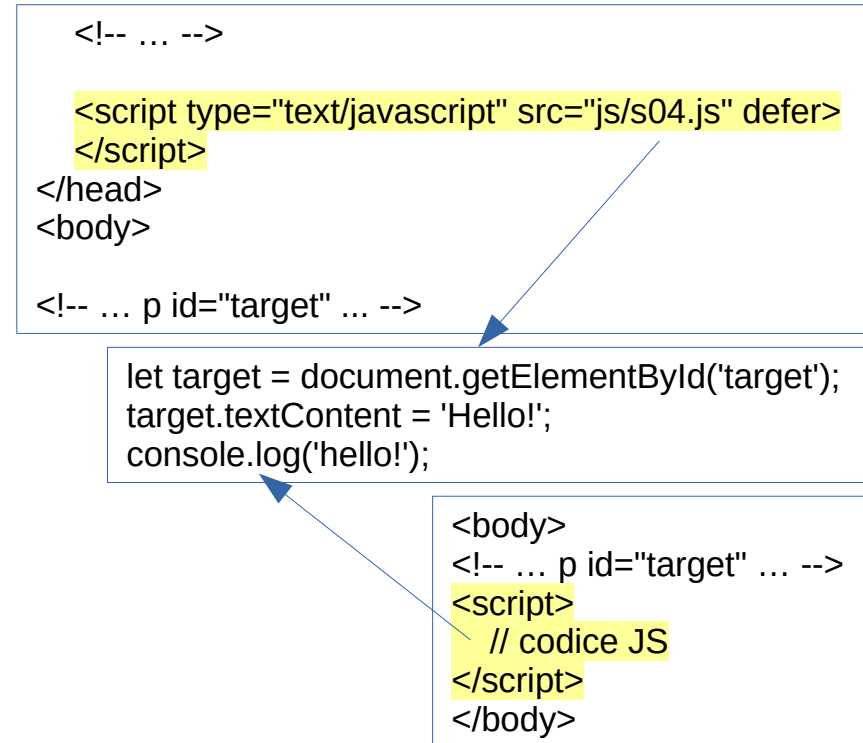


# JavaScript nel browser


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

# 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: oggetti nel browser esposti per l'accesso via JS
- 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**
  - bgColor, fgColor: colore dello sfondo e del testo
  - title, URL
  - forms: array dei form nella pagina
    - ogni form è accessibile per indice o per 'name'
  - **Getter di elementi**
    - getElementById(*id*) // Element con l'id specificato
    - getElementsByClassName(*classes*) // lista di Element della classe (o classi) passata
    - getElementsByTagName(*tag*) // lista di Element del tag passato presenti nel document
    - getElementsByName(*name*) // lista di Element con l'attributo "name" specificato
    - querySelector(*selectors*) // primo Element che corrisponde al selettore(/i) CSS passato
    - querySelectorAll(*selectors*) // lista di Element corrispondenti al selettore(/i) CSS passato

# DOM Node

- Interfaccia alla base di molti oggetti nel DOM
  - Tra cui: Document, Element
  - A sua volta estende EventTarget
- Tra le proprietà e i metodi:
  - firstChild, lastChild
  - appendChild(child)
  - removeChild(child)

# 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 true/false

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

```
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 l'interfaccia `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

```
<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 definisce una callback in onload (o onreadystatechange)
  - Si invoca open() per definire la risorsa richiesta sul server
  - 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"]
  }
}
```

# Node + Express

- Da una nuova directory:
  - npm init 
  - npm install express --save
  - crea il file index.js 
  - esegui l'app
    - node index.js
  - Accedi all'app via browser, porta 3000

index.js

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

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

app.listen(3000, function () {
  console.log('Listening on port 3000');
});
```