

JavaScript nel browser

- Integrazione tra JavaScript e HTML
 - BOM e DOM
- JSON, AJAX
- Progetto di riferimento
 - <https://github.com/egalli64/nesp> (*modulo 3b*)
 - Node.js + Express
 - VS Code

HTML – JavaScript

- Elemento **script**

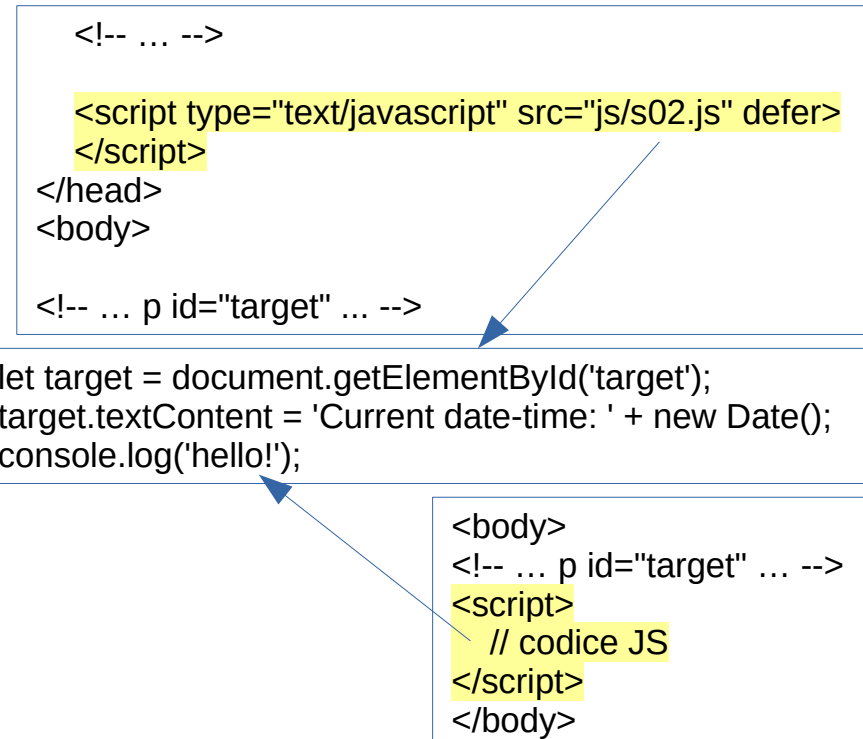
- in **head**, con attributo **defer**
 - Solo per codice esterno
 - (*async* per caricamento in parallelo)
- nel **body**, ultimo elemento

- Il codice può essere:

- Scritto direttamente nell'elemento script
 - sconsigliato in produzione
- Caricato da un file JS esterno
 - specificato nell'attributo **src**

- Commenti JavaScript

- `//` termina a fine riga
- `/*` terminazione esplicita `*/`



BOM: Browser Object Model

- Definito informalmente: API del browser per l'accesso alle sue funzionalità via JS
 - **A**pplication **P**rogramming **I**nterface
- Una pagina web viene visualizzata in un oggetto **window**
 - outerHeight, outerWidth, innerHeight, innerWidth
 - alert(message)
 - confirm(message) // true = OK
- Navigazione nella cronologia via **history**
 - back()
 - forward()
- Il documento HTML corrente viene rappresentato in **document**



DOM: Document Object Model

- La pagina corrente è **document**, istanza della classe **Document**
 - Albero che rappresenta il documento (tipicamente **HTMLDocument**) gestito dal browser
- Tra le proprietà
 - title, URL (read only), head (read only), body, ...
- Tra i metodi
 - **getElementById(id)** // Element con il dato id, o null
 - **getElementsByClassName(classes)** // lista di Element di quella classe (o classi)
 - **getElementsByTagName(tag)** // lista di Element del dato tag
 - **querySelector(selectors)** // primo Element per il selettore(/i) CSS passato
 - **querySelectorAll(selectors)** // lista di Element
 - **createElement(tag)** // factory method
 - **createTextNode(data)** // factory method

DOM EventTarget, Node, Element

- EventTarget
 - Per gli oggetti abilitati alla gestione degli eventi: `addEventListener()`, `removeEventListener()`
- Node
 - Estende EventTarget
 - Base per molti oggetti nel DOM, tra cui: Document, Element
 - Tra le proprietà e i metodi:
 - `firstChild`, `lastChild`, `textContent`
 - `appendChild(child)`, `removeChild(child)`
- Element
 - `innerHTML`: il content dell'elemento, gestito come frammento HTML
 - `style`: lo stile CSS dell'elemento corrente



Eventi su documento

- Associazione di eventi su elemento a codice JavaScript via attributo **on**...
 - Nel documento HTML on in JavaScript
- Per submit, se il risultato è false il comportamento standard viene annullato
 - Se non c'è un comportamento di default, il nostro gestore non deve ritornare niente

```
<form action="first" onsubmit="return check();">  
  <input id="x">  
  <button>OK</button>  
</form>
```

Sconsigliato
in produzione

```
function check() {  
  if (document.getElementById('x').value.length == 0) {  
    return false;  
  }  
  return true;  
}
```

```
<form action="second" id="second">  
  <input id="y">  
  <button>OK</button>  
</form>
```

```
document.getElementById('second').onsubmit = () => {  
  if (document.getElementById('y').value.length == 0) {  
    return false;  
  }  
  return true;  
};
```



addEventListener()

- Ogni oggetto DOM possibile target di un evento implementa EventTarget
- Il metodo EventTarget.addEventListener() registra un gestore per un tipo di evento sul target
 - In questo caso è necessario invocare preventDefault() sull'evento per far sì che il comportamento standard non sia eseguito
- EventTarget.removeEventListener() per rimuovere la funzionalità associata

Moderno

```
<form action="second" id="second">  
  <input id="y">  
  <button>OK</button>  
</form>
```

```
document.getElementById('second').addEventListener('submit', (event) => {  
  if (document.getElementById('y').value.length == 0) {  
    event.preventDefault();  
  }  
});
```

Eventi & attributi

- Caricamento completo in window del documento HTML: `onload`
- Caricamento del documento, potrebbero mancare CSS, immagini, ... :
`DOMContentLoaded` – non esiste un attributo specifico → `addEventListener()`
- Click del button submit in form: `onsubmit`
- Input prende/perde focus: `onfocus`, `onblur`
- Input blur + cambiamento: `onchange`
- Click su un elemento: `onclick`, `ondblclick`
- Mouse entra/esce: `onmouseover`, `onmouseout`
- ...

Gestione dell'attributo class

- La gestione diretta dell'attributo class è scomoda e può causare facilmente errori
- La proprietà **classList** di Element ne semplifica la gestione
- Le classi sono viste come elementi di una DOMTokenList, modificabile via:
 - add(): aggiunge una o più classi alla lista (no duplicati)
 - remove(): rimuove una o più classi alla lista
 - toggle(): toglie la classe se c'è, altrimenti l'aggiunge
 - replace(): toglie la prima classe passata, aggiunge la seconda
- Inoltre:
 - contains(): controlla se la classe passata è in lista
 - item(): ritorna la classe in base all'indice passato (o null)

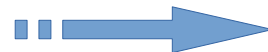
JSON

- JavaScript Object Notation
- Formato per lo scambio di dati basato su
 - Coppie nome-valore (oggetto JS)
 - Array di valori
- Da JSON a stringa
 - JSON.stringify()
- Da stringa a JSON
 - JSON.parse()

```
{  
  name: "tom",  
  job: {  
    title: "developer",  
    languages: ["JavaScript", "HTML", "CSS"]  
  }  
}
```

AJAX e XMLHttpRequest

- **Asynchronous JavaScript And XML**
- Uso dell'oggetto XMLHttpRequest per comunicare con il server (XML, JSON, testo semplice, ...) senza lasciare la pagina corrente
- Dopo aver creato un oggetto XMLHttpRequest
 - Si indica una callback in onload (o onreadystatechange)
 - In onerror si può indicare la callback da eseguire se il browser non riesce ad eseguire la request
 - Si invoca open() indicando il metodo HTTP e l'URL della risorsa richiesta
 - Same-origin policy, l'accesso è garantito per default alle sole risorse dell'app corrente
 - Da altre app se è abilitato CORS (Cross-Origin Resource Sharing)
 - E infine send()



Esempio AJAX

```
<textarea id="target"></textarea>
<button onclick="getInfo();">Get programmer info</button>
```

Vedi anche JQuery, Axios, ...

```
function getInfo() {
  let request = new XMLHttpRequest();
  request.onload = callback;
  request.open("GET", "data/tom.json");
  request.send();
}
```

```
function callback() {
  let target = document.getElementById('target');
  if (this.status !== 200) {
    target.value += "[" + this.status + "]\n";
    return;
  }
  let json = JSON.parse(this.responseText);


  target.value += json.name + '\n';
  target.value += json.job.title + '\n';
  target.value += json.job.languages + '\n';
}
```

```
{
  "name": "tom",
  "job": {
    "title": "developer",
    "languages": ["JavaScript", "HTML", "CSS"]
  }
}
```

Fetch

- Nuova versione di XMLHttpRequest basata su Promise
- Una request via fetch può essere eseguita semplicemente
 - Passando l'URL della risorsa da accedere via HTTP GET
 - Si ottiene una promessa con la response
 - In caso di errore (404, 500, ...) la proprietà **ok** sarà false
 - Si invoca il metodo **json()** – alternative: **text()**, **blob()**, ...
- È possibile passare un secondo parametro
 - Un oggetto contenente opzioni come **method**, **headers**, ...

Node + Express

- Da una nuova directory:
 - npm init 
 - npm install express --save
 - crea il file server.js
 - esegui l'app
 - node server.js
 - Accedi all'app via browser, porta 8080



server.js

```
let express = require('express');
let app = express();

app.get('/', function (req, res) {
  res.send('Hello World');
});

app.listen(8080, function () {
  console.log('Listening on port 8080');
});
```