LINKÖPING UNIVERSITY

729G87 Interaction Programming

Lecture 2 – Intro Web Technologies (JavaScript)

Philipp Hock, PhD
philipp.hock@liu.se

Basic HTML Eco System

Content

text, images
audio, video,…

HTML

Design

color, position,
size,…

CSS

Logic

events,
calculations,…

JavaScript

Learning goals

- You will **not** learn
  - How to code (we assume you already know this to an extend)
  - How to handle basic stuff like loops, if..then..else, functions, etc.
  - Everything about JavaScript
  - Everything you need to complete the assignments

- You will learn
  - Everything to achieve completing the assignments
  - Important aspects about JavaScript
  - Things not necessary to complete the assignments
  - Some good practices and pitfalls of JS

What is JavaScript (JS)

- JavaScript is a programming language that is used in many different applications.
- JavaScript is not Java, they are completely separate things.
- Formally specified as ECMAScript (ISO ISO/IEC 22275:2018)
- First version 1997, current version 2022 (ECMAScript 13)
- http://www.ecma-international.org/publications/standards/Ecma-262.htm
- Programming language features
  - interpreted - like Python
  - dynamically typed - like Python
  - prototype-based programming (similarities with OOP)

ECMA Script

ECMAScript (/ˈɛkməskrɪpt/; **ES**)[1] is a standard for scripting languages, including JavaScript, JScript, and ActionScript. It is also best known as a JavaScript standard intended to ensure the interoperability of web pages across different web browsers.[2] It is standardized by Ecma International in the document ECMA-262. ECMAScript is commonly used for client-side scripting on the World Wide Web, and it is increasingly being used to write server-side applications and services using Node.js and other runtime environments.

https://en.wikipedia.org/wiki/ECMAScript

Some buzzwords

- High level
- Single threaded
- Garbage collected
- Just in time compiled
- Multi-paradigm
- Dynamically typed

https://www.youtube.com/watch?v=DHjqpvDnNGE

Execution environment

What can JavaScript?

- **Manipulate the DOM, i.e.**
  - **Change DOM structure**
  - **Change the ID of DOM nodes**
  - **Change the classes of DOM nodes**
  - **Change contents of DOM nodes**
- **Manipulate the CSS**
  - **Add inline CSS to the DOM**
  - **Change the contents of the stylesheet**
- **Capture web browser events**
  - **changes to the web browser window (size, active/not active, URL etc)**
  - **changes originating from input devices: mouse movement, keyboard key presses, single and multitouch events**

DOs and DON'Ts

- Use JavaScript to
  - 1. Listen for specific web browser events on specific elements in the DOM
  - 2. Change/Create elements in the DOM
  - Send & retrieve information from other resources
- Do not use JavaScript to
  - Validate user input (at least not only JS)
  - Verify credentials
  - Security

Type system

- JavaScript is a loosely typed language with dynamic type checking.
- Languages with static vs dynamic type checking
- Strongly vs weakly/loosely typed languages

Strong vs loose typing

- In general: weak/loose typing
  - less strict rules for datatypes,
  - e.g implicit type conversions
- Strong typing, e.g. Python
- Example: "2" - 1 → ERROR

- Weak typing, e.g. JavaScript,
- Example in JavaScript: "2" - 1 → "1"
- Convenience vs safety

Static vs dynamic type checking

- Static type checking: Type checking is done before runtime.
  - **Datatype** of the value a variable refers to **cannot** be **changed** during runtime
  - Examples: C, Java

  let i = 1
  i = “hello”
  -> error

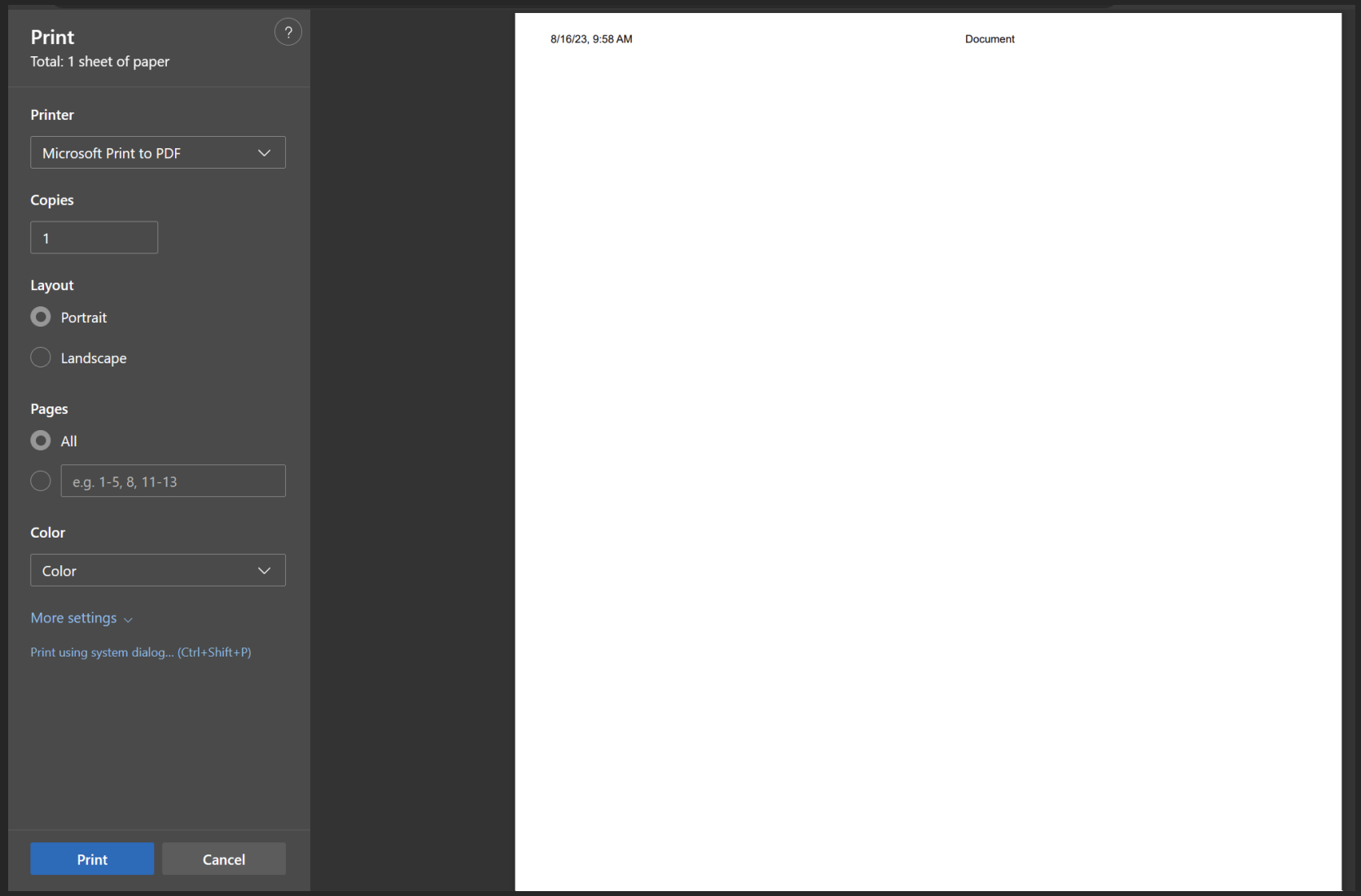
- Dynamic type checking: Type checking is done during runtime.
  - **Datatype** of the value a variable refers to **can** be **changed** during runtime
  - Examples: Python, JavaScript

  let i = 1
  i = “hello”

Hello World: console

```
//main.js
print("hello world")
```

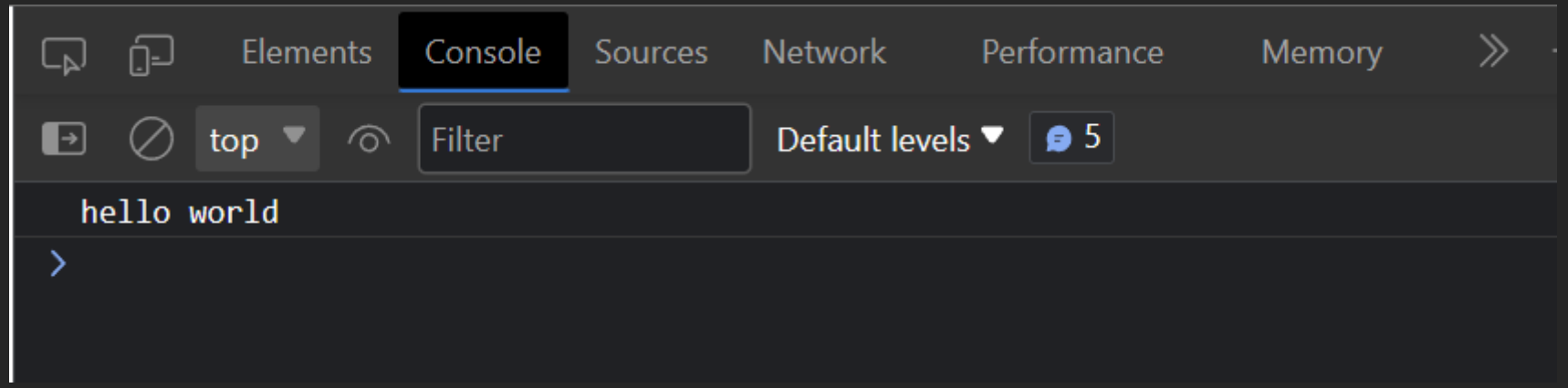
Hello World: console



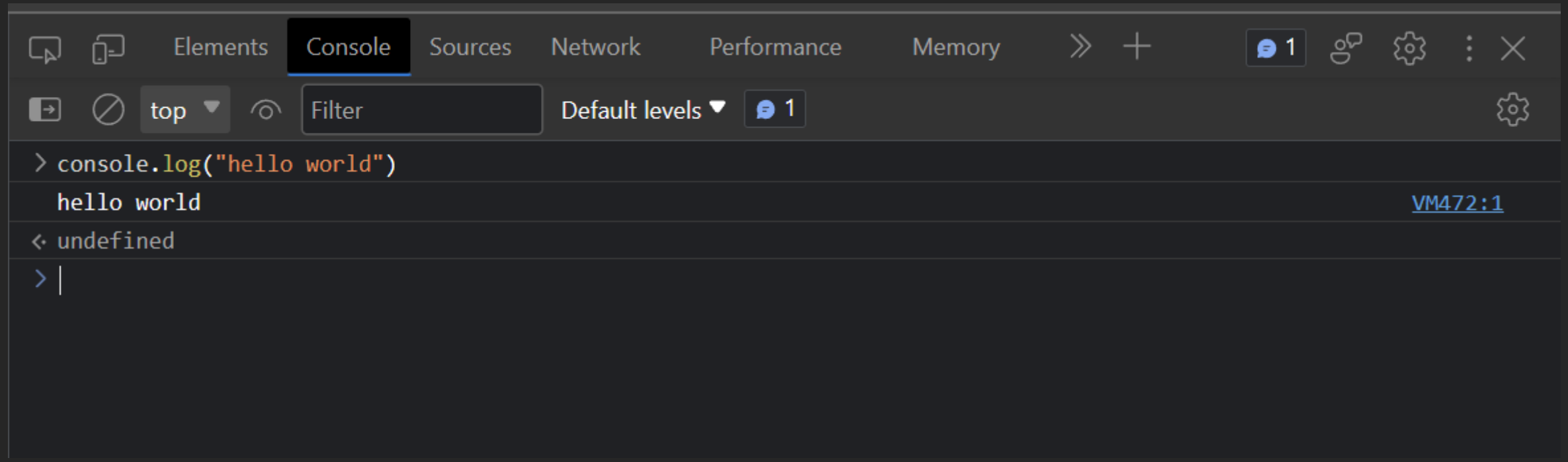
Hello World: console

```
//main.js
console.log("hello world")
```

Developer console

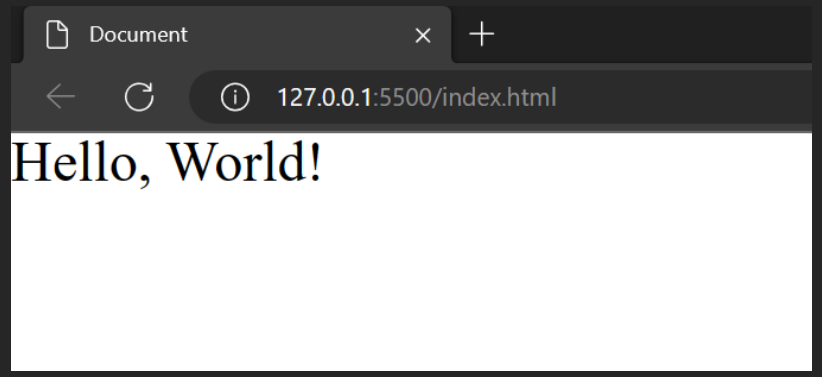


Developer console



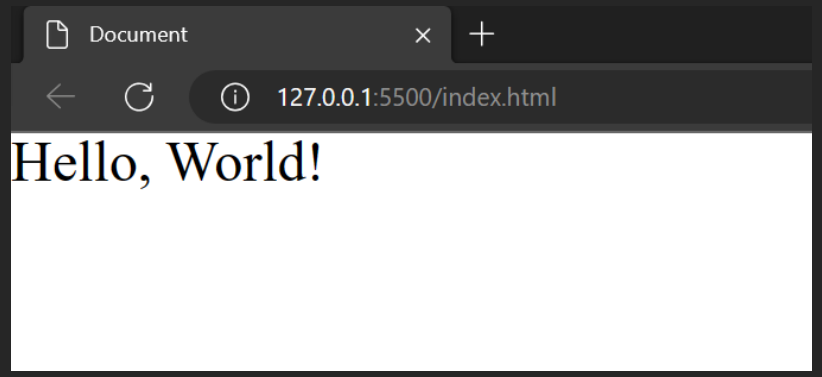
Hello World: in document

```
//main.js
let helloText = document.createElement("p");
helloText.textContent = "Hello, World!";
document.body.appendChild(helloText);
```



Hello World: in document

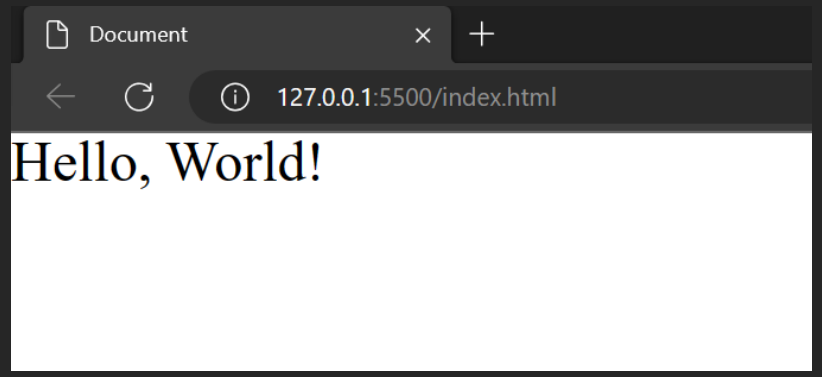
```
<body>
    <p id="displayHelloWorldHere"></p>
</body>
```




```
//main.js
let helloText = document.querySelector("#displayHelloWorldHere")
helloText.textContent = "Hello, World!";
```

Hello World: in document (the wrong way)

```
<body>
    <p id="displayHelloWorldHere"></p>
</body>
```




```
//main.js
var helloText = document.getElementById("displayHelloWorldHere")
helloText.innerHTML = "Hello, World!";
```

Hello World: in document (the wrong way)

```
//main.js
var helloText = document.getElementById("displayHelloWorldHere")
helloText.innerHTML = "Hello, World!";
```

`getElementById`

Old school way to access elements, querySelector supports all CSS selectors

`innerHTML`

Can insert html-tags. Possible security issue!

```
helloText.innerHTML = "<s>escape</s>";
```

Import / Execute JS in HTML

```
<!DOCTYPE html>
<html>
<head>
    <title>JavaScript Execution in HTML</title>
</head>
<body>

<!-- Inline JavaScript -->
<script>
    console.log("Inline JavaScript executed");
</script>

<!-- External JavaScript file -->
<script src="external-script.js"></script>
```

Multiple source files

```
//greet.js
function sayHello() {
  console.log('Hello from classic JavaScript import!');
}
```

```
// main.js
sayHello();
```

Multiple source files

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Classic JavaScript Import Example</title>
</head>
<body>

    <h1>Hello, Classic JavaScript Import!</h1>
    <script src="greet.js"></script>
    <script src="main.js"></script>

</body>
</html>
```

Modules

Requires website accessed from a webserver -> live server plugin!

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>JavaScript Modules Example</title>
</head>
<body>
    <h1>Hello, JavaScript Modules!</h1>
    <script type="module" src="main.js"></script>
</body>
</html>
```

Modules

```
// main.js
import { sayHello } from './module.js';

sayHello();
```

```
// module.js
export function sayHello() {
  console.log('Hello from the module!');
}
```

Extending our boilerplate

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <link rel="stylesheet" href="css/reset.css">
    <link rel="stylesheet" href="css/layout.css">
    <link rel="stylesheet" href="css/style.css">

    <script type="module" src="js/main.js">
    </script>
</head>

<body>
</body>

</html>
```

Find relatives

- myElement.firstElementChild;
- myElement.lastElementChild;
- myElement.parentElement;
- myElement.nextElementSibling;
- myElement.previousElementSibling;
- myElement.querySelector();
- myElement.closest('.header')

let and var

```
var b = 2;
var b = 3;

let a = 1;
let a = 4;
```

```
Cannot redeclare block-scoped variable 'a'. ts(2451)
let a: number
View Problem (Alt+F8)    No quick fixes available
```

let and var

```
var x = 1;

if (x === 1) {
    var x = 2;

    console.log(x);
    // output: 2
}

console.log(x);
// output: 2
```

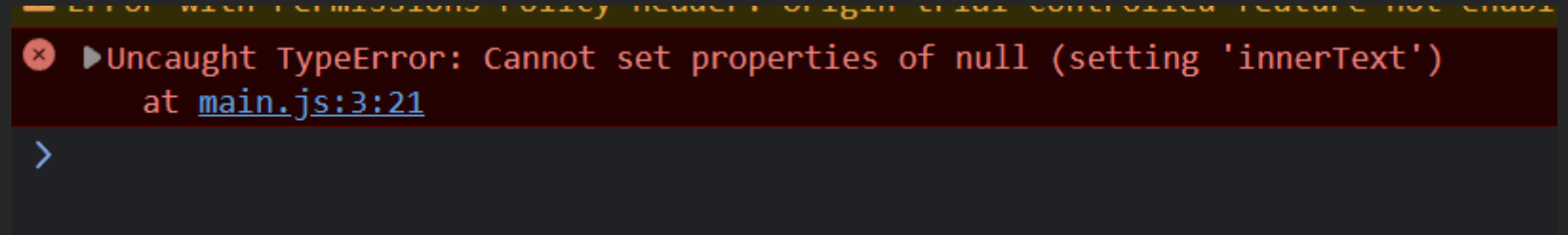
let and var

```
let x = 1;

if (x === 1) {
    let x = 2;

    console.log(x);
    // output: 2
}

console.log(x);
// output: 1
```

do not use var at all!

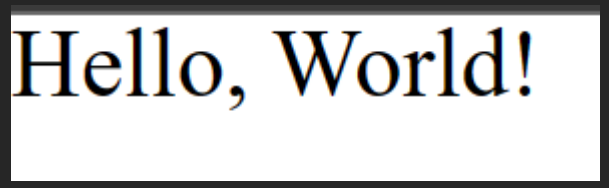
Delay execution

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script  src="js/main.js" >
    </script>
</head>
<body>
    <div id ="displayHelloWorldHere"></div>
</body>
</html>
```



Delay execution

```
<script  src="js/main.js" defer>
<script>

<script  src="js/main.js" type="module">
</script>
```



Delay execution

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <link rel="stylesheet" href="css/reset.css">
    <link rel="stylesheet" href="css/layout.css">
    <link rel="stylesheet" href="css/style.css">

    <script type="module" src="js/main.js">
    </script>
</head>

<body>
    <p id="displayHelloWorldHere"></p>
</body>

</html>
```

Delay execution: events

```
//main.js

document.addEventListener("DOMContentLoaded", (event)=>{
  let helloText = document.querySelector("#displayHelloWorldHere")
  helloText.textContent = "Hello world";

})
```

Events

```
<body>
    <button>Press Me</button>
</body>
```

```
document.querySelector("button").addEventListener("click", (event)=>{
  alert("hello")
});
```

Events

```html
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <link rel="stylesheet" href="css/reset.css">
    <link rel="stylesheet" href="css/layout.css">
    <link rel="stylesheet" href="css/style.css">

    <script  src="js/main.js" >
    </script>
</head>

<body>
    <button onclick="hello();">Press Me</button>
</body>

</html>
```

```js
//main.js
function hello(){
  alert("hello")
};
```

Even this works (don’t do this)

```
<button onclick="console.log('hello');">Press Me</button>
```

Event handling

```
// An event handler function only receives an Event object as an argument.
// The Event object contains information about the event (more on this later)
function handleEvent(event) {
  // code for handling event
}
element.addEventListener('click', handleEvent)
```

Event handling

```
// If we want to pass specific arguments to our event handler function,
// we can wrap it using an anonymous function
function handleOtherEvent(event, param1, param2 ...) {
  // code for handling event
}

element.addEventListener('click', (event) => {
  handleOtherEvent(event, "argument1", "argument2", ...)
});
```

Event bubbling

https://www.w3.org/TR/uievents/images/eventflow.svg

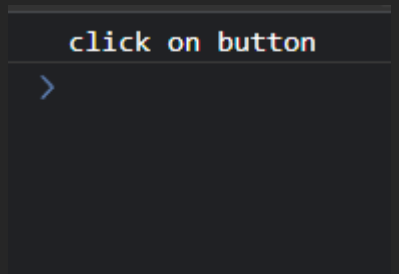
```
<body>
    <button>hello</button>

</body>
<script>
    document.querySelector("html").addEventListener("click", (e)=>{
        console.log("click on html");
    });
    document.querySelector("button").addEventListener("click", (e)=>{
        console.log("click on button");

    });
</script>

</html>
```

```
click on button
click on html
>
```

Event bubbling

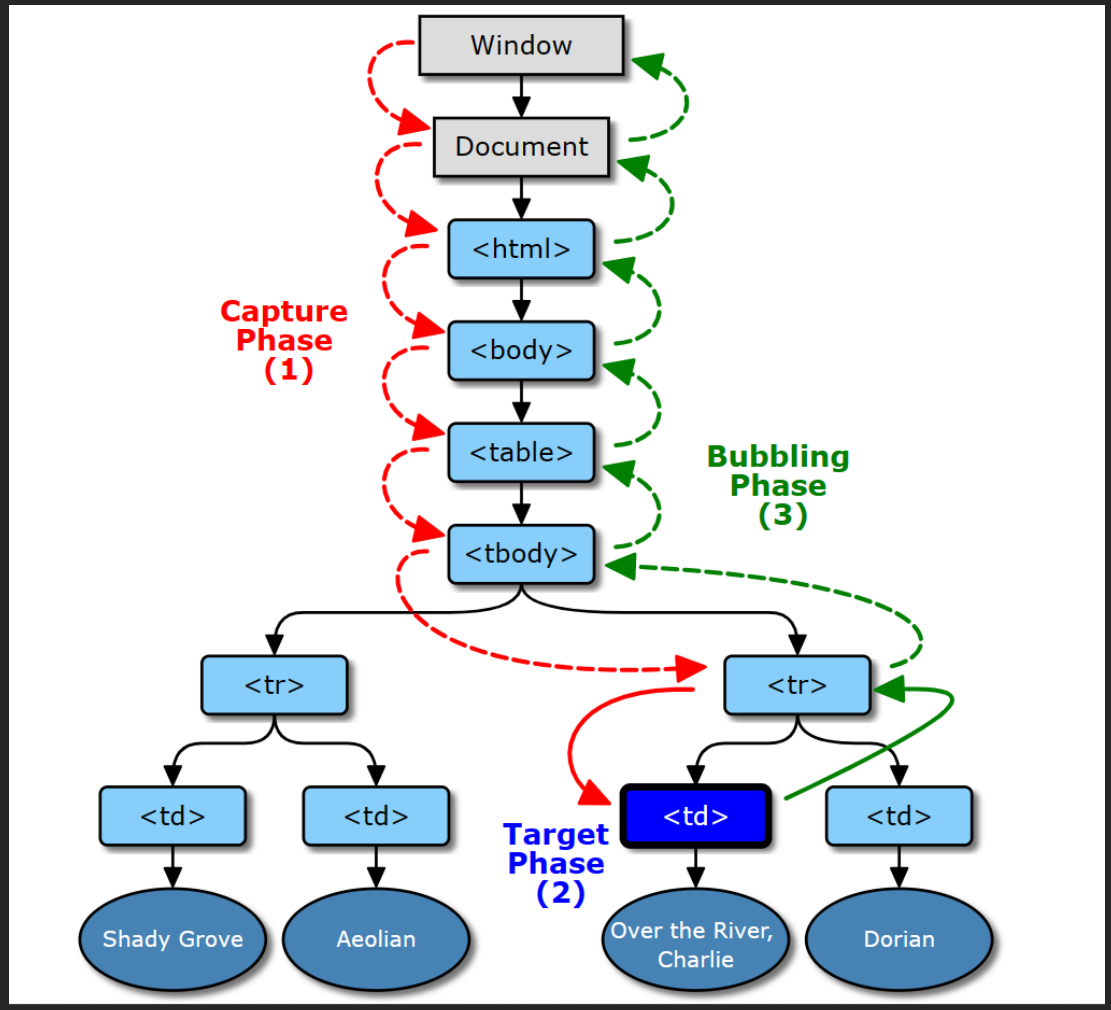
https://www.w3.org/TR/uievents/images/eventflow.svg

```
<body>
    <button>hello</button>

</body>
<script>
    document.querySelector("html").addEventListener("click", (e)=>{
        console.log("click on html");
    });
    document.querySelector("button").addEventListener("click", (e)=>{
        console.log("click on button");
        e.stopPropagation();

    });
</script>

</html>
```



Event bubbling

https://www.w3.org/TR/uievents/images/eventflow.svg



https://codepen.io/Philipp-Hock/pen/jOXOBWV

Data-Attribute

```
<div class="box"
  data-info="This is a data attribute example.">

  Hover over this box to see the data attribute value.

</div>
```

```
 const box = document.querySelector('.box');
 box.addEventListener('click', function(e) {
                  console.log(e.target.dataset['infoElem'])
                  console.log(e.target.dataset.infoElem)
  });
```

Data-Attribute

https://codepen.io/Philipp-Hock/pen/OJrJppy

Data-Attribute

```
<div class="box"
  data-info="This is a data attribute example.">

  Hover over this box to see the data attribute value.

</div>
```

```
 const box = document.querySelector('.box');
 box.addEventListener('mouseover', function() {
   const dataInfo = this.getAttribute('data-info');
   alert(dataInfo);
 });
```

Classes

- There are different kind of classes
  - Classes in an Object Oriented way

```
class Person {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }
}
```

  - HTML classes for css/js

```
<div class="wrapper">

</div>
```

- They have nothing in common

Classlist

- element.classList.add("classname")
- element.classList.remove("classname")
- element.classList.replace("oldclass", "newclass")
- element.classList.toggle("classname")

Inline Style

- Inline styles are styles written using the HTML elements style attribute, e.g. Hello!

- Read or set inline CSS property
- element.style.PROPERTY = String
- PROPERTY should be written in camelCase, e.g.
  element.style.backgroundColor = 'lightblue‘
  element.style.display = 'none'
- Remove inline CSS property

element.style.removeProperty(property-name : String)

property-name should be written hyphenated, e.g.

element.style.removeProperty('background-color")

Objects

```
const myObj = {
  value: 10,
  display: function(){
    return this.value;
  }
}

console.log(myObj.display())
```

Classes

```
class MyObj {
  constructor(value) {
    this.value = value;
  }

  display() {
    return this.value;
  }
}

const myInstance = new MyObj(10);
console.log(myInstance.display());
```

Classes

convert into a class:

```
const myObj = {
 value: 10,
 display: function(){
  return this.value;
 }
}

console.log(myObj.display())
```

Sure! Here's the equivalent code using a class:

javascript Copy code

```javascript
class MyObj {
  constructor(value) {
    this.value = value;
  }

  display() {
    return this.value;
  }
}

const myInstance = new MyObj(10);
console.log(myInstance.display());
```

Arrow functions vs function()

https://www.freecodecamp.org/news/the-difference-between-arrow-functions-and-normal-functions/

```
function classicFunction(){

}

let arrowFunction = ()=>{

}
```

Arrow functions vs function()

https://www.freecodecamp.org/news/the-difference-between-arrow-functions-and-normal-functions/

```
const classicFunctionObj = {
  value: 10,
  display: function(){console.log(this.value)}
}

classicFunctionObj.display();
// displays 10

 const arrowFuncObj = {
   value: 10,
   display: ()=>{console.log(this.value)}
 }

 arrowFuncObj.display();
 // displays undefined
```

with arrow functions there are no binding of this . In regular functions the this keyword represented the object that called the function

String literals

```
const count = 5;
const square = (v)=>v*v;

console.log(`the square of ${count} is ${square(count)}`);

// output: the square of 5 is 25
```

semicolon or not?

Stay consistent!

```
const count = 5;
const square = (v)=>v*v;

console.log(`the square of ${count} is ${square(count)}`);

// output: the square of 5 is 25
```

```
const count = 5
const square = (v)=>v*v

console.log(`the square of ${count} is ${square(count)}`)

// output: the square of 5 is 25
```

Creating Elements

- Create new elements using

  document.createElement(elementName : String)

- Add it to the DOM using e.g.
  - parentElement.append(newElement)

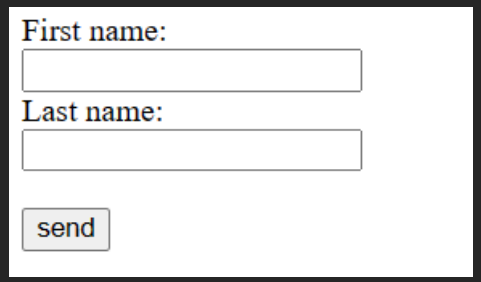
  add newElement as last child of parentElement
  - siblingElement.before(newElement)

  insert newElement before siblingElement

- Remove an element

  element.remove()

Forms




```
<body>
    <form>
        <label for="fname">First name:</label><br>
        <input type="text" id="fname" name="fname"><br>
        <label for="lname">Last name:</label><br>
        <input type="text" id="lname" name="lname"><br><br>
        <input type="submit" value="send">
      </form>
</body>
```

Forms

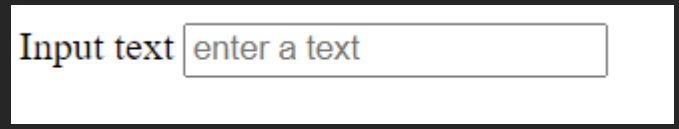
- Often used to enter data that is to be submitted to a server.
- <form> element (parent) contains the form e.g. labels, text fields, checkboxes (children)
- form children have a name attribute that are used to identify them when the form is submitted
- form children can also have an id which is used e.g. to identify them within the DOM
- name attribute and id attribute can be the same, but do not have to
- be the same

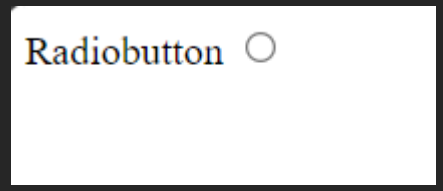
HTML Form elements

- <input>: Basic input element for various data types (text, password, number, etc.).
- <textarea>: Multi-line text input area.
- <select>: Dropdown menu for selecting options.
- <button>: Clickable button for triggering actions.
- <label>: Text label associated with form controls.
- ...

HTML Input Types

- type="radio": Select one option from a group.
- type="checkbox": Select multiple options from a group.
- type="file": Upload files from user's device.
- type="submit": Submit the form data to a server.
- type="reset": Reset form elements to default values.
- type="hidden": Store data on the form without displaying it.
- type="date": Input for date selection.
- type="email": Input for email addresses.
- type="url": Input for website URLs.
- type="color": Input for selecting colors.
- type="range": Slider input for selecting numeric values within a range.
- type="search": Input for search queries.
- type="password“: Conceals the characters
- …





Interaction with JS

- Each HTML element has a corresponding JavaScript WebAPI interface (a class for each element type)
- E.g. <select> elements correspond to objects with the HTMLSelectElement interface.
  - https://developer.mozilla.org/en-US/docs/Web/API/HTMLSelectElement
- There we can e.g. find out that events that HTMLSelectElement elements can fire are change and input.
- We can also find out that HTMLSelectElement object have the property value that we can read to get the selected option

JS Form Example

```
<form id="fruitForm">
    <select id="fruitSelect">
        <option value="apple">Apple</option>
        <option value="banana">Banana</option>
        <option value="orange">Orange</option>
        <option value="grape">Grape</option>
    </select>
</form>

<p id="message"></p>
```

JS Form Example

https://codepen.io/Philipp-Hock/pen/oNJNGYj

```
<script>
    const fruitSelect = document.getElementById('fruitSelect');
    const messageElement = document.getElementById('message');

    fruitSelect.addEventListener('change', function () {
        const selectedOption = fruitSelect.options[fruitSelect.selectedIndex];
        const selectedFruit = selectedOption.value;
        let message = '';
        messageElement.textContent = `You selected ${selectedOption.value}`;
    });
</script>
```

https://www.youtube.com/watch?v=aXOChLn5ZdQ

The JavaScript Survival Guide

https://www.youtube.com/watch?v=9emXNzqCKyg

https://dev.to/novu/10-github-repositories-to-achieve-javascript-mastery-50hk