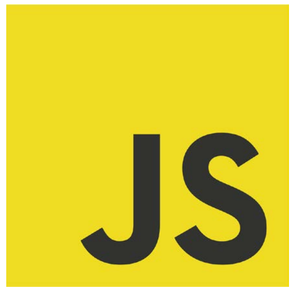
LiveScript
JScript
Ecmascript



JavaScript

History of JavaScript

- **JavaScript** was developed in 1995 for Netscape Navigator
- In 1996 **JScript** was released by Microsoft.
- In 1996 **ECMAScript** developed and it was a standardised version of the language.
- In 1999 **ES3** is published. Adds a number of features and improvements.
- In 2009 **ES5** is published. Adds a number of features and improvements.
- **Node.js** was released in 2009. For the first time JavaScript was running on the server.
- **ES6** was released in 2015. Changes the syntax a lot. Adds a number of features and improvements.

Java vs. JavaScript

- JavaScript and Java has almost nothing in common.
- The name for JavaScript was chosen because of marketing reasons
- Java is a strongly-typed compiled language running on JVM
- JavaScript is a dynamically-typed interpreted language running in the browser and Node.js

What does JavaScript do?

- All the changes in the browser are happening because of JavaScript
- Components of a traditional webpage:
- **HTML** - structure
- **CSS** - style
- **JavaScript** - action
- Webapps rely much more on JavaScript e.g. using React you will have just a few .html files and a lot of .jsx files.

Server side vs Client side

- Browsers run only one language: JavaScript.
- Node.js runs JavaScript code on the server.
- Typical website has a **frontend** and **backend**.
- Frontend is what the user sees and backend is what the user doesn’t see.
- Backend code is secure and frontend code is not.
- Checking passwords, credentials, calling the database all happens in the backend.

DOM (Document Object Model)

- **Document** - webpage
- **Object** - component (node)
- **Model** - set of terms

- **.getElementById(“id“) - Gets element by id.**
- **.getQuerySelector(“#header“);**
- **div.classList.remove(“foo”);**
- **div.classList.add(“anotherclass”)**;
- **.textContent** property represents the text content of a node and its descendants.
- **.appendChild()** method appends a node as the last child of a node.

Naming Variables

- JavaScript variables are named using camelCase naming convention e.g. var secretInformation.
- Other naming conventions:
- **snake_case**: e.g. var secret_information
- **UpperCaseCamel**: var SecretInformation
- **Mixed_Case_With_Underscores**: var Secret_Information

Reserved Words

- abstract
- arguments
- boolean
- break
- byte
- case
- catch
- char
- class
- const
- continue
- debugger
- default
- delete
- do
- double
- else
- num
- eval
- export
- extends
- false
- final
- finally
- float
- for
- function
- goto
- if
- implements
- import
- in
- instanceof
- int
- interfacelet

Reserved Words

- long
- native
- new
- null
- package
- private
- protected
- public
- return
- short
- static
- super
- switch
- synchronised
- this
- throw
- throws
- transient
- true
- try
- typeof
- var
- void
- volatile
- while
- with
- yield

Variables

primitives

- **String** *“Hello world!“*
- **Number** *42*
- **Boolean** *true || false*
- **null** - *Contains nothing*
- **undefined** - *Accsidently contains nothing*
- **NaN** - *Not a number*

complex

- **Array** *[1 ,2 ,”John”]*
- **Object** *{ name: “John”, age: 30 }*
- **Function** *function() {}*

JavaScript Instructions

- A **Script** is made up of a **series of statements**. Each statement is like a step in a recipe.
- Scripts contain very **precise instructions**. For example, you might specify that a value must be remembered before creating a calculation using that value.
- **Variables** are used to temporarily store pieces of information used in the script.
- **Arrays** are special types of variables that store more than one piece of related information.
- **Expressions** rely on operators to calculate a value.

Window Object

- **window.location** - Current URL of window object
- **window.screenX** - X-coordinate of pointer, relative to top left corner of screen (px)
- **window.pageYOffset** - Distance document has been scolled vertically (px)
- **window.document** - Reference to document object, current page.
- **window.alert()** - Creates a dialogbox with message
- **window.open()** - Opens new browser window with URL specified as parameter

Document Object

- **document.title** - Title of a current document
- **document.lastModified** - Date on which document was last modified
- **document.url** - Returns string containing URL of current document
- **document.domain** - Returns domain or current document
- **document.getElementById(‘id’);**
- **document.querySelector(‘.class’);**
- **document.querySelectorAll(‘p’);**
- **document.createElement(‘li’);**
- **document.createTextNode(‘Hello world!’);**

String Object

- **.length** - returns number of characters in the string
- **.toUpperCase()** - changes string to uppercase characters
- **.toLowerCase()** - changes string to lowercase characters
- **.indexOf('e')** - Returns index number of the first time a character 'e' is found.
- **.split(" - ")** - Splits each time " - " is found, then stores each individual part in array
- **.trim()** - removes white space from start and end of the string
- **.replace('a', 'b');** - Finds "a" and replace it with "b";

Number Object

- **isNan()** - checks if the value is not a number
- **.toFixed();** - Rounds to specified number of decimal places (returns a string)
- **.toPrecision()** - Rounds a total number of places (returns a string)
- **.toExponential()** - Returns a string representing the number in exponential notation

```
var balance = 34.567

balance.toFixed(2) // returns “34.57”
```

Math Object

- **Math.PI** - returns pi (3.14159265359).
- **Math.round()** - Rounds number to the nearest integer.
- **Math.sqrt(n)** - Returns square root of positivenumber.
- **Math.ceil()** - Rounds number up to the nearest integer.
- **Math.floor()** - Rounds number down to the nearest integer.
- **Math.random()** - Generates random number between 0 and 1.

Arrays

- **.length** - returns number of items in an array.
- **.sort()** - sorts an array alphabetically.
- **.reverse()** - reverses the elements in an array.
- **.join()** - converts an array to string (separator in ()).
- **.push()** - Adds item(s) to the end of array and returns number of items in it.
- **.shift()** - Removes first item from array and returns it.
- **.pop()** -Removes last item from array and returns it.
- **.unshift()** - Adds item(s) to start of array and returns new length of it.
- **.splice(0, 1);** 0 - position, 1-how many elements to remove.
- **.forEach()** - Executes a function once for each item in array.
- **.filter()** - Creates new array with items that pass a test specified by function.
- **.concat()** - Creates new array containing this array and other arrays/values.
- **.map()** - Calls a function on each item in array and creates new array with results.

Date Object

- **var** today = **new Date()**;
- **var** year = today**.getFullYear()**;

- **.toISOString()** - Returns a date as a string value in ISO format.
- **.toTimeString()** - Shows the time. Number of miliseconds since Jan 1, 1970.
- **.toDateString()** - Returns “date“ as a human-readable string.
- **.getTimezoneOffset()** - Returns a time zone offset in mins for locale.
- **.getHours()** - Returns the hour (0-23).
- **.setHours()** - Sets the hour.
- **.getMinutes()** - Returns the minutes (0-59).
- **.setMinutes()** - Sets the minutes.

Event Object

- **.target** - The target of the event (the most specific element interacted with).
- **.type** - Type of the event that was fired.
- **.preventDefault()** - Cancel default behavior of the event
- **.stopPropagation()** - Stops event from bubbling or capturing any futher

```
function(event) {
    event.preventDefault();
    // do something
}
```

Events

- **UI Events -** load, resize, scroll, error, unload
- **Keyboard events -** keydown, keyup, keypress
- **Mouse events -** click, dblclick, mousedown, mouseup, mousemove, mouseover...
- **Focus events -** focus, blur
- **Form events -** input, submit, reset, change, select, copy, paste, cut
- **Mutation events -** DOMSubtreeModified, DOMNodeInserted...

To get a code of the key: http://keycode.info/

HTML5 Events

- **DOMContentLoaded** - Event fires when the DOM tree is formed (images, css and javascript might still be loading)
- **hashchange** - Event fires when URL hash(#) changes (without the entire window refreshing). Event object will have oldURL and newURL properties
- **beforeunload** - Event fires on the window object before the page is unloaded.

Timing Events

- **setTimeout(function, milliseconds)**
  - Executes a function, after waiting a specified number of milliseconds.

  window.clearTimeout(timeoutVariable)

- **setInterval(function, milliseconds)**
  - Same as setTimeout(), but repeats the execution of the function continuously.

  window.clearInterval(timerVariable)

Event Handlers

- **HTML event handler attributes** - onclick=”alert(‘clicked!’);”

Click me

- **Traditional DOM event handlers** - element.onevent = functionName;

window.onload = function() {};

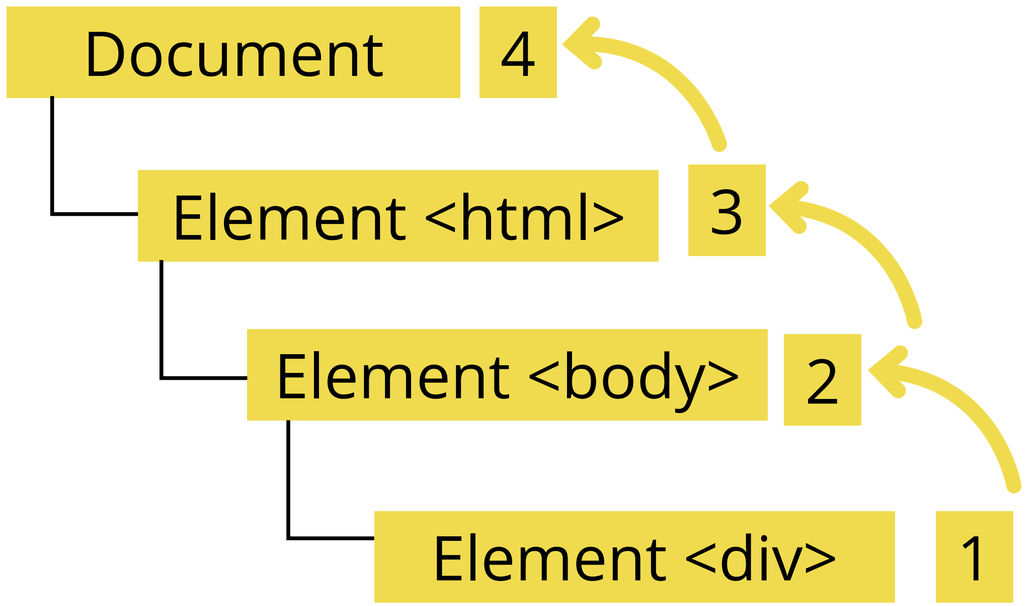
- **Event Listeners** - element.addEventListener(‘event’, functionName [, boolean]);

button.addEventListener(‘click‘, update, false);

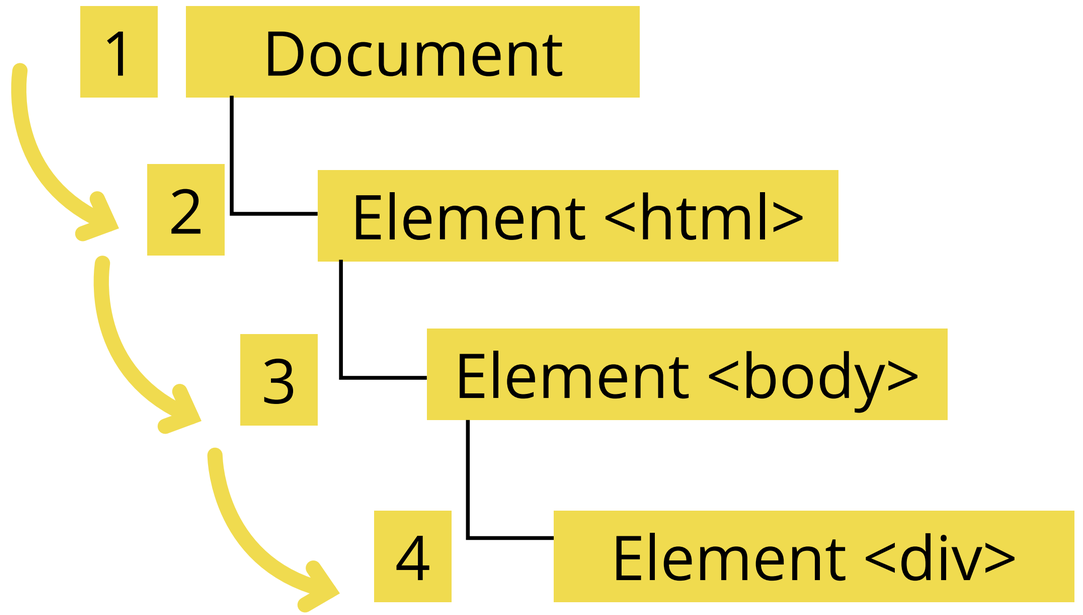
button.removeEventListener(‘click‘, update, false);

Event Flow

- Bubbling



- Capturing



Event Delegation

Add EventListener to a **parent element** which will always be on a page. And when event occurs, find the child element you need.


```
document.querySelector('ul').
addEventListener( 'click',
doSomething );


function doSomething(e) {
   if(e.target && e.target.
nodeName == "LI") {
      // do something...
   }
}
```

Comparison operators

- **==** - Is equal to
- **===** - Strict equal to
- **!=** - Is not equal to
- **!==** - Strict not equal to
- **>** - Greater than
- **<** - Less than
- **>=** - Greater than or equal to
- **<=** - Less than or equal to

“13“ === 13 returns false

“”13“ == 13 returns true

0 == false returns true

0 === false returns false

undefined == null returns true

Logical operators

&&	||	!
And	Or	Not
((2>3) && (2 >1)) *returns false*	*((2>3) || (2 >1))* *returns true*	*!(2>3)* *returns true*

Conditional statements

- **if ... else**

```
if(condition) {
   // do something...
 } else {
   // do something else...
}
```

- **switch**

```
switch(variable) {
   case 1:
      // do something...
      break;
   case 2:
      // do something else...
      break;
   default:
      // do default...
      break;
}
```

- **Ternary Operator**

```
condition ? expr1 : expr2


(a>b) ? c = 1 : c = 2;
```

Loops

- **For loop**

```
for(var i=0; i<10; i++) {
    // do something
}
```

- **While loop**

```
while(i<10) {
    // do something
    i++;
}
```

- **Do .. while loop**

```
do {
    // do something
    i++;
} while(i<10);
```

It is easy to create infinite loop if you forget to increment the index.

Truthy and Falsy

Due to **type coersion**, every value in JavaScript can be treated as if it were **true** or **false**

JS treats everything as **true** except:

- **undefined**
- **null**
- **0**
- **“ “**
- **false**
- **NaN**

} **false**

Functions

Global functions:

- **isNaN()** - Checks if value is a number
- **decodeURI()** - Decodes URI
- **encodeURI()** - Encodes URI
- **parseFloat()** - parses a string and returns a floating point number.
- **parseInt()** - Parses a string and returns an integer.

```
function addNumbers(a, b) {
    return a + b;
}

addNumbers(2,5)  // returns 7
```

To create **local variables** - declare them inside the function. Omitting the **var** keyword will make them global.

IIFE

Immediately Invoked Function Expressions runs themselves.

Can’t be called, because it doesn’t have a name.

```
( function( ) { } ) ( );
```

```
(function (a) {
    conslole.log(a);
} (“Hello world!“));

// returns “Hello world!“
```

AJAX

- Asynchronous JavaScript and XML
- Allows to get data from the server without reloading the page.
- We need a web server to run AJAX

```
function loadDoc() {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
     document.getElementById("demo").innerHTML =
this.responseText;
    }
  };
  xhttp.open("GET", "ajax_info.txt", true);
  xhttp.send();
}
```

this

In the global execution context (outside of any function), **this** refers to the global object.

```
function hello() {
    console.log(this);
}
// this points to
Window object
```

In the object method **this** can, but does not always, point to its own object.

```
var john = {
    sing: function() {
        console.log(this)
    },
    name: “John Lennon“
}
john.sing();
```

Here we call **paul.sing()**, it will call john.sing(), but will set the **calling context** to paul.

This points to paul object

```
var paul = {
    sing: john.sing,
    name: “Paul McCartney“
}
paul.sing();
```

bind

We can add **bind()** call to ensure we have control of the value of **this**

```
var john = {
    name: “John Lennon“,
    sing: function(song) {
        console.log( this.name);
    }
}
var paul = john.sing.bind({ name: “Paul McCartney”});
paul(); // returns “Paul McCartney“
```

jQuery

- Text file which contains 10 thousand lines of JS
- jQuery is JS
- Everything that you can do in jQuery you can do in JS
- Cross-browser
- Can be quite big: 100KB

Javascript

```
document.getElementById('video-player').
style.visibility = "hidden";
```

jQuery

```
$("#video-player").hide();
```

Objects

- Objects are complex types.
- Using **=** (assigment) will **not** create a new independent object. It will create a second variable which **points to the same object**.
- All objects are unique. So object **comparisons** always return false.

```
// jQuery

var newObj = jQuery.extend(true, {}, oldObj);

// JSON

var newObj = JSON.parse( JSON.stringify ( oldObj ) );

// ES5

var newObj = Object.clone(oldObj);

// Underscore

_.extend( newObj, oldObj);
```

Module Pattern

The module pattern is a JavaScript construct than uses several techniques to acheive a degree of encapsulation:

- **functions as first class objects**
- **anonymous functions**
- **closure**
- **self-executing functions**

```
var obj = function() {
    var name = “John Lennon“; // private
    return {
        beatle: name //public
    }
} () ;
console.log (obj.name); // undefined
console.log (obj.beatle) // “John Lennon“
```

JSON

JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate.

- **JSON.parse()** - Converting JSON to JavaScript Object
- **JSON.stringify()** - Converts JavaScript Object to JSON string.

```
{
    “property1“: “value“,
    “property2“: “value“
}

JSON.parse( JSON.stringify( obj ) );
```

https://www.json.org/

ES6

- arrows.
- classes.
- enhanced object literals.
- template strings.
- destructuring.
- default + rest + spread.
- let + const.
- iterators + for..of.

```
// ES5
(function(n) {
    return (n * 2);
})(3);

// ES6
((n => n * 2))(3);
```

http://es6-features.org/

Files

https://goo.gl/odGM7G