{POWER.CODERS}

HTML and the DOM

Agenda

Today we will look into

- JS Recap Quiz
- HTML and the Document Object Model

What is a variable?

What is a variable?

In variables you store data.

How to declare a variable?

How to declare a variable?

`var` defines a variable globally or locally to an entire function.

```
var x;
```

How to declare a variable?

`var` defines a variable globally or locally to an entire function.

```
var x;
```

`let` defines a variable limited in scope to the block, statement or expression in which it is used.

```
let x;
```

How to declare a variable?

`var` defines a variable globally or locally to an entire function.

```
var x;
```

`let` defines a variable limited in scope to the block, statement or expression in which it is used.

```
let x;
```

`const` defines an **immutable** variable in the same scope than **let**.

```
const x;
```

DESTRCUTURE THIS

Which variable keywords would be better than `var`?

```
var person = {
  firstName : "John",
  lastName  : "Doe",
  age       : 50,
  eyeColor  : "blue"
};

var firstName = person.firstName;
var lastName = person.lastName;
var age = person.age;
var eyeColor = person.eyeColor;
```

Destructured

```
const person = {
  firstName : "John",
  lastName  : "Doe",
  age       : 50,
  eyeColor  : "blue"
};

let {lastName} = person;
const {firstName, age, eyeColor} = person;
```

What is scope?

What is scope?

In JavaScript each function, statement and expression create a new scope. **Scope** determines the accessibility (visibility) of the declared variables.

What is the value of a in the alert?

```
function q1() {
  var a = 5;
  if(a > 1) {
      a = 3;
  }
  alert(a);
}
```

What is the value of b in the alert?

```
var b = 0;
function q2() {
  b = 5;
}

function q22() {
  alert(b);
}
```

What is the value of a in the alert?

```
function q3() {
  window.a = "hello";
}
function q32() {
  alert(a); //"hello"
}
```

What is the value of b in the alert?

```
var b = 1;
function q4() {
    var b = "test";
    alert(b);
}
```

What is the value of c in the alert?

```
var c = 2;
if (true) {
    var c = 5;
    alert(c);
}
alert(c);
```

What is the value of c in the alert?

```
let c = 2;
if (true) {
    let c = 5;
    alert(c);
}
alert(c);
```

Data types and structures

Data types and structures

- Number

```
let x = 10;
```

Data types and structures

- Number

```
let x = 10;
```

- Boolean

```
let x = True;
```

DATA TYPES AND STRUCTURES

- Number
```
let x = 10;
```
- Boolean
```
let x = True;
```
- String
```
let x = "text";
```

DATA TYPES AND STRUCTURES

- Number

```
let x = 10;
```

- Boolean

```
let x = True;
```

- String

```
let x = "text";
```

- Array

```
let x = ["item1", "item2", "item3"];
```

Data types and structures

- Number

```
let x = 10;
```

- Boolean

```
let x = True;
```

- String

```
let x = "text";
```

- Array

```
let x = ["item1", "item2", "item3"];
```

- Object

```
let x = {name: "value",age: 24, hello:
function() {return "Hello " + this.name;}};
```

What is a template string?

What is a template string?

Template strings or **template literals** are an alternative way to concating strings with variables.

What is a template string?

Template strings or **template literals** are an alternative way to concating strings with variables.

```
let name = 'David';
let msg = `Welcome ${name}!`;
```

What is a template string?

Template strings or **template literals** are an alternative way to concating strings with variables.

```
let name = 'David';
let msg = `Welcome ${name}!`;
```

Notice that **${expression}** is enclosed by backticks `` ` `` instead of quotes.

Question about switch

```
function moveCommand(direction) {
    var whatHappens;
    switch (direction) {
        case "forward":
            break;
            whatHappens = "you encounter a monster";
        case "back":
            whatHappens = "you arrived home";
            break;
            break;
        case "right":
            return whatHappens = "you found a river";
            break;
```

- Return value of `moveCommand("forward")`
- Return value of `moveCommand("back")`
- Return value of `moveCommand("right")`

QUESTION ABOUT SWITCH

```
function moveCommand(direction) {
    var whatHappens;
    switch (direction) {
        case "forward":
            break;
            whatHappens = "you encounter a monster";
        case "back":
            whatHappens = "you arrived home";
            break;
            break;
        case "right":
            return whatHappens = "you found a river";
            break;
```

- Return value of `moveCommand("forward")` **undefined**
- Return value of `moveCommand("back")`
- Return value of `moveCommand("right")`

Question about switch

```
function moveCommand(direction) {
    var whatHappens;
    switch (direction) {
        case "forward":
            break;
            whatHappens = "you encounter a monster";
        case "back":
            whatHappens = "you arrived home";
            break;
            break;
        case "right":
            return whatHappens = "you found a river";
            break;
```

- Return value of `moveCommand("forward")` **undefined**
- Return value of `moveCommand("back")` **you arrived home**
- Return value of `moveCommand("right")`

QUESTION ABOUT SWITCH

```
function moveCommand(direction) {
    var whatHappens;
    switch (direction) {
        case "forward":
            break;
            whatHappens = "you encounter a monster";
        case "back":
            whatHappens = "you arrived home";
            break;
            break;
        case "right":
            return whatHappens = "you found a river";
            break;
```

- Return value of `moveCommand("forward")` **undefined**
- Return value of `moveCommand("back")` **you arrived home**
- Return value of `moveCommand("right")` **you found a river**

HTML & the Document Object Model

DOM

Document Object Model

Anything found in an HTML or XML document can be accessed, changed, deleted, or added by a programmer using the DOM.

The DOM of an HTML document can be representeded as a nested set of boxes, called DOM Tree.

DOM Tree

The DOM represents a document as a tree structure. HTML elements become **nodes** in that tree.

All nodes in the tree have some kind of relation to each other:

DOM Tree

The DOM represents a document as a tree structure. HTML elements become **nodes** in that tree.

All nodes in the tree have some kind of relation to each other:

- parent
- child
- sibling

`document`

The `document` object is globally available in your browser.

It allows you to access and manipulate the DOM of the current web page:

1. Find the DOM node you want to change
2. Store this DOM node as a variable
3. Manipulate the DOM node
    - Change its attributes
    - Modify its styles
    - Give it new inner HTML
    - Append new nodes to it

Finding a node

The `document` object is the **root** of the DOM.

```
document.body.textContent = "New text";
```

Finding a node

The `document` object is the **root** of the DOM.

```
document.body.textContent = "New text";
```

body is a node inside the DOM and can be accessed using the `document` object.

Finding a node

The `document` object is the **root** of the DOM.

```
document.body.textContent = "New text";
```

body is a node inside the DOM and can be accessed using the `document` object.

All HTML elements are **objects** with **properties** and **methods.**

Selecting elements

Selecting elements

- `document.getElementById(id)`

Selecting elements

- `document.getElementById(id)`
- `document.getElementsByClassName(classname)`

Selecting elements

- `document.getElementById(id)`
- `document.getElementsByClassName(classname)`
- `document.getElementsByTagName(tagname)`

Selecting elements

- `document.getElementById(id)`
- `document.getElementsByClassName(classname)`
- `document.getElementsByTagName(tagname)`

`document.getElementsByClassName` and `document.getElementsByTagName` return all of the elements as an **array**.

Selecting elements

- `document.getElementById(id)`
- `document.getElementsByClassName(classname)`
- `document.getElementsByTagName(tagname)`

`document.getElementsByClassName` and `document.getElementsByTagName` return all of the elements as an **array**.

Thus, you can call single elements by their index, e.g. `[0]` or loop through them.

Storing it as a variable

```
let elm = document.getElementById("demo");
```

Storing it as a variable

```
let elm = document.getElementById("demo");
```

Each element in the DOM has a set of properties and methods we can use to determine its relationships in the DOM.

Working with DOM

Working with DOM

- `elm.childNodes` returns an array of an element's children

Working with DOM

- `elm.childNodes` returns an array of an element's children
- `elm.firstChild` returns the first child node of an element

Working with DOM

- `elm.childNodes` returns an array of an element's children
- `elm.firstChild` returns the first child node of an element
- `elm.lastChild` returns the last child node of an element

Working with DOM

- `elm.childNodes` returns an array of an element's children
- `elm.firstChild` returns the first child node of an element
- `elm.lastChild` returns the last child node of an element
- `elm.hasChildNodes` returns true if there are children

Working with DOM

- `elm.childNodes` returns an array of an element's children
- `elm.firstChild` returns the first child node of an element
- `elm.lastChild` returns the last child node of an element
- `elm.hasChildNodes` returns true if there are children
- `elm.nextSibling` returns the next node at the same level

Working with DOM

- `elm.childNodes` returns an array of an element's children
- `elm.firstChild` returns the first child node of an element
- `elm.lastChild` returns the last child node of an element
- `elm.hasChildNodes` returns true if there are children
- `elm.nextSibling` returns the next node at the same level
- `elm.previousSibling` returns the previous node

Working with DOM

- `elm.childNodes` returns an array of an element's children
- `elm.firstChild` returns the first child node of an element
- `elm.lastChild` returns the last child node of an element
- `elm.hasChildNodes` returns true if there are children
- `elm.nextSibling` returns the next node at the same level
- `elm.previousSibling` returns the previous node
- `elm.parentNode` returns the parent node of an element

Example

```
let elm = document.getElementById("demo");
let arr = elm.childNodes;
arr.forEach(function(el){
  el.textContent = "new text";
});
```

`querySelector`

`querySelector`

- `document.querySelector()` returns the **first element** that matches the specified CSS selector(s)

`querySelector`

- `document.querySelector()` returns the **first element** that matches the specified CSS selector(s)
- `document.querySelectorAll()` returns **all elements** that match the specified CSS selector(s)

`querySelector`

- `document.querySelector()` returns the **first element** that matches the specified CSS selector(s)
- `document.querySelectorAll()` returns **all elements** that match the specified CSS selector(s)

```
let arr = document.querySelectorAll("#demo > *");
arr.forEach(function(el){
  el.textContent = "new text";
});
```

Changing attributes

```
<img src="orange.png" id="myimg">
<script>
  let el = document.querySelector("#myimg");
  el.src="apple.png";
  el.className="landscape";
</script>
```

Changing attributes

```
<img src="orange.png" id="myimg">
<script>
  let el = document.querySelector("#myimg");
  el.src="apple.png";
  el.className="landscape";
</script>
```

Practicallly all attributes of an element can be changed using JavaScript, e.g. `src`, `href` or `value`.

Changing style

```
<p id="demo">Some text.</p>
<script>
  let el = document.querySelector("#demo");
  el.style.color="#6600FF";
  el.style.width="100px";
  el.style.backgroundColor="red";
</script>
```

Changing style

```
<p id="demo">Some text.</p>
<script>
  let el = document.querySelector("#demo");
  el.style.color="#6600FF";
  el.style.width="100px";
  el.style.backgroundColor="red";
</script>
```

All CSS properties can be set and modified using JavaScript. Just remember that you cannot use dashes (-) in the propertey names, e.g. `backgroundColor`, `textDecoration` or `paddingTop`.

Creating new elements

```
<p id="demo">Some <strong>text</strong>.</p>
<script>
  let span = document.createElement("span");
  let node = document.createTextNode("Some new text");
  let parent = document.querySelector("#demo");
  span.appendChild(node);
  parent.appendChild(span);
</script>
```

Creating new elements

```
<p id="demo">Some <strong>text</strong>.</p>
<script>
  let span = document.createElement("span");
  let node = document.createTextNode("Some new text");
  let parent = document.querySelector("#demo");
  span.appendChild(node);
  parent.appendChild(span);
</script>
```

This creates a new span-tag, appending content to it, and afterwards appending the new element to the existing paragraph.

INSERT METHODS

- `node.appendChild(new_node)` adds a node at the end of the list of children

INSERT METHODS

- `node.appendChild(new_node)` adds a node at the end of the list of children
- `parent.insertBefore(new_node,node)` adds a node right before a child you specify

INSERT METHODS

- `node.appendChild(new_node)` adds a node at the end of the list of children
- `parent.insertBefore(new_node,node)` adds a node right before a child you specify
- `node.insertAdjacentElement(position,new_node)` adds a node into a specified position: `afterbegin`, `afterend`, `beforebegin`, `beforened`

Removing elements

```
<p id="demo">Some <strong>text</strong>.</p>
<script>
  let parent = document.querySelector("#demo");
  let child = parent.querySelector("strong");
  parent.removeChild(child);
</script>
```

REMOVING ELEMENTS

```
<p id="demo">Some <strong>text</strong>.</p>
<script>
  let parent = document.querySelector("#demo");
  let child = parent.querySelector("strong");
  parent.removeChild(child);
</script>
```

Notice that `querySelector` can also be used as an element method, not only on the `document` object.

REPLACING ELEMENTS

```
<p id="demo">Some <strong>text</strong>.</p>
<script>
  let span = document.createElement("span");
  let node = document.createTextNode("Some new text");
  let parent = document.querySelector("#demo");
  let strong = parent.querySelector("strong");
  span.appendChild(node);
  parent.replaceChild(span, strong);
</script>
```

Replacing elements

```html
<p id="demo">Some <strong>text</strong>.</p>
<script>
  let span = document.createElement("span");
  let node = document.createTextNode("Some new text");
  let parent = document.querySelector("#demo");
  let strong = parent.querySelector("strong");
  span.appendChild(node);
  parent.replaceChild(span, strong);
</script>
```

The code above creates a new `span` including text that replaces the existing `strong`.

DOM SELECTORS

- getElementsByTagName
- getElementsByClassName
- getElementById

- querySelector
- querySelectorAll

- getAttribute
- setAttribute

Adding, replacing, removing

- innerHTML
- innerText
- textContent

- createElement
- createTextNode

- appendChild
- removeChild
- replaceChild

Changing styles

- style.{property} //ok

- className //recommended
- classList //recommended

- classList.add //check canisue.com
- classList.remove //check canisue.com
- classList.toggle //check canisue.com

JavaScript Events

What are events?

Events are notable things in the DOM that JavaScript detects and can **react** to.

What are events?

Events are notable things in the DOM that JavaScript detects and can **react** to.

When an event occurs on a target element, a **handler** is executed.

What are events?

Events are notable things in the DOM that JavaScript detects and can **react** to.

When an event occurs on a target element, a **handler** is executed.

Events are an essential part of a dynamic website.

What are events?

Events are notable things in the DOM that JavaScript detects and can **react** to.

When an event occurs on a target element, a **handler** is executed.

Events are an essential part of a dynamic website.

You can find a complete list of events on w3schools.com.

Common events

Event	Description
`onclick`	occurs when the user clicks on an element
`onload`	occurs when an object has loaded
`onunload`	occurs once a page has unloaded (for `body`)
`onchange`	occurs when the content of a form element changed (for `input`, `select`, `textarea`)
`onmouseover`	occurs when the pointer is moved over an element or its children
`onfocus`	occurs when an element gets focus
`onblur`	occurs when an element loses focus

Handling Events

Events can be added to HTML elements as attributes.

```
<h2 onclick="this.innerHTML = 'These are events!'">What are events?</h2>
```

Handling Events

Events can be added to HTML elements as attributes.

```
<h2 onclick="this.innerHTML = 'These are events!'">What are events?</h2>
```

With `onclick` we define that the function we want will be executed. In this case we defined the function directly.

HANDLING EVENTS

Events can be added to HTML elements as attributes.

```
<h2 onclick="this.innerHTML = 'These are events!'">What are events?</h2>
```

With `onclick` we define that the function we want will be executed. In this case we defined the function directly.

But you can also call functions. Like I did on this title.

SIMPLE EVENT HANDLER

Events handlers can be assigned to elements in the JS file.

```
var h2 = querySelector("h2");
h2.onclick = function() { writeIntoConsole() };

function writeIntoConsole(){
  console.log(writeIntoConsole);
  alert("Open console!");
}
```

SIMPLE EVENT HANDLER

Events handlers can be assigned to elements in the JS file.

```
var h2 = querySelector("h2");
h2.onclick = function() { writeIntoConsole() };

function writeIntoConsole(){
  console.log(writeIntoConsole);
  alert("Open console!");
}
```

You can attach events to almost all HTML elements.

`onload`

`onload` events can be used if you want to perform actions after the page is loaded.

```
<body onload="writeInConsole()">
```

`onload`

`onload` events can be used if you want to perform actions after the page is loaded.

```
<body onload="writeInConsole()">
```

```
window.onload = function{
  writeInConsole();
}
```

Event Listener

Adding events as HTML attributes as well as simple event handlers have one disadvantage:

Event Listener

Adding events as HTML attributes as well as simple event handlers have one disadvantage:

You can only add **one event handler** to the target element.

Event Listener

Adding events as HTML attributes as well as simple event handlers have one disadvantage:

You can only add **one event handler** to the target element.

The `addEventListener` method in JavaScript allows you to add **many event handlers** (even of the same type) to one element.

An example

```
document.querySelector("h2").addEventListener("click", writeInConsole);
```

An example

```
document.querySelector("h2").addEventListener("click", writeInConsole);
```

- The first parameter is the **event type**, e.g. `click` or `mouseover`. Note that there is no **on**-prefix anymore.

An example

```
document.querySelector("h2").addEventListener("click", writeInConsole);
```

- The first parameter is the **event type**, e.g. `click` or `mouseover`. Note that there is no **on**-prefix anymore.
- The second parameter is the **function** that gets called when the event occurs.

An example

```
document.querySelector("h2").addEventListener("click", writeInConsole);
```

- The first parameter is the **event type**, e.g. `click` or `mouseover`. Note that there is no **on**-prefix anymore.
- The second parameter is the **function** that gets called when the event occurs.
- The third paramter (optional) is the **useCapture** boolean defining the order of event handling.

Event propagation

Imagine you have a child node `<p>` and a parent node `<section>`. Both have `onclick` events. Now you click on the child. Which function should be executed first?

Event propagation

Imagine you have a child node `<p>` and a parent node `<section>`. Both have `onclick` events. Now you click on the child. Which function should be executed first?

This is called **event propagation** and it defines which order the event handlers have.

Event propagation

Imagine you have a child node `<p>` and a parent node `<section>`. Both have `onclick` events. Now you click on the child. Which function should be executed first?

This is called **event propagation** and it defines which order the event handlers have.

- Bubbling: the inner most event is handled first. This is **default**.

Event propagation

Imagine you have a child node `<p>` and a parent node `<section>`. Both have `onclick` events. Now you click on the child. Which function should be executed first?

This is called **event propagation** and it defines which order the event handlers have.

- Bubbling: the inner most event is handled first. This is **default**. **Bubbling goes UP the DOM.**

Event propagation

Imagine you have a child node `<p>` and a parent node `<section>`. Both have `onclick` events. Now you click on the child. Which function should be executed first?

This is called **event propagation** and it defines which order the event handlers have.

- Bubbling: the inner most event is handled first. This is **default**.
  Bubbling goes UP the DOM.
- Capturing: the outer most event is handled first.

Event propagation

Imagine you have a child node `<p>` and a parent node `<section>`. Both have `onclick` events. Now you click on the child. Which function should be executed first?

This is called **event propagation** and it defines which order the event handlers have.

- Bubbling: the inner most event is handled first. This is **default**.
  Bubbling goes UP the DOM.
- Capturing: the outer most event is handled first.
  Capturing goes DOWN the DOM.

EVENT HANDLING ORDER

If you want to use capturing, set the optional third parameter `useCapture` to **True**.

```
document.querySelector("h2").addEventListener("click", writeInConsole,true);
```

When to use useCapture

As the default state is **False** and bubbling propagation is used, the question you probably ask yourself is:

What is capturing propagation good for?

When to use `useCapture`

As the default state is **False** and bubbling propagation is used, the question you probably ask yourself is:

What is capturing propagation good for?

- If you want the click handler on the parent to be executed first.

When to use `useCapture`

As the default state is **False** and bubbling propagation is used, the question you probably ask yourself is:

What is capturing propagation good for?

- If you want the click handler on the parent to be executed first.
- If you have the same events handled for multiple elements, e.g. all `input` elements on `focus`.

When to use `useCapture`

As the default state is **False** and bubbling propagation is used, the question you probably ask yourself is:

What is capturing propagation good for?

- If you want the click handler on the parent to be executed first.
- If you have the same events handled for multiple elements, e.g. all `input` elements on `focus`.
- On events which do not support bubbling, e.g. `load` and `blur`. More info on MDN.

Removing events

The `removeEventListener` method will remove an event handler which was set using `addEventListener`

```
document.querySelector("h2").removeEventListener("click", writeInConsole);
```

Removing events

The `removeEventListener` method will remove an event handler which was set using `addEventListener`

```
document.querySelector("h2").removeEventListener("click", writeInConsole);
```

It's exactly the same, just replace `add` with `remove`.

Online ressources

- W3schools about DOM
- MDN event reference
- Javascript Key codes
- Can i use ...: always check which browsers are supported
- Codepen: find HTML/CSS and JS snippets online

Exercises

Manipulate the DOM

- Create a HTML file with at least 3 elements: `h1`, `p` and `a`
- Assign a new variable for each of these 3 elements.
- For the `h1` variable write a loop with 2 iterations, always adding the number of the iteration to the content of the tag (after the existing content).
- For the `p` variable write a loop with 4 iterations, always adding the number of the iteration to the content of the tag (before the existing content).
- For the `a` variable write a loop with 7 iterations, always replacing the content of the tag with the number of the iteration.

The Book List

Create a webpage with an `h1` of "My Book List".

Add a script tag to the bottom of the page, where all your JS will go. Copy this array of books:

```
var books = [
  {
    title: 'The Design of EveryDay Things',
    author: 'Don Norman',
    alreadyRead: false
  }, {
    title: 'The Most Human Human',
    author: 'Brian Christian',
    alreadyRead: true
  }
];
```

» go to next page

The Book List

Iterate through the array of books. For each book, create a `p` element with the book title and author and append it to the page.

Bonuses:

- Use a `ul` and `li` to display the books.
- Add an `img` to each book that links to a URL of the book cover.
- Change the style of the book depending on whether you have read it or not.

About me

Start with this HTML and save it as "about_me.html":

```html
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8"/>
    <title>About Me</title>
  </head>
  <body>
    <h1>About Me</h1>

    <ul>
      <li>Nickname: <span id="nickname"></span>
      <li>Favorites: <span id="favorites"></span>
      <li>Hometown: <span id="hometown"></span>
    </ul>
  </body>
</html>
```

» go to next page

About me

- Add a `script` tag to the bottom of the HTML body.
- (In the JavaScript) Change the body tag's style so it has a font-family of "Arial, sans-serif".
- (In the JavaScript) Replace each of the spans (nickname, favorites, hometown) with your own information.
- Iterate through each `li` and change the class to "list-item".
- (In the HTML `head`) Add a `style` tag that sets a rule for `.list-item` to make the color red.
- Create a new `img` element and set its `src` attribute to a picture of you. Append that element to the page.

Move Box

- Make a container with a box inside.
- Make the box move with your scroll wheel.
- Find out more about the scroll wheel event.

WORK ON YOUR PROJECT