JavaScript

GERARD ROMA 2021

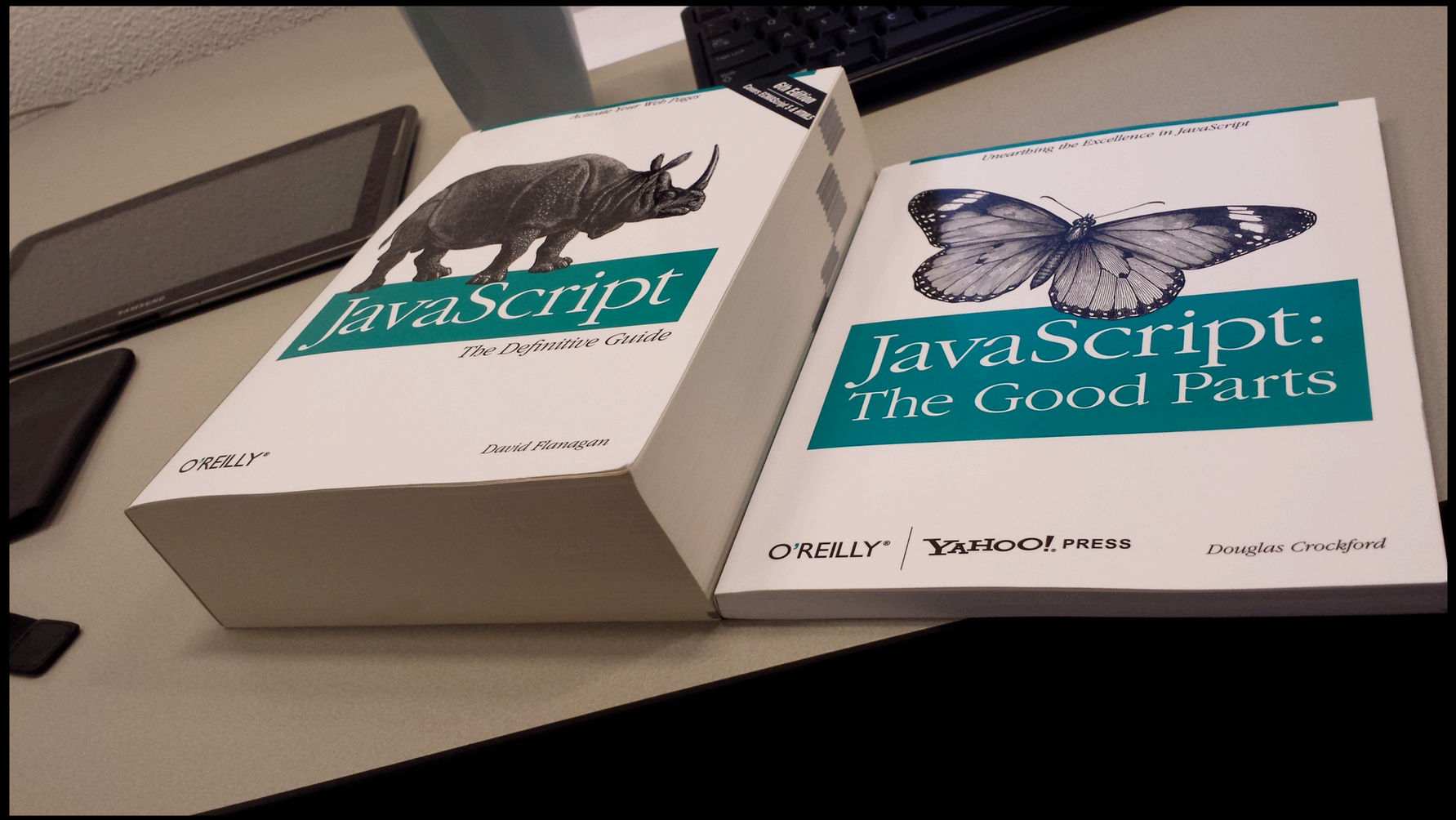
Browser technologies

Behaviour

Text, structure

Appearance

Good parts


JavaScript
The Definitive Guide
David Flanagan
O'REILLY
JavaScript:
The Good Parts
Douglas Crockford
O'REILLY
YAHOO! PRESS

History

- 1995 famously “created in 10 days” by Brendan Eich at Netscape
- 1995-2000 1st Browser War. ECMAScript
- 2005 AJAX, Prototype.js
- 2006 jQuery
- 2009 NodeJS
- 2010 Angular, Backbone
- 2015 EcmaScript 6
- 2018 WebAssembly

HTML head

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>HTML Head Example</title>
    <link rel="stylesheet" href="local/style.css">
    <link rel="stylesheet" href="http://server.com/remote/style.css">
    <script src="local/index.js"></script>
    <script src="http://server.com/remote/index.js"></script>
  </head>
  <body>
  </body>
</html>
```

Trying code

- You can test JavaScript code interactively (AKA REPL - Read Print Eval Loop):
  - Using the console in the browser developer tools.
  - Using Node.js.
- You can write and test longer fragments using a browser and a text editor.

values

Numbers

- Number type includes integers and floats (64 bit)
- Literals can also use scientific notation
- Arithmetic operations

```
let a = 5;
typeof a;
// 'number'
```

```
let b = 2e-10;
```

```
a * 2 + 5;
//15
a * (2 + 5);
//35
```

Strings

- Can use single or double quotes, or backtick

```
let str = "double quotes"
str = 'single quotes’
str = `backtick`
```

- Escape characters

```
console.log("new\nline")
//new
//line
```

- Concatenation

```
"Java" + "Script"
//'JavaScript'
```

Booleans

- Negation

```
let yes = true;
!yes;
//false
```

- Logical AND

```
let no = !yes;
yes && no
//false
```

- Logical OR

```
yes || no
//true
```

- Result of expression

```
typeof (5 > 4)
//'boolean'
```

Empty value

- JavaScript has two different values for “nothing”:
  - Undefined: a variable has been declared but not assigned.
  - Null: explicit nothing. Has to be assigned.

```
let nothing;
//undefined
```

```
let nothing = null;
```

structure

Expressions

- Expressions are bits of code that produce a value.
- Expressions can be nested.

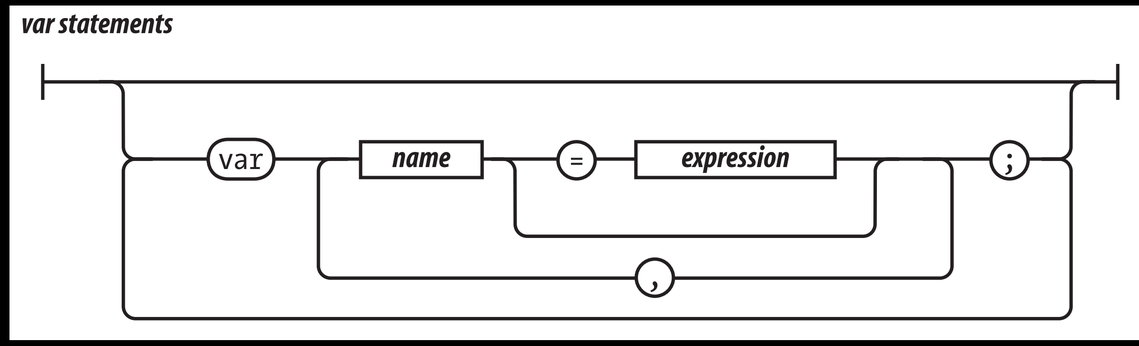
```
5 + 3
// 8
```

```
(5 + 3) +" years"
//'8 years'
```

Statements

- A statement is the smallest unit that represents an action.
- Typically corresponds to a line (but not necessarily).
- In JavaScript statements end with a semicolon: “;” (sometimes with a new line, but it is complicated).
- A program is composed of statements.

Variables



- Variables are names that refer to values in memory.
- A variable declaration is a type of statement.
- Variables can be initialised when declared (usually good idea).
- Multiple keywords:
  - var - old keyword, function scope
  - let - newer, block scope
  - const - constant (needs to be initialised!

```
var thing;
```

```
let five = 5;
```

```
const something;
// error!
```

Function calls

- A function call is an expression that executes a function with a given set of arguments (or none).
- To call a function, use parenthesis () after its name.

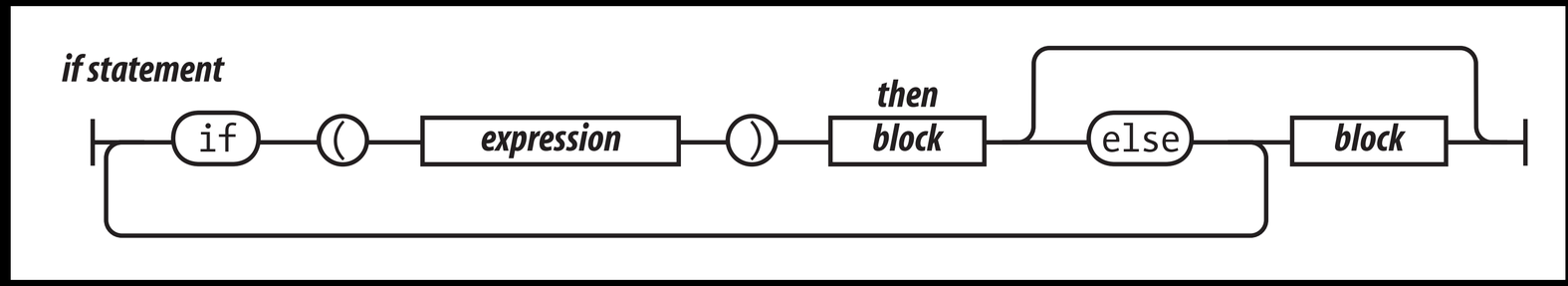
```
alert(“hello!");
//opens dialog box

parseInt(“345.5");
//345
```

Blocks

- A block groups a number of statements into one.
- Delimited by {curly brackets}.
- Usually found in control flow structures (if, loops…)
- But can also be used just to define a **scope**.

```
{
    let x = 5;
    console.log(x);
    //5
}
console.log(x);
//Uncaught
ReferenceError: x is
not defined
```

Branching



- if statement: execute a block only if condition is met.
- Use “else” to execute a block when it is not met.
- Multiple cases can be chained using “else if”.

```
let x = Math.random();
if(x < 0.33){
    x = 0;
}
else if (x < 0.66){
    x = 1;
}
else {
    x = 2;
}
```

Switch

- Switch can be used to substitute a chain of if / else cases.
- Inherited from C: needs “break” for each case, otherwise it continues at the next.

```
switch(x){//we know x is 0, 1 or 2
    case 0:
        console.log("zero");
        break;
    case 1:
        console.log("one");
        break;
    case 2:
        console.log("two");
        break;
    default:
        break;
}
```

While loop

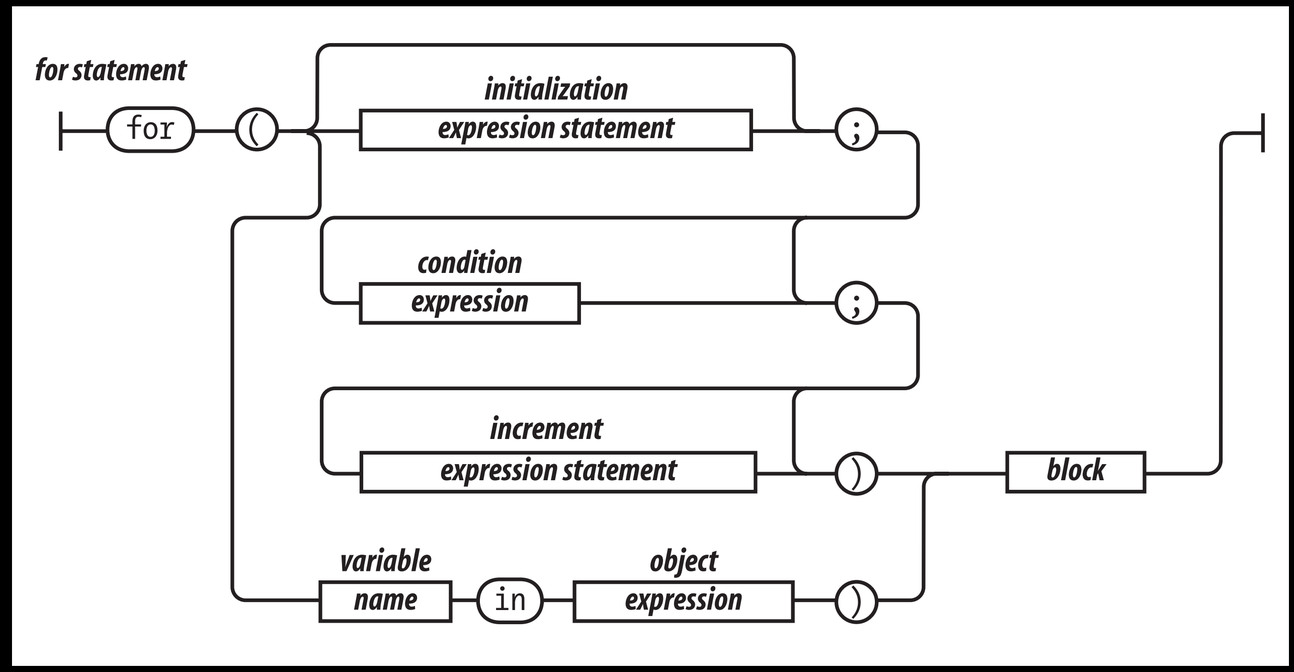
- Loops allow us to automate a sequence of repetitive steps
- The while statement will repeatedly execute a block of code as long as a condition is met.

```
let decades = 0;
while (decades <= 100) {
    console.log(decades);
    decades = decades + 10;
}
// 0, 10, 20...
```

For loop

- Many loops are based on a counter variable.
- For loops allow explicitly managing a counter using three expressions:
  - initialise
  - check
  - update

```
for (let decades = 0;
     decades <= 100;
     decades = decades + 10) {

    console.log(decades);
}
// 0, 10, 20...
```



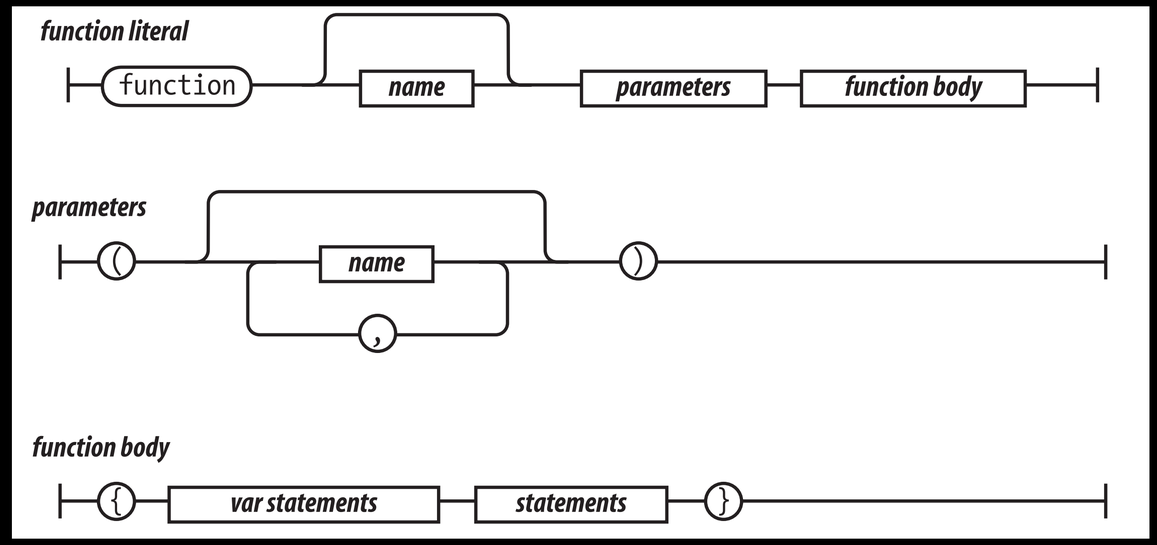
functions

Definition

- Two methods:
  - Function declaration
  - Function expression
- Main elements:
  - name
  - parameters
  - body
  - return

```
function printHi(){
    console.log("hi");
}
```

```
const add5 = function(x){
    return x + 5;
}
```



Parameters

- **Parameters** work like local variables
- Can have default value
- You can pass any number of arguments (may be confusing)

```
function printX(x = "hi"){
    console.log(x);
}

printX()
// hi
printX(4, 5, 6)
//4
```

Scope

- Variables and functions are defined in **Scopes**
- Scopes are nested: a function creates a scope within the global scope. A block inside a function creates a scope within the function.
- Variables and functions defined in a scope are accessible within the scope and its nested scopes.

Functions as values

- A function can be passed as an argument and called with another name.

```
const drawBall = function(){
    circle(0, 0, 20);
}

const draw = function(drawFunc)
{
    drawFunc();
}

draw(drawBall);
```

objects

Properties

- Properties can be seen as variables defined inside an object that are accessible from outside.
- In JavaScript, they can be accessed using either **dot syntax** or **brackets**.

```
let str = "a string";
str.length
//8
str["length"]
//8
```

Methods

- Methods can be seen as **functions** defined inside an object that are accessible from outside.
- Methods have access to the object in which they are defined

```
str.toUpperCase()
//A STRING
```

Objects

- An object is a collection of properties and functions related by some topic.
- Can be defined using {curly brackets}.
- The special keyword **this** allows methods to access their object.

```
let obj = {
    name: "object",
    print:function(){
        console.log("My name is”
                            +this.name);
    }
};

obj.name;
//'object'
obj.print()
//'My name is object'
```

Arrays

- Arrays are actually objects that store collections of things.
- Indices start at 0.
- Elements don’t need to be the same type.

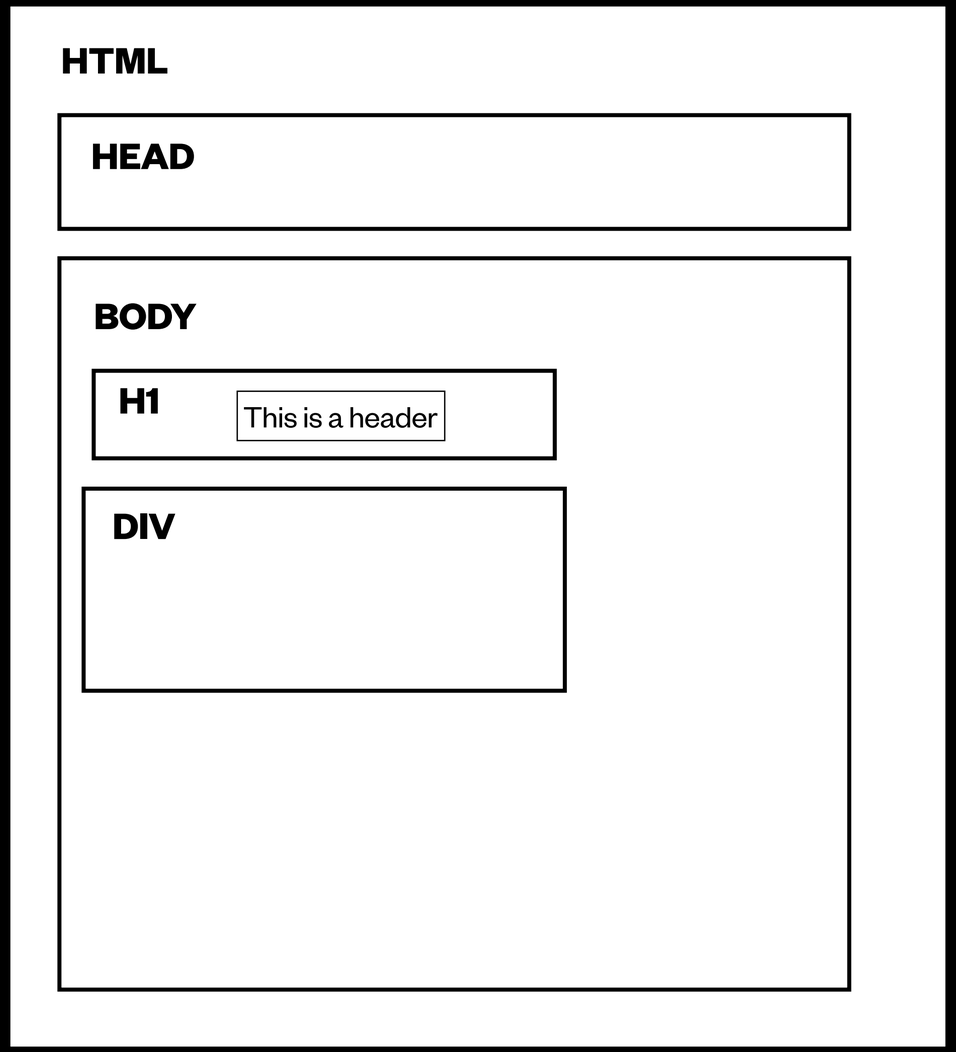
```
let a = [10,3,56,7];
a[0]
//10
```

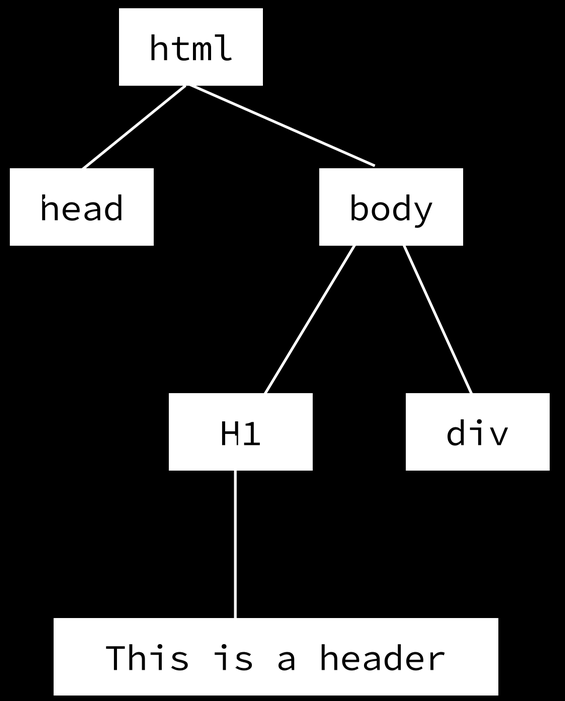
```
let b = ["pizza", 56, [1,2,3]];
b[2]
//  [ 1, 2, 3]
```

DOM

HTML as a tree

- An HTML document can be seen as a tree of elements.
- The Document Object Model (DOM)represents this tree using JavaScript objects.





Querying

- **document** is a global variable representing the HTML document.
- Querying methods:
  - getElementById
  - getElementsByTagName
  - getElementsByClassName
- Advanced querying:jQuery, selectors API
- Traversal: firstChild, nextSibling…

in HTML

```
<div id="wrapper"></div>
```

in JS

```
let el = document.getElementById(“wrapper");
```

Modifying

- The DOM API allows modifying the document using javascript.
- Examples:
  - Dynamically set the text of an element.
  - Modify the CSS style (create animations or interactive effects)

in HTML

```
<a id="link" href = "http://mywebsite.com"> My website</a>
```

in JS

```
var el = document.getElementById("link");
el.innerText = “Blog";
el.href = “http://mywebsite.com/blog”;
el.style.color = "red";
```

Adding

- New HTML elements can also be created from scratch and added to the DOM.

in HTML

```
<div id="wrapper"></div>
```

in JS

```
let el =
document.getElementById("wrapper");
let node = document.createElement("p");
element.appendChild(node);
```

Events

Event handler

- Events are used to deal with unpredictable operations, such as user input or network communication.
- Event handlers are functions that are called by the browser or runtime when an event occurs.
- This asynchronous model avoids having to continuously query the state of a resource (e.g. the mouse, keyboard or a network socket).

Load event

- Script tags are executed as soon as they are found. A linked script may be loaded in parallel as the html keeps loading.
- The window load event is used to start code when all the resources in the opage are loaded, and the DOM related to the HTML structure is available.
- Usually the starting point of a browser-based JavaScript program.

Three equivalent ways:

```
<body onload=“myFunction()"></body>
```

```
window.onload = function(e){
    console.log("loaded");
}
```

```
window.addEventListener(“load”, function(e){
    console.log("loaded");
});
```

Event object

- Event handlers get passed an event object.
- We can query this object to find information about the event.
- For example, if a key is pressed, we can get the key identifier.

```
    document.body.onkeydown = function(e){
        console.log(e.key);

    };
    //prints A, w, Shift, Escape ...
```

More resources

- M. Haverbeke, “Eloquent JavaScript”
https://eloquentjavascript.net/
- D. Crockford, “JavaScript: The Good Parts”
https://www.crockford.com/javascript/
- Mozilla JavaScript reference
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference