

Examen Frontend Development 2025

HTML

Întrebarea 1. Analiza și corectarea codului.

Codul HTML de mai jos e folosit pentru a crea o postare într-un layout de blog și conține un header, o zonă main cu conținut article, aside și un footer. Însă conține și anumite erori semantice și structurale. Identifică cel puțin 3 erori și explică de ce sunt incorecte din punct de vedere semantic și a accesibilității.

1. Folosirea greșită a elementului ``.
Corect ar fi de folosit un `<main>` sau `<section>` în locul lui ``
2. `` nu transmite nicio semnificație semantică.
Dacă textul este important sau un titlu, ar trebui `<h2>`, `<h3>` etc.
Structura ierarhică `h1`, `h2`, `h3` este importantă pentru claritate.
3. Este recomandat de atribut `alt` pentru imagine pentru evidențierea în cazul unei erori de afisare a acesteia

```
<div>
  <div class="header">
    <h1>Blogul meu</h1>
  </div>
  <span>
    <p>Main Content</p>
    <div>
      <b>Articol</b>
      <br>
      
      <p>Primul paragraf...</p>
    </div>
    <div class="sidebar">
      <b>Linkuri</b>
      <ul>
        <li><a href="#">Link 1</a></li>
        <li><a href="#">Link 2</a></li>
      </ul>
    </div>
  </span>
  <div class="footer">
    Copyright 2025
  </div>
</div>
```

Întrebarea 2. Meta taguri pentru Social Media

Când un utilizator face share la pagină pe o rețea socială precum Facebook sau Instagram (sau Teams), dorim ca pagina noastră să aibă un preview care conține o imagine, un titlu și o descriere. Cu ce set de meta taguri putem obține acest lucru?

- A) `<meta name="author">`, `<meta name="keywords">`, `<meta name="description">`
- ☒ B) Taguri OpenGraph precum `<meta property="og:title">` și `<meta property="og:image">`
- C) `<meta name="viewport">` și `<meta charset="UTF-8">`
- D) Taguri Dublin Core precum `<meta name="DC.title">` și `<meta name="DC.title">`

Întrebarea 3. Formulare și Validare

Creăm un formular pentru o pagină de produse unde putem selecta o cantitate. Câmpul de introducere a datelor trebuie să urmeze următoarele reguli:

- Cantitatea minimă permisă e 10
- Cantitatea maximă permisă e 50
- Cantitatea trebuie să fie incrementată în pași de 5 (ex. 10, 15, 20, etc.)

Ce input de HTML e configurat corect cu aceste reguli?

- A) `<input type="number" min="10" maxlength="50" step="5">`
- B) `<input type="text" min="10" max="50" step="5">`
- C) `<input type="number" min="10" max="50" step="5">`**
- D) `<input type="number" range="10-50" step="5">`

CSS

Întrebarea 1. Ordinea ierarhică a selectorilor.

Analizează cele 2 reguli CSS de mai jos care sunt aplicate aceluiași element `<h1>`:

HTML:

```
<header id="page-header" class="site-header">
|   <h1 class="title">Salut!</h1>
</header>
```

CSS:

```
/* Regula A */
#page-header h1 {
  color: blue;
}

/* Regula B */
header.site-header h1.title {
  color: red;
}
```

Care din cele 2 reguli va fi aplicată elementului și de ce?

Regula A deoarece apeleaza corect elementul H1 din interiorul elementului cu id

Întrebarea 2. Analiza unui layout și debuggingul codului.

Un developer vrea să creeze un navigation bar unde logoul paginii este pe partea stângă iar link-urile de navigație sunt grupate împreună pe partea dreaptă. Developerul folosește flexbox, însă toate elementele se grupează împreună pe stânga în loc de a se crea spațiu între dânsese.

Identifică eroarea din codul de CSS și explică din ce cauză nu se creează acel layout de tip 'spațiu-între'. Apoi scrie linia corectă.

HTML:

```

<nav class="main-nav">
  <div class="logo">
    Logo
  </div>
  <ul class="nav-links">
    <li><a href="#">Home</a></li>
    <li><a href="#">About</a></li>
    <li><a href="#">Contact</a></li>
  </ul>
</nav>

```

CSS:

```

.main-nav {
  display: flex;
  align-items: space-between;
  background-color: #eee;
  padding: 10px;
}

.nav-links {
  display: flex;
  list-style: none;
  padding: 0;
  margin: 0;
}

.nav-links li {
  margin-left: 20px;
}

```

NU e corect align items: space-between ci justify-content: space-between:
tot la .main-nav mai adaugam align items center

```

.main-nav {
  display: flex;
  justify-content: space-between; /* <- corect */
  padding: 10px;
}

```

Întrebarea 3. Box Model.

Un element are următoarele proprietăți de CSS:

```
width: 100px;  
padding: 20px;  
border: 5px solid black;  
box-sizing: border-box;
```

Care va fi lățimea totală vizibilă a acestui element pe ecran?

- A) 150px
- B) 125px
- C) 100px**
- D) 145px

Întrebarea 4. Elemente avansate ale CSS-ului

Dorim să stilizăm prima literă a fiecărui paragraf dintr-un element articol cu anumite proprietăți (de exemplu să fie mai mare sau să aibă o culoare diferită). Ce selector CSS din următoarele ar fi cel mai corect și eficient pentru a face asta?

- A) article p::first-letter**
- B) article p:first-child
- C) .article-paragraph::first-letter
- D) article > p:first-letter

Întrebarea 5. Adevărat sau fals.

Aplicând proprietatea **z-index:999;** pentru un <div> va face ca întotdeauna acest div să fie plasat peste restul elementelor pe pagină.

A) Adevărat

☒ B) Fals



Întrebarea 6. Media Queries.

Mai jos avem 2 bucăți de cod care aplică culori diferite de fundal la <body> în baza mărimii ecranului. Una din aceste bucăți de cod folosește metoda de lucru **mobile-first**, cealaltă folosește metoda de lucru **desktop-first**. Care din dânsele e mobile-first și care e desktop-first?

```
/* Bucata de cod A */
body {
  background-color: lightblue;
}

@media (min-width: 768px) {
  body {
    background-color: steelblue;
  }
}
```

A- mobile first
B- desktop first

```
/* Bucata de cod B */  
body {  
  background-color:  steelblue;  
}  
  
@media (max-width: 767px) {  
  body {  
    background-color:  lightblue;  
  }  
}
```

JavaScript

Întrebarea 1. DOM.

Avem următorul cod de HTML și JS. După ce executăm JavaScriptul, apăsăm pe butonul "Adaugă Paragraf Nou". Ce va fi afișat la consolă?


```

<div id="container">
  <p>Paragraf A</p>
  <p>Paragraf B</p>
</div>
<button id="btnAdauga">Adaugă Paragraf Nou</button>

<script>
  const cont = document.getElementById('container');

  const pInitNodeList = cont.querySelectorAll('p');
  let nrPInit = pInitNodeList.length;

  document.getElementById('btnAdauga').addEventListener('click', () => {
    const pNou = document.createElement('p');
    pNou.textContent = 'Paragraf C';
    cont.appendChild(pNou);

    const pActualNodeList = cont.querySelectorAll('p');
    let nrPActual = pActualNodeList.length;

    console.log(`La început: ${nrPInit}, Acum: ${nrPActual}`);
  });
</script>

```

- A) La început: 2, Acum: 2
- B) La început: 3, Acum: 3
- C) La început: 2, Acum: 3
- D) La început: 3, Acum: 2

Întrebarea 2. Operatori

Care din următoarele expresii va returna rezultatul **True**?

- A) `null === undefined`
- B) `0 == false`
- C) `"" == 0`
- D) `NaN === NaN`

Întrebarea 3. Storage

Session storage basteaza informatia pe browser pana nu inchidem browserul dat
exemplu: pastreaza informatia din cosul de cumparaturi al unui site unde nu este necesara logarea

Local storage stocheaza informatia si dupa ce inchidem informatia;
exemplu: salvarea login si parolei pentru un account

Explică 2 diferențe dintre localStorage și sessionStorage. Dă câte un exemplu de utilizare pentru fiecare.

Întrebarea 4. Clase

Care e scopul apelului metodei **super()** în constructorul clasei Patrat?

```
class Forma {  
  constructor(name) {  
    this.name = name;  
  }  
}  
  
class Patrat extends Forma {  
  constructor(length) {  
    super('pătrat');  
    this.length = length;  
  }  
}
```

- ☒ A) Apelează constructorul clasei părinte și trimite "pătrat" ca parametrul name.
- B) Crează o nouă instanță a clasei Forma în interiorul obiectului Patrat
- C) Verifică dacă clasa părinte Forma are un constructor valid
- D) Copiază toate metodele din clasa Forma în clasa Patrat

Întrebarea 5. Module în ES6

Avem 2 fișiere. Cum importăm corect **pi** și **add** din [math.js](#) în [app.js](#)?

[math.js](#):

```
export const pi = 3.14;

export default function add(a, b) {
  return a + b;
}
```

[app.js](#):

```
// Ce statement de import scriem aici?
```

- ☒ A) import { add, pi } from './[math.js](#)';
- B) import add, { pi } from './[math.js](#)';
- C) import { default as add, pi } from './[math.js](#)';
- D) import add and { pi } from './[math.js](#)';

Partea Practică

Vei crea o pagină web pentru managementul unei liste de sarcini (task list). Proiectul este împărțit în 4 părți, fiecare bazându-se pe cea anterioară.

Setup Inițial (înainte de a începe):

- Creează un repository nou pe GitHub și clonează-l pe calculatorul tău.
- În folderul proiectului, creează fișierele: **index.html**, **style.css**, **script.js** și **db.json**
- După fiecare parte **Fă un commit**.

Partea 1: Structura paginii (HTML)

- ✓ 1) Creează structura HTML standard (<!DOCTYPE html>, etc)
- ✓ 2) În <head>, adaugă un titlu și conectează fișierul style.css
- ✓ 3) Conectează un font la alegere de pe Google Fonts
- ✓ 4) Adaugă elementele semantice specifice structurii paginii, împreună cu un <h1> cu titlul proiectului
- ✓ 5) În main, creează un formular. În interiorul formularului adaugă un input de tip text, un label și un buton pentru acest buton.
- ✓ 6) Sub formular, tot în main creează o secțiune.
- ✓ 7) În interiorul secțiunii adaugă un subtitlu (h2). Aici vei pune todos-urile
- 8) Adaugă lista ordonată/neordonată în care vor fi afișate sarcinile
- 9) Adaugă în footer un text simplu (ex copyright)
- 10) Adaugă scripturile

Partea 2. CSS

- 1) Importă fontul de pe google fonts pe care îl ai și în HTML.
- 2) Setează toate elementele cu tipul de cutie border-box.
- 3) Stilizează body (setează fontul, culoare de fundal etc)
- 4) Stilizează elementele de layout (header, main etc) pentru a structura vizual pagina (folosește width, max-width, margin, padding, background-color)
- 5) Stilizează câmpul input adăugând stiluri și pentru starea în care avem inputul selectat
- 6) Stilizează butonul pentru starea în care mouse-ul este pe buton și pentru starea în care butonul este apăsat.
- 7) Adaugă o tranziție pentru ambele elemente de mai sus.
- 8) Scoate stilurile default ale listei
- 9) Pentru fiecare element al listei , seteaza-l ca fiind un container flexbox pentru a alinia elementele pe interior.

- 10) Folosește proprietățile Flexbox pentru a poziționa textul sarcinii în stânga și butoanele de editare/ștergere în dreapta.
- 11) Crează o clasă de CSS (de exemplu .completed) care va fi adăugată cu JS la sarcinile finalizate. Ar trebui să aplice stiluri precum **text-decoration: line-through;** sau **opacity: 0.6.** **Nu vom implementa actualizarea, deoarece ar lua prea mult timp, dar pregătim terenul pentru 'potențiale' modificări viitoare.**
- 12) Adaugă un media query pentru ecranele mici până la 600px, în interiorul acestui bloc ajustează stilurile pentru a face aplicația utilizabilă pe mobil.

Partea 3. JS

Recomandări: Dacă dorești, folosește axios vei avea mai puțin cod de scris. Nu este o cerință obligatorie

- 1) Preia toate elementele din DOM (input, lista, buton etc)
- 2) Crează o variabilă unde vei avea adresa pentru json-server, de exemplu
const API_URL = '<http://localhost:3000/tasks>';
- 3) Afișare (GET). Crează o funcție fetchTasks() care:
 - a) Folosește metoda **get** pentru a prelua toate taskurile de pe server
 - b) Afișează fiecare sarcină în listă, adăugând butoane de Edit și Delete, adaugă la fiecare sarcină un atribut **data-id** care va conține id-ul sarcinii.
 - c) Apelează funcția atunci când pagina se încarcă
- 4) Adăugare (POST). Adaugă un event listener pe butonul de Adaugă care:
 - a) Trimite sarcina nouă din formular către server
 - b) Golește inputul și face refresh la lista de sarcini.
- 5) Ștergere (DELETE). Adaugă un event listener pentru el care, la un click pe butonul Șterge va face următoarele:
 - a) Va obține ID-ul sarcinii din atributul data-id
 - b) Trimite o cerere de delete către json-server pentru acel id.

c) Face refresh la lista de sarcini.

Partea 4. Unelte de lucru.

- 1) Inițializează npm și configurează proiectul.
- 2) Instalează dependența json-server.
- 3) Configurează fișierul db.json (adaugă o listă de sarcini inițiale).
- 4) Pornește serverul.
- 5) Instalează dependența prettier și 'curăță' proiectul.
- 6) Testează proiectul apoi fă commit-ul final.
- 7) Fă push pe github.