COESIONE ITALIA 21-27 PIEMONTE
Cofinanziato dall'Unione europea
REGIONE PIEMONTE

Web Developer

Programmazione - Javascript e Typescript

Docente: Shadi Lahham

Intervento realizzato da ITS TECNOLOGIE DELL'INFORMAZIONE E DELLA COMUNICAZIONE

The DOM

Document Object Model

Shadi Lahham - Web development

Accessing the DOM

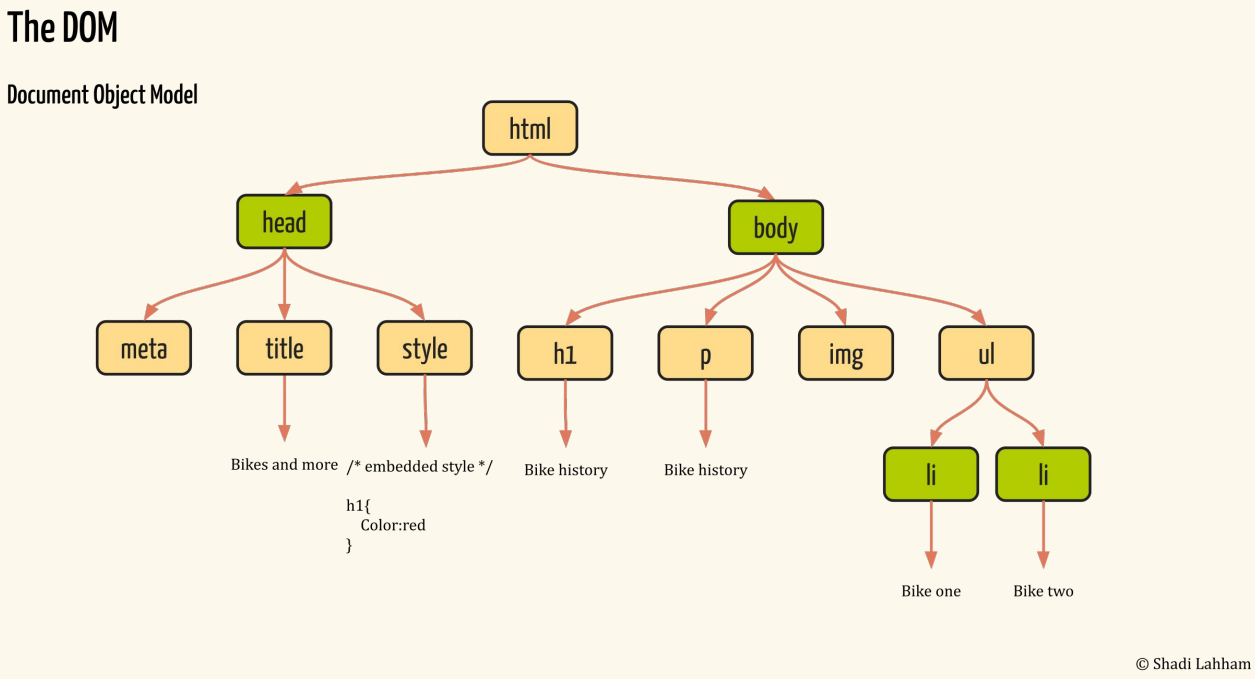
What is the DOM?

- Document Object Model - a map of our HTML document
- The logical structure of an HTML document, how it is accessed and manipulated
- Elements in an HTML document can be accessed, changed, deleted, or added using the DOM opzioni che si possono fare verso gli elementi del documento

DOM (Document Object Model) rappresenta la STRUTTURA LOGICA di un documento HTML come una mappa, dove ogni elemento della pagina è accessibile e modificabile.

È la base per accedere, manipolare, e aggiornare dinamicamente i contenuti, elementi e lo stile di una pagina web.

What is the DOM?



Inspecting the DOM

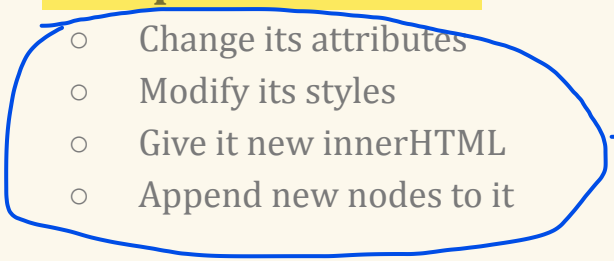
- Google Chrome or Mozilla Firefox
  - Right-click on a web page and select 'Inspect Element'
  - Windows shortcuts
    - Ctrl-Shift-i
    - or F12
  - macOS
    - Cmd-Opt-i
- Safari
  - Unlock the Develop Menu by opening Safari > Preferences > Advanced, and checking the box, 'Show Develop menu in menu bar'
  - Access by same methods as Chrome

Try it now on any web page

Accessing the DOM

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 using an access method
2. **Store** this DOM node as a variable
3. **Manipulate** the DOM node
    - Change its attributes
    - Modify its styles
    - Give it new innerHTML
    - Append new nodes to it



cosa si può fare all'elemento dopo averlo memorizzato

DOM Access Methods

```
// finding DOM nodes using the id:
document.getElementById(id);

// finding DOM nodes use the tag name:
document.getElementsByTagName(tagName);

// finding DOM nodes using the class name:
document.getElementsByClassName(className);

// finding DOM nodes using a query selector:
document.querySelector(cssQuery);
document.querySelectorAll(cssQuery);
```

02. getElementsByTagName, getElementsByClassName e querySelectorAll:

restituiscono collezioni di nodi simili ad array, che possono essere iterate.

01. getElementById e querySelector:

restituiscono un singolo nodo

Selecting Nodes From the DOM

index.html

```
<ul id="hobby-list">
  <li class="hobby">Playing the banjo</li>
  <li class="hobby">Paddleboarding</li>
</ul>
```

main.js

```
// by Id
let hobbiesListElement = document.getElementById('hobby-list');

// by Tag Name
let hobbies = document.getElementsByTagName('li');

// by Class Name
let alsoHobbies = document.getElementsByClassName('hobby');
```

Selecting Nodes From the DOM

index.html

```
<ul id="hobby-list">
  <li class="hobby">Playing the banjo</li>
  <li class="hobby">Paddleboarding</li>
</ul>
```

main.js

```
// by CSS Query
let firstHobby = document.querySelector('ul li.hobby');
let allItems = document.querySelectorAll('ul li.hobby');
```

Return values

01. getElementById e querySelector:

restituiscono un singolo nodo

```
// a single DOM node
let hobbiesListElement = document.getElementById('hobby-list');
let firstHobby = document.querySelector('ul li.hobby');
```

return value
getElementById() and querySelector() return a **single value**

Return values

02. getElementsByTagName, getElementsByClassName e querySelectorAll:

restituiscono collezioni di nodi simili ad array, che possono essere iterate.

```
// a collection of nodes
let hobbies = document.getElementsByTagName('li');
let alsoHobbies = document.getElementsByClassName('hobby');
let allItems = document.querySelectorAll('ul li.hobby');
```

return value
getElementsByClassName(), getElementsByTagName(), querySelectorAll() return a **collection** of items that behaves like an array, an **array-like**

Return values

```
// a single DOM node
let hobbiesListElement = document.getElementById('hobby-list');
let firstHobby = document.querySelector('ul li.hobby');

// all true
console.log(firstHobby instanceof Node);
console.log(firstHobby instanceof Element);
console.log(firstHobby instanceof HTMLElement);
console.log(hobbiesListElement instanceof Node);
console.log(hobbiesListElement instanceof Element);
console.log(hobbiesListElement instanceof HTMLElement);
```

return value

getElementById() and querySelector() return an HTMLElement which is basically a DOM Node

Return values

```
// a collection of nodes
let hobbies = document.getElementsByTagName('li'); // an HTMLCollection
let alsoHobbies = document.getElementsByClassName('hobby'); // an HTMLCollection
let allItems = document.querySelectorAll('ul li.hobby'); // a NodeList

// iterate over the collection
Array.from(hobbies).forEach(item => console.log(item.textContent));
Array.from(alsoHobbies).forEach(item => console.log(item.textContent));
allItems.forEach(item => console.log(item.textContent));
```

return value
An array-like which is either a live HTMLCollection which changes with DOM changes or a static NodeList which doesn't change

While an array-like can be looped over it doesn't have most of the array methods
To use it like an array convert it to an array using Array.from()

HTMLCollection vs NodeList

JavaScript HTML DOM Collections
HTMLCollection vs NodeList : The main difference
HTMLCollection vs NodeList - the Difference

NodeList
HTMLCollection

Return values

```
// a collection of nodes
let catNames = document.querySelectorAll('ul li.catname');
let firstCatName = catNames[0];

/// but shouldn't it be an array like?
catNames.forEach(catName => console.log(catName.textContent));

// all false - prolly the prev, here check for HTMLCollection
console.log(catNames instanceof Node);
console.log(catNames instanceof Element);
console.log(catNames instanceof HTMLElement);
```

return value
getElementsByClassName(), getElementsByTagName(), and querySelectorAll() return an HTMLCollection which is a collection of HTML elements

It behaves **like an array** (also called **array-like**), meaning it has a length property and can be iterated, but doesn't have array methods

Manipulating the DOM

Manipulating attributes

Attributi: Puoi modificare gli attributi di un nodo HTML usando JavaScript.

index.html

```
<img id="my-cat" src="https://picsum.photos/200/300">
```

main.js

```
// select the DOM node
let catImage = document.getElementById('my-cat');

// access and change attributes of a DOM node using dot notation
// change the src of an image
let oldImageSource = catImage.src;
catImage.src = 'https://picsum.photos/300/200';

// change the className of the DOM node
catImage.className = 'portrait';
```

Manipulating style - bad example

Stili: Si possono applicare stili in modo diretto o aggiungere classi.
Per evitare mescolare HTML e CSS, è preferibile usare le classi CSS.

è detto SEPARAZIONE DELLE RESPONSABILITA!!!

style.css

```css
body {
  color: green;
  background-color: white;
  padding-top: 12px;
}
```

main.js

```js
let pageNode = document.body;
pageNode.style.color = 'red';
pageNode.style.backgroundColor = 'pink';
pageNode.style.paddingTop = '10px';
```

Note:

changing DOM node styles requires camelCasing hyphenated CSS properties and adding units to numbers
inline styling should be avoided due to separation of concerns and code maintainability

Manipulating style

preferibile come detto prima aggiungere le classi con regole di stile da applicare ai nodi degli elementi

style.css

```
body {
  color: green;
  background-color: white;
  padding-top: 12px;
}

.custom-style {
  color: red;
  background-color: pink;
  padding-top: 10px;
}
```

main.js

```
let pageNode = document.body;
pageNode.classList.add('custom-style');
```

note: cleaner code and better separation of concerns

Manipulating content - bad example

```
// each DOM node has an innerHTML attribute which contains the HTML of all its children
let pageNode = document.body;
console.log(pageNode.innerHTML);

// set innerHTML to replace the contents of the node
pageNode.innerHTML = '<h1>Oh, no! Everything is gone!</h1>';

// or add to innerHTML instead
pageNode.innerHTML += '<p>P.S. Please do write back</p>';
```

Contenuto: Usare textContent per modificare solo il testo è più sicuro rispetto a innerHTML, che permette modifiche dirette dell'HTML e può esporre la pagina a vulnerabilità di sicurezza (come XSS).

Manipulating content

```
// select the pageNode
let pageNode = document.body;

// create new DOM nodes
let headingNode = document.createElement('h1');
headingNode.textContent = 'Oh, no! Everything is gone!';

let paragraphNode = document.createElement('p');
paragraphNode.textContent = 'P.S. Please do write back';

// replace all existing child nodes of pageNode with the new nodes
pageNode.replaceChildren(headingNode, paragraphNode);
```

textContent vs. innerHTML

For modifying text within a node, use textContent instead of innerHTML

textContent

- Works in newer browsers
- Faster: doesn't require HTML parsing
- More secure: prevents code execution

innerHTML

- Compatible with older browsers
- More powerful: can manipulate HTML code directly
- More dangerous: vulnerable to cross-site scripting (XSS) and potential syntax errors

Creating nodes

Creating DOM Nodes

Per creare nuovi elementi da zero, puoi usare metodi come:

- document.createElement(tagName)
- document.createTextNode(text).

```
// the document object also allows us to create new nodes from scratch
document.createElement('tagName');
document.createTextNode('text');
document.appendChild(childToAppend);

// example - adding content dynamically
let body = document.body;
let newImg = document.createElement('img');
newImg.src = 'https://placehold.co/300x200';
newImg.style.border = '1px solid black';
body.appendChild(newImg);

let newParagraph = document.createElement('p');
let newText = document.createTextNode('new paragraph content!');
newParagraph.appendChild(newText);
body.appendChild(newParagraph);
```

Node reference

properties

Node properties

methods

Node methods

JavaScript HTML DOM Elements (Nodes)

Your turn

1.About me

Start with this 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>
```

Continues on next page >>>

1.About me

- Add an external javascript file called main.js
- In JavaScript:
  - Change the body style so it has a font-family of "Arial, sans-serif"
  - Replace each of the spans (nickname, favorites, hometown) with your own information
  - Iterate through each li and change the class to "list-item"
  - Create a new img element and set its src attribute to a picture of you
  - Append that element to the page
- Add an external css file using Javascript
  - The external css file should make items with the .list-item class white, bold and with an orange background
  - The external css file should be applied after 4 seconds

2.Book list

Use an array of books like this
You should have at least 4 books

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

Continues on next page >>>

2.Book list

- Create a *complete* webpage with a title, description and all other HTML tags
- In the body add an h1 title of "My Book List"
- In javascript, iterate through the array of books.
    - For each book, create HTML element with the book title and author and append it to the page
    - Use a ul and li to display the books
    - Add a url property to each book object that contains the cover image of the book
    - Add the image to the HTML using Javascript
    - Using javascript change the style of the book depending on whether you have read it or not
- Add an external css file that applies after 5 seconds
    - Now change the style of the book depending on whether you have read it or not using both css and javascript (the CSS should use a different color for read books)

3.DOM Detective

- Go to www.gog.com
- Use the devtools to view the DOM and write Javascript in the console
- Use the DOM access methods to find the following:
  - Every image on the page
  - The main menu at the top of the page
  - All the news items under "News"
  - The footer
  - All the social media links at the bottom of the page
- Produce a readme.md file with
  - snippets of your Javascript code
  - explanations of what which elements they select

4.Custom Detective

- Choose a news website that you like
- Use the devtools to view the DOM and write Javascript in the console
- Use the DOM access methods to find:
  - At least 10 different elements or collections of elements in the page
  - Choose interesting elements that require complex selectors to reach
- Produce a readme.md file with
  - A link to the website that you chose
  - snippets of your Javascript code
  - explanations of what which elements they select

Bonus

5.Arrivals

- Implement the arrivals page of an airport such as this one
  - Create a complete proper webpage with a title, description and all other HTML tags
  - Add Javascript and CSS files
  - Include as much detail as you can to each flight row
  - Add a Status to each flight. Status can be DEPARTING, DELAYED, ON_TIME, ARRIVED, etc
- Simulate a real arrivals list
  - The list should start empty and update every 10 seconds
  - Flights that have arrived should be removed after 60 seconds
  - Flights should change status in time. E.g. departing>on_time>delayed>arrived
  - Flights that are delayed should be displayed in red
  - New flights should be added to the bottom of the list
  - The list should be sorted by date and hour

6.Identity Hijack

Change the Stanford website using elements from the Berkeley website

- Brand and name
  - Find any elements with the word 'Stanford' and replace it with 'Berkeley'
  - Remember to change the title of the page as well
  - Replace any symbols of Stanford University with Berkeley
- Colors
  - Find all elements with the 'Stanford' color(s) and replace them with the 'Berkeley' color(s)
- Links
  - Manually find all the links in the navigation area and replace them with references to the Berkeley website if there are similar pages there. Otherwise links should point to the Berkeley homepage
- Submit a Javascript file with all the changes

7.The DOM washer

Create a simulation of a dishwasher system using two stacks of dishes

- one stack represents dirty dishes, and the other represents clean dishes
- the dirty stack has a random number of plates 10 - 15
- useful functions
  - washDish - moves a dish from the dirty stack to the clean stack
  - drawStacks - displays the current state of both stacks in the page updating the DOM
  - runSimulation - simulate washing all dirty dishes adding a random delay between steps
- Use correct HTML and CSS as needed for this exercise

Bonus

1. have three stacks of dirty dishes and one clean stack
2. the dishwasher is able to wash two dishes at a time

References

JavaScript HTML DOM
JavaScript DOM HTML
JavaScript DOM CSS
JavaScript DOM Elements

More detailed
The Document-Object Model
Node methods | MDN