CURS 4

JavaScript - Programarea Interactivității Web

4.1 Ce este JavaScript și Cum Funcționează în Browser

JavaScript este singurul limbaj de programare nativ al browser-ului. Este interpretat și executat direct de motorul JavaScript al browser-ului (V8 în Chrome, SpiderMonkey în Firefox).

Caracteristici esențiale JavaScript

- Dinamic: tipurile variabilelor sunt determinate la execuție, nu la compilare
- Interpretat (JIT compiled): nu necesită compilare separată - browserul îl rulează direct
- Single-threaded: rulează pe un singur fir de execuție (event loop pentru asincronism - Cursul 5)
- Bazat pe prototipuri: moștenire prin prototipuri (nu clase tradiționale - deși ES6 adaugă sintaxa class)
- First-class functions: funcțiile sunt valori - pot fi stocate în variabile, pasate ca argumente, returnate

> Node.js (pe care îl vom folosi din Cursul 9) permite rularea JavaScript pe server, în afara browserului - același limbaj, context diferit. Aceasta este una din marile avantaje ale JavaScript: un singur limbaj pentru front-end și back-end.

4.2 Variabile și Tipuri de Date

Declararea variabilelor: const, let, var

```
// const - valoare care nu se reatribuie (bloc-scoped)
const PI = 3.14159;
const utilizator = { nume: 'Ana', varsta: 28 };
// utilizator = {} este EROARE - nu poți reatribui const
// utilizator.nume = 'Ion' este OK - modifici conținut, nu referința

// let - valoare care se poate reatribui (bloc-scoped)
let contor = 0;
contor = contor + 1;  // OK
contor++;             // shorthand

// var - EVITAT în codul modern
// Problems: function-scoped (nu bloc), hoisted, reatribuibil
var nume = 'vechi';  // nu folosi
```

> Regula simplă: Folosește CONST implicit. Treci la LET doar dacă știi că vei reatribui variabila. Nu folosi VAR niciodată în cod nou.

Cele 7 tipuri primitive JavaScript

Tip	Exemple	Note importante
string	"text" 'text' `template ${literal}`	Imutabile. Metode utile: .length, .toUpperCase(), .includes(), .split(), .trim(), .replace(), .slice()
number	42, 3.14, -7, 0.1 + 0.2 === 0.3 // FALSE!	Toate numerele sunt float-uri (IEEE 754). NaN = Not a Number. Infinity. 0.1 + 0.2 !== 0.3 (float precision!)
boolean	true, false	Falsy values: false, 0, "", null, undefined, NaN. Orice altceva e truthy.
null	null	Absența intenționată a unei valori. typeof null === "object" - bug istoric în JS.
undefined	undefined	Variabilă declarată dar neatribuită. typeof undefined === "undefined"
symbol	Symbol('id')	Identificatori unici (utilizare avansată, rareori în cod de zi cu zi)
bigint	9007199254740993n	Numere întregi arbitrar de mari. Adăugat în ES2020.

Operatori și Coerciție de Tip

```
// === vs == - ÎNTOTDEAUNA folosește ===
0 == '0'    // true  - coerciție de tip (periculos!)
0 === '0'   // false - comparație strictă (corect!)
null == undefined   // true
null === undefined  // false

// Operatori logici
true && false   // false (AND)
true || false   // true  (OR)
!true           // false (NOT)

// Nullish coalescing ?? - valoare implicită pentru null/undefined
const nume = utilizator.nume ?? 'Anonim';
// Spre deosebire de ||, ?? nu tratează 0 sau '' ca falsy

// Optional chaining ?. - accesează proprietăți fără erori
const oras = utilizator?.adresa?.oras;
// Dacă adresa e null/undefined, returnează undefined în loc de eroare
```

4.3 Funcții

Funcțiile sunt blocuri de cod reutilizabile. În JavaScript, funcțiile sunt 'first-class citizens' - pot fi tratate ca orice altă valoare.

```
// 1. Declarație de funcție - hoistată, disponibilă înainte de declarare
function saluta(nume) {
```

```
  return `Bună ziua, ${nume}!`;
}

// 2. Expresie de funcție - nu hoistată
const saluta = function(nume) {
  return `Bună ziua, ${nume}!`;
};

// 3. Arrow function - sintaxă concisă, fără propriul this
const saluta = (nume) => `Bună ziua, ${nume}!`;
// Arrow function cu bloc:
const saluta = (nume) => {
  const mesaj = `Bună ziua, ${nume}!`;
  return mesaj;
};

// Parametri cu valori implicite
function conecteaza(host, port = 3000, protocol = 'https') {
  return `${protocol}://${host}:${port}`;
}
conecteaza('exemplu.ro');          // https://exemplu.ro:3000
conecteaza('exemplu.ro', 8080);    // https://exemplu.ro:8080

// Rest parameters - colectează argumente rămase
function suma(...numere) {
  return numere.reduce((total, n) => total + n, 0);
}
suma(1, 2, 3, 4, 5);  // 15

// Funcții ca valori - callback
const numere = [3, 1, 4, 1, 5, 9];
const sortate = numere.sort((a, b) => a - b);
```

4.4 Array-uri și Obiecte

Array-uri - liste ordonate

```
const fructe = ['măr', 'banană', 'cireșe'];

// Accesare și modificare
fructe[0];           // 'măr'
fructe[fructe.length - 1];  // ultimul element
fructe.push('kiwi');         // adaugă la final
fructe.pop();                // elimină ultimul
fructe.unshift('ananas');    // adaugă la început
fructe.shift();              // elimină primul

// Metode funcționale (returnează array nou, nu modifică originalul)
const numere = [1, 2, 3, 4, 5];

// map - transformă fiecare element
const patrate = numere.map(n => n * n); // [1, 4, 9, 16, 25]

// filter - păstrează elementele care trec un test
const pare = numere.filter(n => n % 2 === 0); // [2, 4]

// reduce - acumulează o singură valoare
const suma = numere.reduce((acc, n) => acc + n, 0); // 15

// find - primul element care trece testul
const prima = numere.find(n => n > 3); // 4

// some / every - testează elemente
numere.some(n => n > 4);  // true (cel puțin unul)
```

```
numere.every(n => n > 0); // true (toți)

// Destructurare
const [primul, aldoilea, ...restul] = numere;
// primul = 1, aldoilea = 2, restul = [3, 4, 5]
```

Obiecte - structuri cheie-valoare

```
const utilizator = {
  id: 1,
  nume: 'Ana Popescu',
  email: 'ana@exemplu.ro',
  activ: true,
  adresa: {             // obiect imbricat
    oras: 'București',
    tara: 'România'
  },
  saluta() {            // metodă (funcție în obiect)
    return `Bună, sunt ${this.nume}`;
  }
};

// Accesare
utilizator.nume;           // 'Ana Popescu'
utilizator['email'];       // 'ana@exemplu.ro'
utilizator.adresa.oras;    // 'București'

// Destructurare obiect
const { nume, email, adresa: { oras } } = utilizator;

// Spread - copiere/fuzionare
const actualizat = { ...utilizator, activ: false, ultima_autentificare: new
Date() };

// Metode Object
Object.keys(utilizator);    // ['id', 'nume', 'email', ...]
Object.values(utilizator);  // [1, 'Ana Popescu', ...]
Object.entries(utilizator); // [['id', 1], ['nume', 'Ana'], ...]
```

4.5 Manipularea DOM

DOM (Document Object Model) este reprezentarea JavaScript a documentului HTML - un arbore de obiecte pe care JavaScript îl poate citi și modifica dinamic.

Selectarea elementelor

```
// Cel mai versatil - selector CSS, returnează primul element găsit
const titlu = document.querySelector('h1');
const primaClasa = document.querySelector('.card');
const primuButton = document.querySelector('#form-inregistrare
button[type="submit"]');

// Toate elementele care se potrivesc - returnează NodeList
const toateCelulele = document.querySelectorAll('td');
const toateCardurile = document.querySelectorAll('.card-speaker');

// Iterare NodeList
toateCardurile.forEach(card => {
  card.classList.add('animat');
```

```
});

// Selector clasic (ușor mai rapid pentru ID-uri frecvent accesate)
const container = document.getElementById('speakeri');
```

Modificarea elementelor

```
const element = document.querySelector('.mesaj');

// Conținut text - SIGUR (nu parsează HTML)
element.textContent = 'Înregistrare reușită!';

// Conținut HTML - ATENȚIE la XSS dacă vine de la utilizator!
element.innerHTML = '<strong>Succes!</strong> Vei primi email de confirmare.';

// Atribute
element.setAttribute("aria-live", "polite");
element.getAttribute("class");
element.removeAttribute("hidden");

// Clase CSS
element.classList.add('succes', 'animat');
element.classList.remove('eroare');
element.classList.toggle('activ');        // adaugă dacă nu există, elimină dacă
există
element.classList.contains('succes');     // true/false

// Stiluri inline (rar necesar - preferă classList)
element.style.display = 'block';
element.style.backgroundColor = '#d4edda';
```

Crearea și inserarea elementelor

```
function creeazaMesajEroare(text) {
  const span = document.createElement('span');
  span.textContent = text;
  span.className = 'mesaj-eroare';
  span.setAttribute('role', 'alert');
  span.setAttribute('aria-live', 'assertive');
  return span;
}

const input = document.querySelector('#email');
const eroare = creeazaMesajEroare('Email invalid');

// Inserare după input
input.after(eroare);
// Sau: input.insertAdjacentElement('afterend', eroare);

// Inserare la finalul unui container
const container = document.querySelector('.erori-container');
container.appendChild(eroare);

// Ștergere
eroare.remove();
// Sau: eroare.parentNode.removeChild(eroare);
```

4.6 Evenimente

Evenimentele permit JavaScript să reacționeze la acțiunile utilizatorului și ale browserului.

```
// Pattern standard: addEventListener
const buton = document.querySelector('#btn-submit');

buton.addEventListener('click', function(event) {
  console.log('Click!', event.target);
  event.preventDefault();  // previne comportamentul implicit
});

// Arrow function (mai concis)
buton.addEventListener('click', (e) => {
  e.preventDefault();
  handleSubmit();
});

// Stocarea referinței pentru a putea elimina event listener-ul
function handleClick(e) { console.log('clicked'); }
buton.addEventListener('click', handleClick);
buton.removeEventListener('click', handleClick);
```

Tipuri de evenimente esențiale

Eveniment	Când se declanșează și proprietăți utile
click	Clic stânga pe element. e.target = elementul click-at. e.clientX, e.clientY = coordonate.
submit	Trimiterea formularului. Apelează e.preventDefault() pentru a preveni reîncărcarea paginii!
input	La fiecare tastă apăsată sau modificare valorii unui input. e.target.value = valoarea curentă.
change	Când valoarea se schimbă ȘI input-ul pierde focusul. Ideal pentru select, checkbox, radio.
keydown / keyup	Apăsare/eliberare tastă. e.key = 'Enter', 'Escape', 'ArrowUp', etc.
focus / blur	Câștigarea/pierderea focusului de un input. Ideal pentru validare.
mouseover / mouseout	Cursor intrat/ieșit din element.
DOMContentLoaded	DOM-ul a fost parsat complet (fără a aștepta imagini). Rulează codul de inițializare aici.
load	Pagina completă s-a încărcat, inclusiv imagini și resurse externe.

Event Delegation - pattern eficient

În loc să adaugi event listener pe fiecare element al unei liste, adaugă unul singur pe container:

```
// ✗ Ineficient: un listener per card
document.querySelectorAll('.card-speaker').forEach(card => {
  card.addEventListener('click', handleCardClick);
});

// ☑ Eficient: un singur listener pe container
const gridSpeakeri = document.querySelector('.grid-speakeri');
gridSpeakeri.addEventListener('click', (e) => {
  const card = e.target.closest('.card-speaker');
```

```
    if (!card) return;  // click nu a fost pe un card
    const speakerId = card.dataset.id;
    afiseazaDetaliiSpeaker(speakerId);
  });
  // Avantaj: funcționează și pentru elemente adăugate dinamic!
```

4.7 LocalStorage și SessionStorage

Browser Storage permite salvarea datelor direct în browser, fără server. Util pentru: preferințe utilizator, form progress, cache date, stare aplicație.

```
  // localStorage - persistă între sesiuni (rămâne după închiderea tab-ului)
  // sessionStorage - se șterge la închiderea tab-ului

  // SCRIERE - valori simple
  localStorage.setItem('tema', 'dark');
  localStorage.setItem('limba', 'ro');

  // CITIRE
  const tema = localStorage.getItem('tema'); // 'dark'
  const absent = localStorage.getItem('cheie-inexistenta'); // null

  // ȘTERGERE
  localStorage.removeItem('tema');
  localStorage.clear(); // șterge TOT

  // Stocarea obiectelor (trebuie serializate JSON)
  const dateFormular = {
    nume: 'Ana Popescu',
    email: 'ana@exemplu.ro',
    workshop: 'react-avansat'
  };
  localStorage.setItem('form-devconnect', JSON.stringify(dateFormular));

  // Citire și parsare
  const salvat = localStorage.getItem('form-devconnect');
  const date = salvat ? JSON.parse(salvat) : null;

  // Pattern sigur cu try-catch (JSON.parse poate arunca erori)
  function citesteDinStorage(cheie) {
    try {
      const valoare = localStorage.getItem(cheie);
      return valoare ? JSON.parse(valoare) : null;
    } catch (e) {
      console.warn('Eroare citire storage:', e);
      return null;
    }
  }
```

> Limitări localStorage: max ~5MB, sincron (blochează thread-ul pentru date mari), doar string-uri (necesită JSON.stringify/parse). NU stoca parole, tokeni de autentificare sau date sensibile în localStorage - sunt accesibile prin JavaScript, deci vulnerabile la XSS.

4.8 Exemplu Practic Complet în VS Code - DevConnect: JavaScript Interactiv

Adaugă folderul js/ în proiectul DevConnect cu fișierele descrise în Proiect-Model Lab 4 din Volumul I. Urmărește structura și codul de acolo.

Debugging JavaScript în VS Code

VS Code are debugger integrat pentru JavaScript. Configurare pentru browser:

1. Deschide Command Palette (Ctrl+Shift+P)
2. Tastează 'Open launch.json' și selectează Chrome ca browser
3. VS Code creează .vscode/launch.json
4. Setează breakpoint-uri (clic pe numărul liniei - apare un punct roșu)
5. Apasă F5 pentru a porni debugging-ul
6. Utilizează F10 (step over), F11 (step into), F5 (continue)

Alternativ: folosește extensia 'Live Server' + DevTools din browser (F12 → Sources). Ambele metode sunt valide - DevTools din browser este mai rapid pentru debugging rapid.

Verifică implementarea în browser

7. Deschide pagina DevConnect în browser (Live Server)
8. Deschide Console (F12 → Console). Adaugă console.log("Script încărcat") la începutul app.js
9. Reîncarcă pagina. Dacă vezi mesajul în consolă, scriptul este conectat corect
10. Testează validarea: completează câmpul email cu o adresă invalidă și observă mesajul de eroare
11. Testează localStorage: completează parțial formularul, închide și redeschide tab-ul - datele ar trebui restaurate

La finalul Cursurilor 1–4 și Laboratoarelor 1–4 ai construit o pagină web profesională, semantică, stilizată și interactivă. Proiectul DevConnect are deja: HTML5 valid și semantic, CSS responsiv cu design system, JavaScript cu validare live, date dinamice și persistență localStorage.

Referințe pentru Cursul 4

★ javascript.info - Cel mai complet tutorial JavaScript modern (gratuit)
★ MDN JavaScript Guide - documentație oficială
Eloquent JavaScript, 3rd ed. (carte gratuită online)
JS Visualizer - vizualizare execuție și event loop
freeCodeCamp - JavaScript Algorithms and Data Structures
You Don't Know JS (serie gratuită pe GitHub)

Notă privind elaborarea materialelor de curs

Vreau să fiu transparent cu voi: structura și conținutul acestor note de curs au fost generate cu ajutorul unui instrument de inteligență artificială (Claude, de la Anthropic), pe baza cerințelor și direcțiilor pe care le-am formulat eu ca titular de curs.

De ce vă spun asta? Pentru că:
• Nu pot garanta că fiecare noțiune tehnică are 100% acuratete sau este actualizată
• Vă încurajez să verificați activ sursele bibliografice indicate
• Utilizarea responsabilă a AI în educație înseamnă transparență, nu ascundere

Considerați aceste materiale un ghid structurat de studiu, nu un manual definitiv. Dacă identificați o eroare sau o neclaritate, veniți cu ea la curs