

JavaScript de la zero

Caiet de proiect și laborator introductiv

Revizie parțială, 20 octombrie 2026. Prima lecție este aprofundată; lecțiile 2-8 păstrează ediția anterioară. Proiectele sunt fictive. Lectura și practica au durate distincte.

Cum folosești documentul

Fișa anterioară este păstrată integral după această copertă. Suplimentul final conține lecția introductivă nouă, codul complet, experimentele și reperele de rezolvare. Numerotările originale ale secțiunilor sunt independente. Pentru prima lecție folosește suplimentul. Caietul complet va fi consolidat după aprofundarea tuturor lecțiilor.

Calculator interactiv de buget

Fișă publică de proiect · exerciții și criterii de acceptare

Construiești un calculator cu trei categorii de cost, validare și stări clare. Separi logica matematică de DOM pentru a putea verifica rezultatele.

Livrabile

- Funcție de calcul
- Interfață cu formular
- Teste manuale
- Mesaje pentru erori

Criterii de acceptare

- Valorile invalide sunt respinse
- Zero produce rezultat corect
- Nu apare HTML interpretat din input

Script încărcat după parsarea documentului

JavaScriptul din browser interacționează cu DOM-ul și cu API-urile disponibile în acel mediu. Același limbaj poate rula pe server, dar nu toate obiectele sunt comune. Pentru laborator, folosește un script extern cu defer, astfel încât documentul să fie parsat înaintea execuției. Separarea fișierelor face mai ușor de observat o eroare de încărcare sau de sintaxă. Consola trebuie verificată la fiecare pas, nu doar când pagina devine complet nefuncțională.

Exemplu de lucru

```
index.html:  
<p id="stare">Neinițializat</p>  
<script src="app.js" defer></script>
```

```
app.js:  
const stare = document.querySelector('#stare');  
if (stare) {  
  stare.textContent = 'Script încărcat';  
}  
console.log('Laborator browser');
```

Interpretarea rezultatului

Textul se schimbă după încărcarea scriptului, iar consola arată mesajul. Dacă app.js răspunde cu 404, codul nu ajunge să ruleze. Dacă selectorul nu găsește elementul, verificarea if evită accesarea unei proprietăți pe null, dar nu explică de ce elementul lipsește.

Exercițiu de consolidare

Schimbă deliberat numele fișierului în HTML și observă separat eroarea Network și lipsa mesajului din consolă.

1. Unde rulează JavaScript

Schimbă deliberat numele fișierului în HTML și observă separat eroarea Network și lipsa mesajului din consolă.

Verificare orientativă

Textul se schimbă după încărcarea scriptului, iar consola arată mesajul. Dacă app.js răspunde cu 404, codul nu ajunge să ruleze. Dacă selectorul nu găsește elementul, verificarea if evită accesarea unei proprietăți pe null, dar nu explică de ce elementul lipsește.

De evitat

JavaScript este limbajul; API-urile depind de mediul de execuție.

Conversie explicită înaintea calculelor

Valorile unui formular vin de obicei ca șiruri. Operatorul plus poate concatena, de aceea conversia trebuie făcută înaintea calculului și verificată. Un șir gol nu trebuie acceptat accidental ca zero dacă regula cere o valoare. Folosește comparații stricte pentru a evita egalități surprinzătoare între tipuri. Pentru bani, stabilește o strategie de reprezentare și rotunjire; exemplul de mai jos este un calcul didactic simplu, nu un motor fiscal.

Exemplu de lucru

```
const text = '12.50';
const pret = Number(text);
if (!Number.isFinite(pret)) throw new Error('Preț invalid');
const cantitate = 3;
const total = pret * cantitate;
console.log(total.toFixed(2)); // 37.50
console.log('12' + 3);        // 123
console.log(Number('12') + 3); // 15
```

Interpretarea rezultatului

toFixed întoarce un șir pentru afișare, nu un număr cu precizie monetară specială. Păstrează separată valoarea de calcul de cea formatată. Testează gol, text nevalid, zero și număr negativ conform regulii produsului, fără a accepta tot ce poate converti Number.

Exercițiu de consolidare

Adaugă o validare care respinge șirul gol înainte de conversie și o cantitate care nu este întreg pozitiv.

2. Variabile, tipuri, operatori

Adaugă o validare care respinge șirul gol înainte de conversie și o cantitate care nu este întreg pozitiv.

Verificare orientativă

toFixed întoarce un șir pentru afișare, nu un număr cu precizie monetară specială. Păstrează separată valoarea de calcul de cea formatată. Testează gol, text nevalid, zero și număr negativ conform regulii produsului, fără a accepta tot ce poate converti Number.

De evitat

Input-urile HTML livrează în mod obișnuit text.

Funcție pură și domeniu de vizibilitate

O funcție care primește toate intrările și întoarce rezultatul fără a modifica exteriorul este mai simplu de testat. Variabilele declarate în bloc cu `const` sau `let` nu trebuie presupuse disponibile în afara lui. Nu folosi o variabilă globală pentru total dacă acesta poate fi returnat. Separă calculul de actualizarea interfeței; astfel aceeași funcție poate fi verificată fără browser.

Exemplu de lucru

```
function subtotal(cantitate, pretBani) {
  if (!Number.isInteger(cantitate)) throw new Error('Cantitate invalidă');
  if (cantitate < 0) throw new Error('Cantitate negativă');
  if (!Number.isInteger(pretBani)) throw new Error('Preț invalid');
  if (pretBani < 0) throw new Error('Preț negativ');
  return cantitate * pretBani;
}
console.assert(subtotal(3, 1250) === 3750);
console.assert(subtotal(0, 1250) === 0);
```

Interpretarea rezultatului

Funcția întoarce bani întregi și nu atinge DOM-ul. Verificarea cu zero arată un caz de frontieră; dacă produsul nu permite cantitate zero, regula se schimbă explicit. Pentru cantități foarte mari, trebuie verificată și limita numerică sigură a JavaScriptului.

Exercițiu de consolidare

Apelează funcția cu 1.5 și explică de ce această intrare este respinsă în contractul ales.

3. Funcții și scope

Apelează funcția cu 1.5 și explică de ce această intrare este respinsă în contractul ales.

Verificare orientativă

Funcția întoarce bani întregi și nu atinge DOM-ul. Verificarea cu zero arată un caz de frontieră; dacă produsul nu permite cantitate zero, regula se schimbă explicit. Pentru cantități foarte mari, trebuie verificată și limita numerică sigură a JavaScriptului.

De evitat

O funcție care citește variabile globale e mai greu de testat.

Parcurgere care păstrează identitatea elementelor

Pentru un coș demonstrativ, fiecare rând are identificator, cantitate și preț în bani. Calculează subtotalul rândului și apoi totalul, fără a pierde legătura cu produsul. O buclă simplă este suficientă când vrei să urmărești fiecare pas. Nu modifica array-ul în timpul parcurgerii decât dacă ai proiectat explicit acest comportament. Cazul fără produse trebuie tratat ca rezultat valid sau eroare, potrivit contractului.

Exemplu de lucru

```
const cos = [
  { id: 'A', qty: 2, price: 1250 },
  { id: 'B', qty: 1, price: 800 }
];
let total = 0;
for (const item of cos) {
  const linie = item.qty * item.price;
  total += linie;
  console.log(item.id, linie);
}
console.assert(total === 3300);
```

Interpretarea rezultatului

Produsul A contribuie cu 2.500 bani, B cu 800, iar totalul este 3.300 bani. Dacă rândurile sunt duplicate, bucla le va număra pe ambele; deduplicarea este o regulă separată, nu o proprietate a parcurgerii. Testează coșul gol înainte de a formata rezultatul.

Exercițiu de consolidare

Adaugă un produs cu cantitate zero și decide dacă îl excluzi la validare sau îl păstrezi cu subtotal zero.

4. Condiții, bucle, array-uri

Adaugă un produs cu cantitate zero și decide dacă îl excluzi la validare sau îl păstrezi cu subtotal zero.

Verificare orientativă

Produsul A contribuie cu 2.500 bani, B cu 800, iar totalul este 3.300 bani. Dacă rândurile sunt duplicate, bucla le va număra pe ambele; deduplicarea este o regulă separată, nu o proprietate a parcurgerii. Testează coșul gol înainte de a formata rezultatul.

De evitat

reduce fără valoare inițială poate eșua pe array gol.

DOM actualizat fără interpretarea textului ca HTML

Când afișezi un mesaj primit de la utilizator, folosește `textContent` pentru text simplu. `innerHTML` interpretează markup și poate introduce vulnerabilități dacă datele nu sunt controlate. Pentru o listă, creează elementele și atribuie textul separat. Păstrează mesajele de stare într-un element potrivit și nu șterge tot containerul paginii pentru a actualiza o valoare. Selectoarele trebuie să fie suficient de precise și verificate înaintea folosirii.

Exemplu de lucru

```
const lista = document.querySelector('#rezultate');
const mesaje = ['Cerere primită', '<img src=x onerror=alert(1)>'];
if (lista) {
  lista.replaceChildren();
  for (const mesaj of mesaje) {
    const li = document.createElement('li');
    li.textContent = mesaj;
    lista.append(li);
  }
}
```

Interpretarea rezultatului

Al doilea mesaj este afișat ca text, nu executat ca imagine. Acesta este un test util pentru diferența dintre text și markup, nu o suită completă de securitate. Dacă aplicația are nevoie de HTML bogat, proiectează o politică de sanitizare separată și verificabilă.

Exercițiu de consolidare

Adaugă un mesaj cu diacritice și ghilimele. Confirmă că apare integral fără a fi necesară construirea unui șir HTML.

5. DOM: selectare și modificare

Adaugă un mesaj cu diacritice și ghilimele. Confirmă că apare integral fără a fi necesară construirea unui șir HTML.

Verificare orientativă

Al doilea mesaj este afișat ca text, nu executat ca imagine. Acesta este un test util pentru diferența dintre text și markup, nu o suită completă de securitate. Dacă aplicația are nevoie de HTML bogat, proiectează o politică de sanitizare separată și verificabilă.

De evitat

HTML necontrolat poate introduce vulnerabilități XSS.

Eveniment de formular cu prevenirea trimiterii implicite

Ascultă evenimentul submit al formularului, nu numai click-ul butonului; utilizatorul poate trimite și cu Enter. În exemplul local, preventDefault oprește navigarea fiindcă nu există backend. Citește datele, validează-le și actualizează starea. Într-o aplicație reală, validarea din browser este pentru experiență; serverul trebuie să repete verificările. Nu considera un buton dezactivat drept autorizare sau protecție împotriva cererilor directe.

Exemplu de lucru

```
const form = document.querySelector('#cerere');
const stare = document.querySelector('#stare');
form.addEventListener('submit', (event) => {
  event.preventDefault();
  const data = new FormData(form);
  const nume = String(data.get('nume') ?? '').trim();
  if (!nume) { stare.textContent = 'Completează numele fictiv.'; return; }
  stare.textContent = `Date pregătite pentru ${nume}.`;
});
```

Interpretarea rezultatului

Fragmentul cere elemente existente cu id-urile cerere și stare și un câmp name=nume. Mesajul spune „pregătite”, nu „trimise”, deoarece nu există apel de rețea. Această diferență previne confirmări false în prototipuri.

Exercițiu de consolidare

Testează trimiterea prin click și Enter, apoi un nume format doar din spații. Rezultatul trebuie să fie consecvent.

6. Evenimente

Testează trimiterea prin click și Enter, apoi un nume format doar din spații. Rezultatul trebuie să fie consecvent.

Verificare orientativă

Fragmentul cere elemente existente cu id-urile cerere și stare și un câmp name=nume. Mesajul spune „pregătite”, nu „trimise”, deoarece nu există apel de rețea. Această diferență previne confirmări false în prototipuri.

De evitat

Un handler pe buton poate rata trimiterea prin tastatură.

Fetch cu verificarea statusului și a formei datelor

fetch poate întoarce un răspuns HTTP de eroare fără a respinge promisiunea. Verifică response.ok înaintea parsării și validează apoi structura datelor. În laborator, folosește un fișier local JSON pentru a evita dependența de un API public instabil. Separă starea de încărcare de eroare și rezultat gol. Un array gol este un rezultat posibil, nu automat o eroare de rețea.

Exemplu de lucru

```
async function incarca() {
  const response = await fetch('/produse.json');
  if (!response.ok) throw new Error(`HTTP ${response.status}`);
  const data = await response.json();
  if (!Array.isArray(data)) throw new Error('Format neașteptat');
  return data;
}
incarca().then(console.log).catch(console.error);
// produse.json: [{"id":1,"name":"Produs demo"}]
```

Interpretarea rezultatului

Rulează prin serverul local al laboratorului. Un fișier lipsă produce eroarea HTTP, iar un obiect în locul array-ului produce eroarea de format. Acestea cer remedieri diferite. Pentru obiecte din array, adaugă și validarea câmpurilor înainte de afișare.

Exercițiu de consolidare

Schimbă fișierul în JSON invalid și observă eroarea de parsare. Nu o afișa utilizatorului ca „nu există produse”.

7. Fetch și date de la API

Schimbă fișierul în JSON invalid și observă eroarea de parsare. Nu o afișa utilizatorului ca „nu există produse”.

Verificare orientativă

Rulează prin serverul local al laboratorului. Un fișier lipsă produce eroarea HTTP, iar un obiect în locul array-ului produce eroarea de format. Acestea cer remedieri diferite. Pentru obiecte din array, adaugă și validarea câmpurilor înainte de afișare.

De evitat

fetch nu respinge automat promisiunea pentru orice status HTTP 4xx sau 5xx.

Mini-proiect complet: filtrarea unei liste locale

Construiește o pagină care filtrează produse fictive după nume și arată numărul rezultatelor. Datele sunt locale, deci proiectul poate fi verificat fără cont sau serviciu extern. Păstrează funcția de filtrare separată de randare și folosește `textContent`. Un rezultat gol trebuie să aibă mesaj clar. Pentru un catalog real, filtrarea și prețurile trebuie reconciliate cu datele serverului; acest exemplu nu este un coș de cumpărături.

Exemplu de lucru

```
HTML: <label for="q">Caută</label><input id="q"><p id="numar" aria-live="polite"></p><ul
id="lista"></ul>
JS:
const produse = ['Cască', 'Lanț', 'Pompă'];
const q = document.querySelector('#q');
const lista = document.querySelector('#lista');
function render() {
  const gasite = produse.filter(x =>
x.toLocaleLowerCase('ro').includes(q.value.trim().toLocaleLowerCase('ro')));
  lista.replaceChildren(...gasite.map(x => { const li = document.createElement('li');
li.textContent = x; return li; }));
  document.querySelector('#numar').textContent = `${gasite.length} rezultate`;
}
q.addEventListener('input', render);
render();
```

Interpretarea rezultatului

La încărcare apar trei produse; „pom” păstrează unul; un șir fără potrivire produce zero rezultate. Încarcă scriptul cu `defer` sau ca modul. Testează șirul gol, spațiile și literele mari. Căutarea fără diacritice nu este implementată implicit; nu pretinde că „casca” găsește „Cască”.

Exercițiu de consolidare

Adaugă normalizare pentru căutarea fără diacritice într-o funcție separată și testează dacă nu schimbă textul afișat.

8. Un mini-proiect interactiv

Adaugă normalizare pentru căutarea fără diacritice într-o funcție separată și testează dacă nu schimbă textul afișat.

Verificare orientativă

La încărcare apar trei produse; „pom” păstrează unul; un șir fără potrivire produce zero rezultate. Încarcă scriptul cu `defer` sau ca modul. Testează șirul gol, spațiile și literele mari. Căutarea fără diacritice nu este implementată implicit; nu pretinde că „casca” găsește „Cască”.

De evitat

Nu salva date financiare reale în exercițiul public.

Atelier final: Calculator cu intrări controlate

Timp de lucru estimat: 40 de minute. Folosește datele de laborator de mai jos și salvează rezultatul separat de orice proiect real.

Date de pornire

Categorii fictive: design 500 lei, implementare 1.200 lei, verificare 300 lei. Set de test: valorile inițiale, toate zero, un câmp gol, un număr negativ și text în locul unui număr.

1. Pregătire — 5 minute

Extrage intrările și condițiile din enunț. Notează ce rezultat trebuie produs și ce informații nu sunt confirmate. Pregătește fișierele sau tabelul în care vei păstra dovada lucrului.

2. Construire — 15 minute

Separă funcția de total de evenimentele DOM. Afișează rezultatul și erorile folosind `textContent`. Scrie intrarea și rezultatul așteptat pentru fiecare test.

3. Variație și verificare — 15 minute

Aduagă o categorie nouă și repetă testele. Identifică dacă modificarea cere schimbarea funcției de calcul sau doar a datelor și interfeței.

4. Autoevaluare — 5 minute

Compară rezultatul cu reperele de mai jos. Înregistrează cel puțin o eroare identificată sau un caz-limită verificat, plus corectura ori explicația comportamentului observat.

Repere de rezolvare

Totalul inițial este 2.000 lei; zero rămâne valid. Câmpurile goale sau invalide urmează o regulă explicită și nu produc NaN în interfață. Textul introdus nu se execută ca HTML.

Ce predai

- Funcție de calcul
- Interfață cu formular
- Teste manuale
- Mesaje pentru erori

Criterii de acceptare

- Valorile invalide sunt respinse
- Zero produce rezultat corect
- Nu apare HTML interpretat din input

Păstrează o versiune inițială și una revizuită, cu explicația diferențelor. Dacă un criteriu nu este îndeplinit, revino la lecția relevantă și repetă verificarea. Durata este orientativă; instalarea mediului și aprofundarea pot necesita timp suplimentar.

Registrul de lucru

Pentru fiecare etapă notează intrarea, rezultatul, dovada verificării, eroarea găsită și corectura. Nu introduce date personale sau secrete în fișa partajată.

Documentație și surse

- JavaScript Guide — MDN

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide>

Laborator: Unde rulează JavaScript

Lectură estimată: 14 minute

Estimare pentru text, la un ritm de 180 de cuvinte pe minut. Timpul pentru analizarea codului, exerciții și proiecte se adaugă separat și depinde de ritmul tău.

Ce construim în această lecție

În această lecție pregătești baza unui calculator de buget pentru un proiect fictiv. Nu introducem încă formularul și nici validarea sumelor. Construim mai întâi o pagină care poate încărca un fișier JavaScript, modifica un mesaj și explica prin consolă ce s-a întâmplat. Dacă această fundație nu funcționează, orice funcție de calcul adăugată ulterior va fi greu de verificat. Scopul este să poți urmări traseul dintre fișierul salvat, browser și rezultatul observat.

La final vei avea trei fișiere: `index.html` pentru structură, `styles.css` pentru prezentare și `app.js` pentru comportament. Vei porni pagina printr-un server local, vei verifica răspunsurile HTTP și vei provoca deliberat câteva erori într-o copie a laboratorului. Fiecare eroare are un simptom, o explicație și o corectură. Nu lucrăm într-un proiect de producție și nu introducem informații personale.

Păstrează un registru cu patru coloane: schimbarea făcută, rezultatul așteptat, rezultatul observat și explicația. Prima dată scrie ce crezi că se va întâmpla, apoi rulează experimentul. Această ordine te obligă să formulezi o ipoteză. Dacă scrii numai după ce ai văzut rezultatul, riști să confunzi recunoașterea unui mesaj cu înțelegerea cauzei lui.

Limbaajul și mediul în care îl execuți

JavaScript este un limbaj de programare. Browserul este unul dintre mediile care îl execută și îi pune la dispoziție obiecte pentru lucrul cu pagina. Node.js este un alt mediu, folosit inclusiv pentru programe în terminal și aplicații server. Lista nu se reduce la aceste două exemple. Pentru început ne interesează să separăm regulile limbajului de facilitățile oferite de mediul ales.

Un calcul precum $500 + 1200 + 300$ nu are nevoie de o pagină. În schimb, `document.querySelector` folosește documentul expus de mediul browserului. Dacă același fișier este executat direct în Node.js, nu primește automat pagina deschisă într-o altă aplicație. Numele fișierului poate fi identic, dar contextul de execuție este diferit. Când raportezi o eroare, spune întotdeauna unde ai rulat codul.

Imaginează-ți că ai o rețetă și două bucătării. Instrucțiunile despre ordine și cantități rămân aceleași, însă echipamentele disponibile diferă. Analogia are o limită: programele nu improvizează când un obiect lipsește, ci urmează regulile limbajului și produc un rezultat sau o eroare. De aceea nu este suficient să spui „codul este JavaScript”; trebuie să precizezi și mediul necesar.

HTML, CSS și JavaScript au responsabilități distincte

HTML descrie structura documentului: titlul, paragrafele și zonele de conținut. CSS controlează prezentarea. JavaScript adaugă comportamentul pe care îl programăm. Nu orice pagină are nevoie de JavaScript pentru a fi utilă. În laborator, explicația proiectului rămâne vizibilă chiar dacă scriptul nu se încarcă. Numai mesajul de stare depinde de execuția lui.

Această separare face diagnosticul mai clar. Dacă titlul lipsește complet, verifici documentul și adresa paginii. Dacă textul există, dar aspectul nu corespunde, investighezi stilurile. Dacă prezentarea este corectă, însă mesajul nu se schimbă, urmărești scriptul. Pot exista probleme care combină mai multe straturi, dar această

ordine îți oferă un punct de pornire.

Nu vom transforma textul „script neinițializat” într-un mesaj de succes doar pentru că arată mai bine. Starea inițială trebuie să descrie sincer ce știm înainte de execuție. Ulterior, aplicațiile au stări de încărcare, succes și eroare. Aici exersăm aceeași disciplină într-un exemplu mic: afișăm confirmarea numai după ce codul ajunge efectiv la operațiunea respectivă.

Pregătirea directorului de lucru

Creează un director nou numit `laborator-js-01`. În el salvează cele trei fișiere ale lecției cu un editor de text simplu sau un editor de cod. Verifică extensiile reale: `app.js.txt` nu este același nume cu `app.js`. Activarea afișării extensiilor în exploratorul de fișiere poate clarifica această greșeală pe Windows.

Fișierele se salvează în UTF-8, pentru ca diacriticele să fie păstrate. Nu copia marcasele care delimitează blocurile de cod din lecție. De exemplu, linia cu numele fișierului este o indicație pentru tine, nu o instrucțiune JavaScript. Începe prin salvarea tuturor fișierelor înainte de a deschide pagina; o filă deschisă în editor nu înseamnă că informația există deja pe disc.

În această etapă nu ai nevoie de un framework, un cont de găzduire sau de un proces de compilare. Mai puține componente înseamnă mai puține surse posibile ale erorilor. Vom adăuga complexitate când apare o nevoie pe care o putem explica, nu pentru că un proiect demonstrativ trebuie să semene de la început cu o aplicație mare.

Fișierul `index.html`

Documentul de mai jos are un singur titlu principal și un paragraf identificat prin id. Identificatorul `stare` este legătura dintre structura HTML și codul care o va modifica. Stilurile și scriptul sunt fișiere distincte, aflate în același director. Atributul `lang` indică limba documentului, iar `viewport` permite o prezentare potrivită pe ecrane înguste.

```
<!doctype html>
<html lang="ro">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Primul laborator JavaScript</title>
  <link rel="stylesheet" href="styles.css">
  <script src="app.js" defer></script>
</head>
<body>
  <main>
    <h1>Calculator de buget: pregătirea</h1>
    <p>Proiect fictiv. În această lecție verificăm încărcarea scriptului.</p>
    <p id="stare">Script neinițializat.</p>
    <noscript>JavaScript este dezactivat în acest browser.</noscript>
  </main>
</body>
</html>
```

Textul din `noscript` ajută când executarea scripturilor este dezactivată. El nu apare automat pentru orice eroare din `app.js`. Un fișier lipsă sau o greșeală de sintaxă nu este același lucru cu dezactivarea JavaScript. Păstrează această distincție când verifici o pagină: mecanismele de rezervă trebuie proiectate pentru condițiile pe care le pot detecta.

Fișierul styles.css

Stilurile sunt intenționat puține. Limităm lățimea textului, oferim spațiere și permitem împărțirea șirurilor lungi. Nu adăugăm animații sau elemente decorative care ar îngreuna observarea mesajului. Pe parcursul exercițiilor, aspectul paginii trebuie să rămână suficient de constant încât schimbarea textului să fie ușor de văzut.

```
* { box-sizing: border-box; }
body {
  margin: 0;
  padding: 1rem;
  font: 1rem/1.6 system-ui, sans-serif;
  color: #202838;
  background: #f4f6fb;
}
main {
  max-width: 46rem;
  margin: 2rem auto;
  padding: 1.5rem;
  background: white;
  border: 1px solid #ccd3df;
  border-radius: 0.75rem;
}
h1 { font-size: 1.8rem; line-height: 1.2; }
p { overflow-wrap: anywhere; }
#stare { padding: 1rem; background: #edf2ff; }
```

Salvează și acest fișier înainte de test. Dacă aspectul nu se aplică, nu modifica imediat JavaScriptul. Verifică mai întâi calea styles.css și răspunsul resursei. O eroare CSS poate face pagina mai puțin plăcută, dar nu demonstrează că scriptul nu rulează. În laborator vom observa separat încărcarea fiecărui fișier.

Fișierul app.js, explicat pe rând

Codul pornește cu un mesaj în consolă. Apoi caută elementul și verifică dacă există. Dacă lipsește, aruncă o eroare cu o explicație precisă. Dacă există, îi schimbă textul și confirmă în consolă finalizarea. Folosim `const` pentru referința la element; lecția următoare dezvoltă variabilele și tipurile.

```
console.log("Pornire laborator JS.");
const stare = document.querySelector("#stare");
if (stare === null) {
  throw new Error("Lipsește elementul #stare din document.");
}
stare.textContent = "Script încărcat. Laborator pregătit.";
console.log("Inițializare încheiată.");
```

`querySelector` întoarce primul element care corespunde selectorului sau `null` dacă nu găsește unul. Semnul `#` indică un identificator CSS. Nu căutăm textul paragrafului și nu căutăm numele unei variabile din editor. Căutăm în structura documentului pe care browserul îl are disponibil în acel moment.

Verificarea explicită este utilă aici deoarece elementul este obligatoriu pentru exercițiu. Dacă am scrie doar „dacă există, modifică-l”, pagina ar putea rămâne în starea inițială fără o explicație clară. În alte componente, absența unui element poate fi acceptabilă. Alegerea trebuie să reflecte contractul componente, nu preferința de a ascunde erorile.

`textContent` introduce text. Nu interpretează șirul ca un fragment HTML. Dacă mesajul ar conține caractere precum `<` și `>`, ele ar rămâne text vizibil. În această lecție nu avem input de utilizator, dar alegerea pregătește un obicei util pentru calculatorul de buget: afișăm valori și mesaje fără să le transformăm inutil în markup.

De ce folosim defer

Pentru acest script clasic extern, defer permite descărcarea în timpul parsării HTML și amână execuția până după parsarea documentului. Nu înseamnă că toate imaginile și toate resursele paginii s-au terminat de încărcat. Ordinea ne permite să căutăm paragraful din body chiar dacă elementul script se află în head.

Fără defer, un script clasic extern aflat aici poate rula înainte ca parserul să fi ajuns la paragraful nostru. Nu orice script fără defer este greșit: un calcul care nu folosește elementele respective are alte condiții. Problema concretă este dependența dintre cod și existența elementului pe care îl caută.

Nu înlocui defer cu async presupunând că sunt sinonime. async nu oferă aceeași garanție de execuție după parsarea documentului. Scripturile module au propriul comportament; le vom aborda separat. Pentru laborator păstrează exact forma externă cu defer și investighează doar variațiile cerute, într-o copie.

Pornește laboratorul prin HTTP local

Dacă ai Python 3 disponibil, deschide terminalul în directorul laborator-js-01 și pornește un server local. Pe Windows, într-o instalare care oferă launcherul py, poți folosi comanda de mai jos. Pe Linux sau macOS folosește python3 în loc de py -3. Dacă Python lipsește, utilizează un server de dezvoltare de încredere deja instalat, cu aceeași rădăcină de documente.

```
py -3 -m http.server 18771 --bind 127.0.0.1
```

Deschide `http://127.0.0.1:18771/index.html` în browser. Terminalul serverului rămâne deschis. Portul este ales pentru laborator, nu are o semnificație specială. Dacă este ocupat, alege altul și modifică adresa în consecință. Serverul demonstrativ nu este o soluție de publicare în producție.

Acest exemplu cu script clasic poate funcționa și prin deschiderea fișierului local, dar folosim HTTP ca să putem observa clar răspunsurile resurselor. Nu generaliza rezultatul la module, fetch sau orice alt API. Un exercițiu care merge prin file nu demonstrează că toate etapele viitoare vor avea același comportament.

Ce verifici în browser

Deschide instrumentele pentru dezvoltatori din meniul browserului. În Network, reîncarcă pagina și identifică `index.html`, `styles.css` și `app.js`. Pentru fiecare verifică adresa, statusul și conținutul răspunsului. Dacă serverul întoarce o pagină HTML în locul unui script, un status 200 nu este suficient pentru a declara resursa corectă.

În Console urmărește cele două mesaje din cod. În pagină trebuie să apară „Script încărcat. Laborator pregătit.”. Aceste trei observații răspund la întrebări diferite: fișierul a fost cerut, codul a ajuns la punctele marcate și documentul a fost modificat. Păstrează diferența dintre ele în notițe.

Pentru o verificare suplimentară, deschide fișierul `app.js` în panoul Sources și pune un breakpoint pe linia care atribuie `textContent`. Reîncarcă pagina, inspectează valoarea stare și continuă execuția. Un breakpoint îți permite să observi starea programului într-un punct precis; un `console.log` îți oferă o urmă textuală. Nu trebuie să alegi unul pentru totdeauna.

Experimentul 1: fișierul nu este găsit

Într-o copie a laboratorului, schimbă `src` din `app.js` în `app-lipsa.js`, fără să creezi acel fișier. Reîncarcă pagina. Textul ar trebui să rămână în starea inițială, iar Network să arate cererea nereușită. Mesajele definite în `app.js` nu mai sunt dovezi noi ale acestei încălcări; golește consola sau delimitează clar încercările.

Rezolvarea este să restabilești calea corectă, nu să adaugi `setTimeout` în JavaScript. Codul care nu a fost încărcat nu poate fi reparat printr-o instrucțiune din același fișier. Scrie în registru diferența dintre „fișierul lipsește” și „fișierul există, dar conține o eroare”. Sunt etape distincte ale traseului.

Experimentul 2: sintaxă invalidă

Restabilește mai întâi laboratorul funcțional. Apoi elimină deliberat ghilimeaua de închidere a primului mesaj din app.js. Salvează și reîncarcă. Fișierul poate fi descărcat cu succes, dar parserul JavaScript nu îl poate interpreta. În această situație, nici mesajul de la începutul aceluiași script nu este executat.

Repară ghilimeaua și repetă testul. Nu memora formularea exactă a erorii, deoarece poate diferi între browsere. Recunoaște categoria și poziția indicată. Uneori locul unde parserul detectează problema este după greșeala reală. Citește și liniile din jur, în special perechile de ghilimele, paranteze și acolade.

Experimentul 3: elementul obligatoriu lipsește

Cu fișierele din nou corecte, schimbă id="stare" în id="mesaj" numai în HTML. Acum scriptul pornește și primul console.log apare. Căutarea întoarce null, iar eroarea noastră oprește restul acestui script. Mesajul „Inițializare încheiată.” nu trebuie să apară pentru această execuție.

Corectura este să aliniezi identificatorul și selectorul. Nu elimina verificarea doar ca să dispară mesajul. Ea a descoperit exact nepotrivirea pe care voiam să o vedem. În raport explică de ce avem dovada pornirii, dar nu și a finalizării, spre deosebire de eroarea de sintaxă din experimentul anterior.

Experimentul 4: schimbarea din consolă nu rescrie fișierul

În consola paginii funcționale, execută instrucțiunea următoare. Observă mesajul nou, apoi reîncarcă pagina fără să modifici app.js.

```
document.querySelector("#stare").textContent = "Schimbare temporară.";
```

După reîncărcare revine mesajul produs de fișier. Ai modificat documentul curent din memoria browserului, nu sursa salvată în editor. Această diferență explică multe situații în care un începător spune că „browserul nu păstrează corectura”. Pentru o schimbare persistentă în proiect, modifică fișierul potrivit, salvează și reîncarcă.

Consola este un loc de execuție, nu un spațiu în care orice text copiat este inofensiv. Pentru această lecție tastează instrucțiunile pe care le explicăm, în pagina laboratorului. Nu rula cod primit de la persoane necunoscute în pagini unde ești autentificat și nu folosi consola ca depozit pentru secrete.

Comparație opțională cu Node.js

Dacă ai deja Node.js instalat, salvează acest fragment într-un fișier mediu.js și rulează node mediu.js în terminal. Execută aceleași instrucțiuni și în consola paginii laboratorului, într-un context obișnuit, fără biblioteci care adaugă obiecte globale.

```
console.log(500 + 1200 + 300);  
console.log(typeof document);  
console.log(typeof process);
```

Calculul produce 2000 în ambele medii. În pagina browserului, document este disponibil, iar process nu este în mod obișnuit un global oferit de browser. În Node.js, process este disponibil, iar document nu este furnizat ca DOM al unei pagini. Folosim typeof pentru această comparație fără a accesa direct un identificator absent.

Nu instala Node.js numai pentru a trece de această lecție dacă vrei momentan să lucrezi exclusiv în browser. Comparația este opțională. Dacă o execuți, notează versiunea și rezultatele reale. Nu concluziona că toate API-urile sunt diferite: mediile au și facilități comune, iar compatibilitatea evoluează.

Exerciții cu repere de rezolvare

Prima sarcină este să schimbi mesajul final în „Laboratorul meu este pregătit.” fără să modifice identificatorul. Rezolvarea se află în atribuirea `textContent` din `app.js`. Verifică faptul că schimbarea persistă după reload și că Network livrează fișierul salvat, nu o copie veche din alt director.

A doua sarcină este să explici două cazuri care arată identic în pagină: script lipsă și element cu identificator greșit. Reperele sunt diferite în instrumente. În primul caz, cererea scriptului eșuează și codul nu pornește. În al doilea, resursa se încarcă, primul mesaj apare și verificarea elementului produce eroarea. Un screenshot al paginii singure nu este suficient.

A treia sarcină este să atribui mesajului textul literal „Pregătit”. Cu `textContent`, parantezele unghiulare și numele elementului trebuie să fie afișate ca text, fără a crea un element `strong` în acel paragraf. Rezolvarea nu cere `innerHTML`. Scopul este să verifici mecanismul ales, nu să obții text îngroșat.

A patra sarcină este să explici ce se întâmplă dacă oprești serverul și încerci o cerere nouă cu cache-ul dezactivat. Nu confunda pagina deja încărcată cu accesibilitatea serverului. Documentul existent poate rămâne vizibil; o cerere nouă nu va primi răspuns de la procesul oprit. Încheie exercițiul repornind serverul și verificând revenirea.

Criterii de finalizare și pasul următor

Lecția este încheiată când poți recrea cele trei fișiere, identifica resursele încărcate și explica cele trei categorii de eroare: încărcare, sintaxă și execuție. Predă registrul cu cel puțin o observație pentru fiecare experiment și o copie funcțională a laboratorului. Nu lăsa versiunea cu erori drept singura variantă salvată.

Timpul afișat pentru lecție estimează lectura prozei. Configurarea mediului, tastarea codului și depanarea se adaugă separat. Pentru prima parcurgere rezervă o sesiune în care poți opri și relua experimentele fără presiunea de a termina într-un număr fix de minute. În lecția următoare folosim această bază pentru variabile, tipuri și conversii, înainte să construim totalul bugetului.

Documentație verificată la 20 octombrie 2026

- JavaScript și rolul său în pagină
- Elementul `script` și `defer`
- Browser și Node.js
- Depanare în Chrome DevTools
- `textContent`
- `querySelector`
- Serverul demonstrativ Python