCelsius** refers to the function object, and **toCelsius()** refers to the function result.

➔ Spot the small difference of the parentheses **()**

Accessing a function **without ()** will return the **function object** instead of the function result

```
<script>
 function toCelsius(fahrenheit) {
   return (5/9) * (fahrenheit - 32);
 }

 document.getElementById("demo").innerHTML = toCelsius;
</script>
```

TRY IT >>

Function used as variable values

Functions can be used the same way as you use variables, in all types of formulas, assignments and calculations.

Function used as variable values (cont.)

Functions can be used the same way as you use variables, in all types of formulas, assignments and calculations.

Instead of using a variable to store the return value of a function:

```
<script>

 function toCelsius(fahrenheit) {
   return (5/9) * (fahrenheit - 32);
 }

 let x = toCelsius(88);
 let text = "The temperature is " + x + " Celsius";
</script>
```

Function used as variable values (cont.)

Or even direct while calling:

```
<script>

  function toCelsius(fahrenheit) {
    return (5/9) * (fahrenheit - 32);
  }

  let text = "The temperature is " + toCelsius(88) + " Celsius";
</script>
```

Exercise 3

1. Create and execute a function with an alert box that says "Hello World!"

2. Modify that function so it return that message (as opposed to the alert box).

3. Use that function to change an inner HTML's value of an element with the ID "demo"

- Try it in an Editor (Visual Studio Code / Webstorm) or Online

Objects

Real life objects, properties and methods

Object	Properties	Methods
	car.name = Fiat	car.start()
	car.model = 500	car.drive()
	car.weight = 850kg	car.brake()
	car.color = white	car.stop()

All cars have the same **properties**, but the property **values differ** from car to car.

All cars have the same **methods**, but the methods are **performed** at **different times**.

Objects

You have already learned that JavaScript variables are containers for data values.

This code assigns a **simple value** (Fiat) to a **variable** named car:

```
<script>
  let car = "Fiat";
</script>
```

Objects (cont.)

Objects are variables too.

But objects can contain **many values**.

The values are written as **name:value pairs** (name and value separated by a colon).

This code assigns **many values** (Fiat, 500, green) to a **variable** named car:

```
<script>
  let car = {type: "Fiat", model: "500", color: "green"};
</script>
```

Object definition

You define (and create) a JavaScript object with an object literal:

```
<script>
 let person = {firstName: "John", lastName: "Doe", age: 50,
eyeColor: "blue" };
</script>
```

Spaces and line breaks are **not important**.

An object definition can span over **multiple lines**.

Object Properties

The **name:values** pairs in JavaScript objects are called properties:

Property	Property Value
firstName	John
lastName	Doe
age	50
eyeColor	blue

Accessing Object Properties

You can access object properties in two ways:

```
objectName.propertyName
```

Or:

```
objectName["propertyName"]
```

Object Methods

Objects can also have methods (similar to functions).

Methods are actions that can be performed on objects.

Methods are stored in properties as function definitions.

Property	Property Value
firstName	John
lastName	Doe
age	50
eyeColor	blue
fullName	function() {return this.firstName + " " + this.lastName;}

Object Methods

A method is a **function stored** as a **property**.

```
let person = {
  firstName: "John",
  lastName: "Doe",
  id: 5566,
  fullName: function () {
    return this.firstName + " " + this.lastName;
  }
};
```

The "this" Keyword

In a function definition, this refers to the **"owner"** of the function.

In the example before, this is the person object that "owns" the fullName method.

In other words, this.firstName means the firstName property of **this object**.

Accessing Object Methods

You can access object methods in two ways:

```
objectName.methodName()
```

Or:

```
name = this.fullName()
```

Exercise 4

1. Create a "person" object with first name "John", last name "Doe"

2. Alert "John" by extracting information from the person object.

3. Add the property "country" with the value "Norway" to the person object.

4. Add the property "age" with the value 50 to the person object.

5. Finally, alert ("John is 50 and lives in Norway").

- Try it in an Editor (Visual Studio Code / Webstorm) or <u>Online</u>

Exercise Solution

```javascript
let person = {
  firstName: "John",
  lastName: "Doe",
  country: "Norway",
  age: 50,
};

alert(person.firstName + " is " + person.age + " and lives in " +
person.country);
```

Conditional Statements

Conditional statements

Very often when you write code, you want to perform **different actions** for **different decisions**.

You can use **conditional statements** in your code to do this.

In JavaScript we have the following conditional statements:

- Use **if** to specify a block of code to be executed, if a specified condition is **true**

- Use **else** to specify a block of code to be executed, if the **same condition** is **false**

- Use **else if** to specify a new condition to test, if the **first condition** is **false**

- Use **switch** to specify **many alternative** blocks of code to be executed

The if statement

Use **if** to specify a block of code to be executed, if a specified condition is **true**

```
if (condition) {
  // block of code to execute if the condition is true
}
```

The if statement (cont.)

Make a "Good day!" greeting if the hour is less than 18:00:

```
if (hour < 18) {
  greeting = "Good day!"
}
```

Conditional statements

Use **else** to specify a block of code to be executed, if the **same condition** is **false**

```
if (condition) {
  // block of code to execute if the condition is true
} else {
  // block of code to execute if the condition is false
}
```

The if statement (cont.)

If the hour is less than 18:00, create a "Good day!" greeting, otherwise "Good evening":

```
if (hour < 18) {
  greeting = "Good day!"
} else {
  greeting = "Good evening!"
}
```

Conditional statements

Use **else if** to specify a new condition to test, if the **first condition** is **false**

```
if (condition1) {
  // block of code to execute if the condition1 is true
} else if (condition2) {
 // block of code to execute if the condition1 is false and condition2 is true
} else {
 // block of code to execute if the condition1 and condition2 is false
}
```

The if statement (cont.)

If time is less than 10:00, create a "Good morning!" greeting, if not, but time is less than 20:00, create a "Good day!" greeting, in all other cases, a "Good evening" greeting:

```javascript
if (hour < 10) {
  greeting = "Good morning!"
} else if (hour < 20) {
  greeting = "Good day!"
} else {
  greeting = "Good evening!"
}
```

It is important to note, that JavaScript works "Top-Down" here!

Conditional statements

Use **switch** to specify **many alternative** blocks of code to be executed

```
switch (hour) {
  case "10":
   alert("It is 10 o'clock");
  case "11":
   alert("It is 11 o'clock");
  case "12":
   alert("It is 12 o'clock");
  case "13":
   alert("It is 1 o'clock");
}
```

Exercise 4

1. Create an if statement that uses the variable numberOfTheMonth to alert the user. Conditions:

   1. When numberOfTheMonth is less than 3 → Winter
   2. When numberOfTheMonth is less than 6 → Spring
   3. When numberOfTheMonth is less than 9 → Summer
   4. When numberOfTheMonth is less than 12 → Fall

- Try it in an Editor (Visual Studio Code / Webstorm) or Online

Exercise Solution

```javascript
if (numberOfTheMonth < 3) {
  alert("Winter");
} else if (numberOfTheMonth < 6) {
  alert("Spring");
} else if (numberOfTheMonth < 9) {
  alert("Summer");
} else {
  alert("Fall");
}
```

Arrays

What is an array?

An array is a special variable, which can hold more than one value at a time.

If you have a list of items (a list of car names, for example), storing the cars in a single variable could look like this:

```
let car1 = "Fiat";
let car2 = "Tesla";
let car3 = "Subaru";
```

What is an array? (cont.)

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 3000?

The solution is an array!

An array can hold **many values under a single name**, and you can access the values by referring an **index number**.

Creating an array

Using an array literal is the easiest way to create a JavaScript Array

Syntax:

```
let array_name = [item1, item2, item3, ...];
```

Creating an array (cont.)

Example:

```
let cars = ["Fiat", "Tesla", "Subaru"];
```

Spaces and line breaks are not important.

A declaration can span multiple lines:

```
let cars = [
    "Fiat",
    "Tesla",
    "Subaru"
    ];
```

Using the keyword new

The following example also creates an Array, and assigns values to it:

```
let cars = new Array("Fiat", "Tesla", "Subaru");
```

Access the elements of an array

You can access an array element by referring to the **index number**.

This statement accesses the value of the **first element** in cars:

```
let specialName = [0];
```

Example:

```
let cars = ["Fiat", "Tesla", "Subaru"];
document.getElementById("demo").innerHTML = cars[0];
```

Changing an array element

This statement changes the value of the first element in cars:

```
let cars = ["Fiat", "Tesla", "Subaru"];
cars[0] = "Opel";
document.getElementById("demo").innerHTML = cars[0];
```

Access the full array

With JavaScript, the full array can be accessed by referring to the array name:

```
let cars = ["Fiat", "Tesla", "Subaru"];
document.getElementById("demo").innerHTML = cars;
```

Loops

Loops

Loops are handy, if you want to run the same code **repeatedly**, each time with a different value.

Often this is the case when working with arrays:

Loops

Instead of writing:

```
text += cars[0] + "<br />";
text += cars[1] + "<br />";
text += cars[2] + "<br />";
text += cars[3] + "<br />";
text += cars[4] + "<br />";
text += cars[5] + "<br />";
```

You can write:

```
let i;
for (i = 0; i < cars.length; i++) {
  text += cars[i] + "<br />";
}
```

TRY IT >>

Loops

JavaScript supports **different kinds of loops:**.

- **for** – loops through a block of code several times
- **for/in** – loops through the properties of an object
- **for/of** – loops through the values of an **iterable** object
- **while** – loops through a block of code while a specified condition is true
- **do/while** – also loops through a block of code while a specified condition is true

The for loop

The for loop has the following syntax:

```
for (statement 1; statement 2; statement 3) {
  // code block to be executed
}
```

Statement 1: Is executed (one time) before the execution of the code block.

Statement 2: defines the condition for executing the code block.

Statement 3: Is executed (every time) after the code block has been executed.

The for loop (cont.)

Example

```
for (let i = 0; i < 5; i++) {
  text += "The number is " + i + "<br />";
}
```

From the example above, you can read:

Statement 1: sets a variable before the loop starts (**let i = 0**).

Statement 2: defines the for the loop to run (i needs to be less than 5, **i < 5**).

Statement 3: Increases a value (**i++**) each time the code block in the loop has been executed.

TRY IT >>

Statement 1

Normally you will use statement 1 to initialize the variable used in the loop (let i = 0).

This is not always the case; JavaScript does not care. Statement 1 is optional.

You can initiate many values in statement 1 (separated by comma)

```javascript
for (let i = 0, len = cars.length, text = i; i < 5; i++) {
  text += "The number is " + i + "<br />";
}
```

Statement 1

And you can omit statement 1 completely, if the values are set beforehand.

```javascript
let i = 0, len = cars.length, text = i;
for (; i < 5; i++) {
  text += "The number is " + i + "<br />";
}
```

TRY IT >>

Statement 2

Often statement 2 is used to evaluate the condition of the initial variable.

This is not always the case; JavaScript does not care. Statement 2 is optional as well.

If statement 2 returns true, the loop will start over again, if it returns false, the loop will end.

Statement 3

Often statement 2 increments the value of the initial variable.

This is not always the case; JavaScript does not care. Statement 3 is optional as well.

Statement 3 can do anything like negative increment (i--), positive increment (i = i + 15), or anything else.

Statement 3 can also be omitted (like when you increment your values inside the loop):

```javascript
let i = 0, len = cars.length;
for (; i < 5;) {
  text += "The number is " + i + "<br />";
  i++;
}
```

TRY IT >>

Exercise 5

1. Create a loop that runs from 0 to 9. It starts with the value 0 and adds 1 after each loop. Display the numbers by using document.write() and with a line break in between each other.

- Try it in an Editor (Visual Studio Code / Webstorm) or Online

Exercise Solution

```javascript
for (let i = 0; i < 10; i++) {
  document.write(i + "<br />")
}
```

The for in loop

The JavaScript **for in** statement loops through the **properties** of an object:

```
for (key in object) {
  // code block to be executed
}
```

The for in loop (cont.)

How would that look like for our "John Doe" Object?

```javascript
const person = {firstName:"John", lastName:"Doe", age:25};

let text = "";
for (let x in person) {
  text += person[x];
}
```

Explanation:

- The for in loop **iterates** over a **person object**
- Each **iteration** returns a **key (x)**
- The key is used to access the **value of the key**
- The value of the key is person[x]

TRY IT >>

The for of loop (ES6 feature)

The JavaScript **for of** statement loops through the **values** of an **iterable object.**

```
for (variable of iterable) {
  // code block to be executed
}
```

variable - For every iteration the value of the next property is assigned to the variable. Variable can be declared with const, let, or var.

iterable - An object that has iterable properties.

The for of loop (cont.)

Back to our car example:

```
const cars = ["Fiat", "Tesla", "Subaru"];

let text = "";
for (let x of cars) {
  text += x;
}
```

TRY IT >>

While Loop

The while loop loops through a block of code as long as a specified condition is true.

```
while (condition) {
  // code block to be executed
}
```

Example:

```
while (i < 10) {
  text += "The number is " + i;
  i++;
}
```

Important: If you forget to increase the variable used in the condition, the loop will never end. This will crash your browser.

TRY IT >>

Do While Loop

The do while loop is a variant of the while loop. This loop will execute the code block once, **before checking if the condition is true**, then it will repeat the loop as long as the condition is true.

```
do {
  // code block to be executed
}
while (condition);
```

Example:

```
do {
  text += "The number is " + i;
  i++;
}
while (i < 10);
```
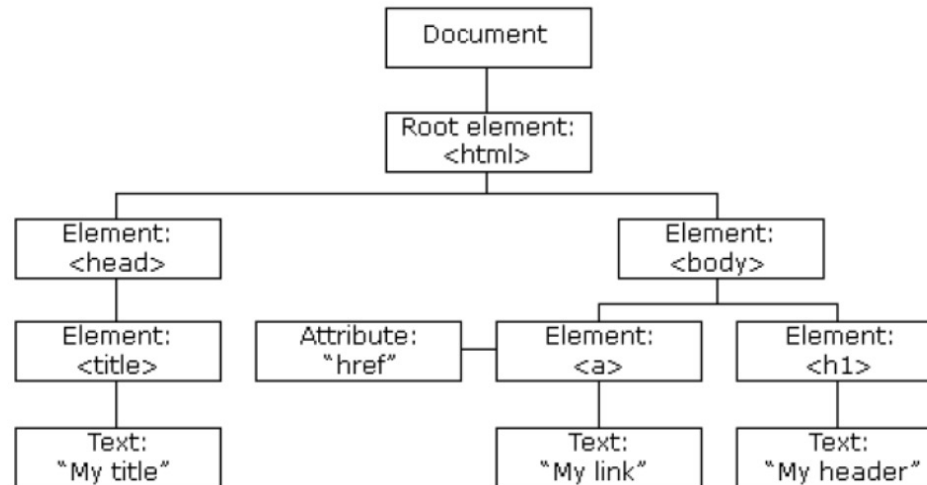
TRY IT >>

HTML DOM

Topics

- Intro

- Methods

- Document

- Elements

- HTML

- CSS

The HTML DOM

When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page.

The **HTML DOM** model is constructed as a tree of **Objects**:

The HTML DOM

With the object model, JavaScript gets all the power it needs to create dynamic HTML:

- JS can **change** all the HTML **elements** in the page
- JS can **change** all the HTML **attributes** in the page
- JS can **change** all the **CSS styles** in the page
- JS can **remove existing HTML elements** and attributes
- JS can **add new HTML elements** and attributes
- JS can **react** to all **existing HTML events** in the page
- JS can **create new HTML events** in the page

What is the HTML DOM

The HTML DOM is a standard **object model** and **programming interface** for HTML. It defines:

- The HTML elements as objects

- The properties of all HTML elements

- The methods to access all HTML elements

- The events for all HTML elements

In other words: **The HTML DOM is a standard for how to get, change, add, or delete HTML elements**.

HTML DOM Methods

- HTML DOM methods are actions you can perform (on HTML elements).

- HTML DOM properties are values (of HTML elements) that you can set or change.

HTML DOM Methods

The following example changes the content (the **innerHTML**) of the **\<p\>** element with **id="demo"**:

```
<html lang="en">
<body>
  <p id="demo"></p>

  <script>
    document.getElementById("demo").innerHTML = "Hello World!";
  </script>
</body>
</html>
```

In this example, getElementById is a **method**, while innerHTML is a **property**.

The getElementById Method

The most common way to access an HTML element is to use the **id** of the element.

➔ This makes sense, because an **id** needs to be **unique** on a page

In the example before, the getElementById method used id="demo" to find the element.

The innerHTML Property

The easiest way to get the content of an element is by using the innerHTML property.

The innerHTML property is useful for getting or replacing the content of HTML elements.

The HTML DOM Document Object

The **document object** represents your web page.

If you want to access any element in an HTML page, you always start with accessing the document object.

On the next slides are some examples of how you can use the document object to access and manipulate HTML.

Finding HTML Elements

Method	Description
document.getElementById(id)	Find an element by element id
document.getElementsByTagName(name)	Find elements by tag name
document.getElementsByClassName(name)	Find elements by class name

Changing HTML Elements

Property	Description
element.innerHTML = *new html content*	Change the inner HTML of an element
element.attribute = *new value*	Change the attribute value of an HTML element
element.style.*property* = *new style*	Change the style of an HTML element

Adding or Deleting Elements

Method	Description
document.createElement(*element*)	Create an HTML element
document.removeChild(*element*)	Remove an HTML element
document.appendChild(*element*)	Add an HTML element
document.replaceChild(*new, old*)	Replace an HTML element
document.write(*text*)	Write into the HTML output stream

Further details

You can find many more details on:

JavaScript DOM Document (w3schools.com)

Changing HTML Content

The easiest way to modify the content of an HTML element is by using the innerHTML property.

To change the content of an HTML element, use this syntax:

```
document.getElementById(id).innerHTML = "new HTML"
```

Changing HTML Content (cont.)

Let's see this in action on www.digitec.ch with (with your F12 console):

```
document.getElementById("pageContent").innerHTML = "<h1>Hello
JavaScript course</h1>"
```

Hint: No harm has been done with this experiment, since it is only viewable for you. But if it was in the source-code of digitec.ch, exactly the same would happen ☺

Changing HTML Content (cont.)

Here is a more generic example:

The HTML document contains an <img> element with id="myImage"

We use the HTML DOM to get the element with id="myImage"

A JavaScript changes the src attribute of that element from "smiley.gif" to "landscape.jpg"

```html
<img id="myImage" src="smiley.gif">
<script>
   document.getElementById("myImage").src = "landscape.jpg"
</script>
```

TRY IT >>

Changing CSS

To change the style of an HTML element, use this syntax:

```
document.getElementById(id).style.property = "new style"
```

Changing CSS (cont.)

The following example changes the style of a <p> element:

```html
<p id="p1">Hello World!</p>
<p id="p2">Hello World!</p>

<script>
    document.getElementById("p2").style.color = "blue";
    document.getElementById("p2").style.fontFamily = "Arial";
    document.getElementById("p2").style.fontSize = "larger";
</script>
```

TRY IT >>

Using Events

The HTML DOM allows you to **execute code** when an **event occurs**.

Events are generated by the browser when "**things happen**" to HTML elements.

Using Events (cont.)

What are events?

- When a user clicks the mouse

- When a web page has loaded

- When an image has been loaded

- When the mouse moves over an element

- When an input field is changed

- When an HTML form is submitted

- When a user strokes a key

- … and many more

Find out more about events:

https://www.w3schools.com/js/js_htmldom_events.asp

Further information and reference

This course is based on the content of:

JavaScript Tutorial (w3schools.com)

And

https://wiki.selfhtml.org/wiki/JavaScript/Tutorials

And former JavaScript Courses at UZH by Alain A. Asik

JS Quiz

Quiz

Finally, a little quiz ☺

W3Schools Quiz v3.0

One hint: Mathematics are done with the "Math" object, and can be looked up here: JavaScript Math Object (w3schools.com)

End