

Repàs General JavaScript

Preparats per a l'Examen

DOM • Esdeveniments • Formularis • Asincronia



Objectiu: Organitzar conceptes, no aprendre contingut nou.

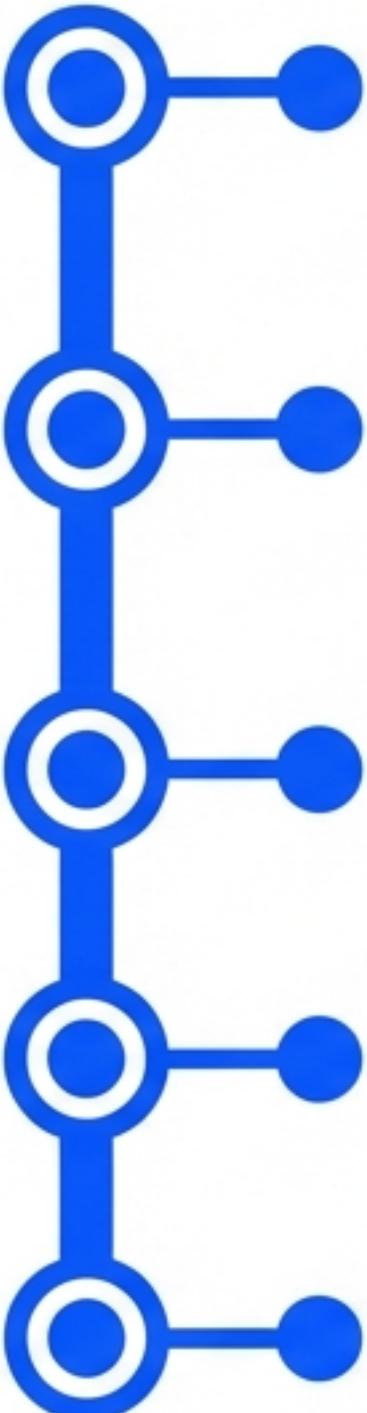


Ja heu treballat: DOM, Events, Promeses, APIs.



Resultat: Seguretat i claredat mental.

El Mapa de l'Examen (Què entra?)

- **Manipulació bàsica del DOM**
Selecció, creació, modificació
- Gestió d'esdeveniments**
click, submit
- Ús de promeses i async / await**
- Consum d'una API amb fetch**
- Mostrar dades al DOM**
Renderitzat

Idea Clau

No es demana memoritzar sintaxi exacta, sinó entendre el flux d'execució del codi.

Manipulació del DOM (Teoria)

Conceptes Essencials

- **Selecció**

- `querySelector`, `querySelectorAll`,
`getElementById`

- **Creació**

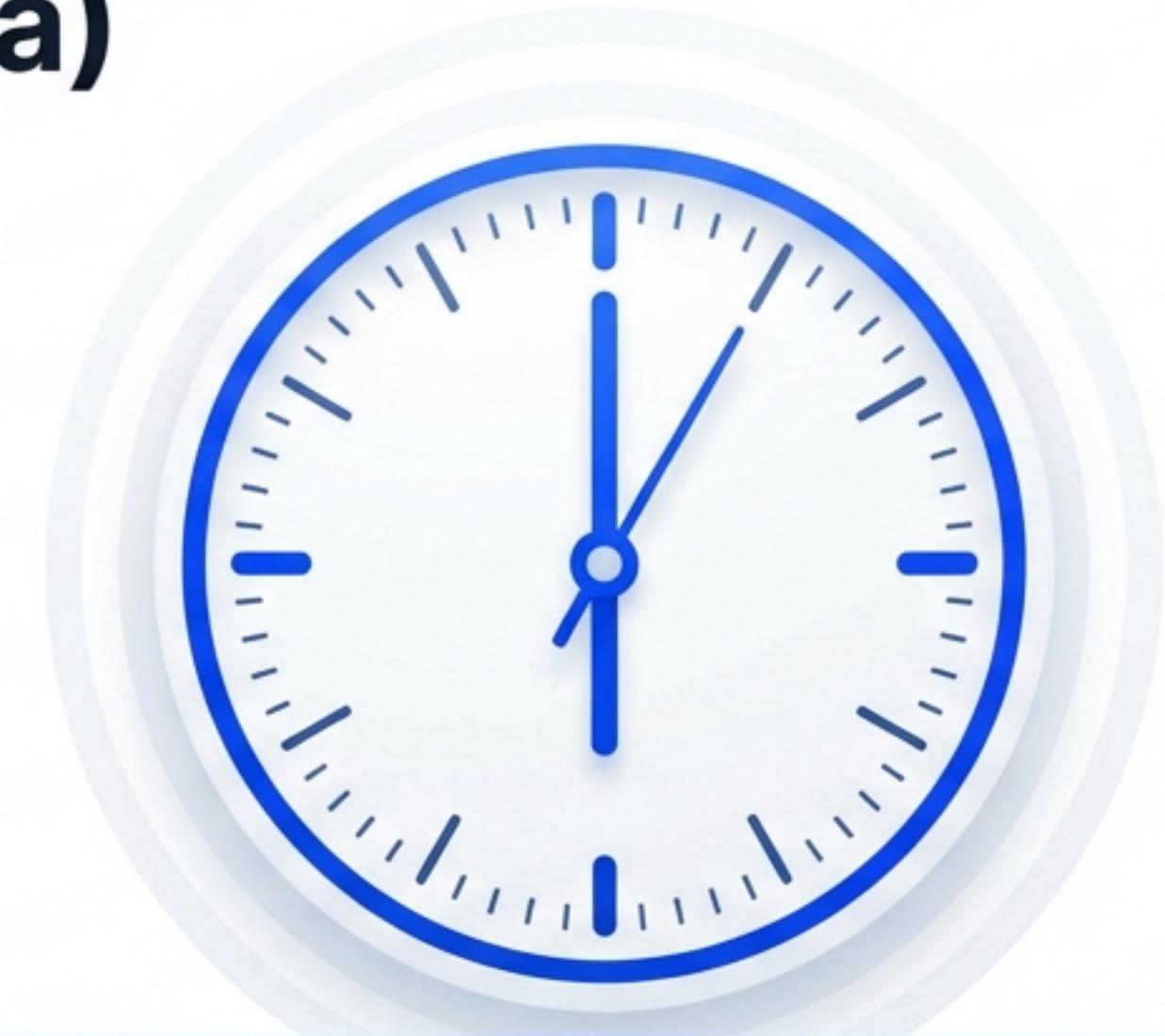
- `document.createElement()`

- **Actualització**

- Preferir `textContent` (ràpid) vs
`innerHTML` (lent/risc)

- **Estils**

- `classList (add, remove, toggle)`



Idea Clau

El DOM és síncron. Els canvis són immediats.

Diferenciar entre Node (qualsevol part del document) i Element (etiquetes HTML).

El DOM a la Pràctica

```
let el = document.createElement('p'); ← Punt d'inici obligatori  
el.innerText = 'Element Creat en un <p>';  
document.body.append(el); // Pot rebre text i nodes ← Versàtil: accepta Text i Nodes  
  
let div = document.createElement('div');  
div.innerText = 'Sóc un text en un <div>';  
  
el.replaceWith(div); // Substitució  
  
// Inserció precisa  
document.body.insertAdjacentElement('beforebegin', div); ← Control total de la posició  
document.body.insertAdjacentElement('beforeend', div);
```

Punt d'inici obligatori

Versàtil: accepta Text i Nodes

Control total de la posició

Secció 2: Interactivitat

Esdeveniments (Events)



Sintaxi Moderna

```
`addEventListener('tipus', funció)`
```

Objecte Event (e)

Conté `e.target` (element originari)

Tipus Comuns

`click`, `submit`, `input`, `change`

Idea Clau: Un esdeveniment executa una funció **QUAN** passa una acció (no abans).

Formularis i Validació



Patró de Formularis: Lògica i Renderitzat

Part 1: Separació de Responsabilitats

```
1 let validate = data => {                                Lògica (Validació)
2     let errors = {};
3     if (!data.name) errors.name = 'Camp obligatori';
4     if (!data.lastname) errors.lastname = 'Camp obligatori';
5     return errors;
6 }
7
8 let render = ({ data, errors }) => {                  Vista (Renderitzat)
9     return `
10         <input name='name' value='${data.name}' /> ${errors.name || ''}
11         <input name='lastname' value='${data.lastname}' /> ${errors.lastname || ''}
12         <button>Enviar</button>
13     `;
14 }
```

Funció pura: Rep dades, retorna HTML string.

Patró de Formularis: El Listener

Part 2: Connectant l'esdeveniment

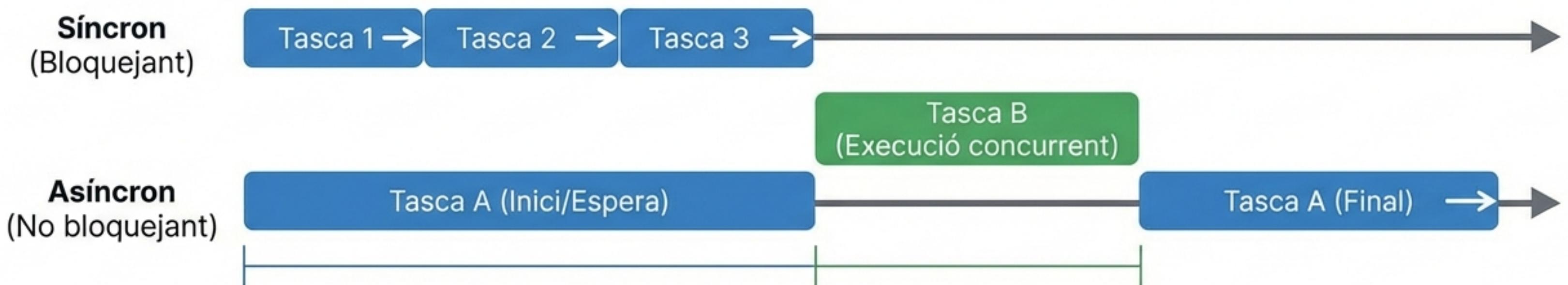
```
1 form.addEventListener('submit', e => {
2     e.preventDefault(); // PAS 1: Aturar enviament
3
4     // PAS 2: Llegir dades
5     let data = Array.from(e.target.elements).reduce((acc, el) => {
6         if (!el.name) return acc;
7         acc[el.name] = el.value;
8         return acc;
9     }, {})
10
11    // PAS 3: Validar
12    const errors = validate(data);
13
14    // PAS 4: Gestionar errors
15    if (Object.keys(errors).length > 0) {
16        let html = render({ errors, data });
17        form.innerHTML = html;
18        return;
19    }
20    // Si no hi ha errors, continuar...
21})
```



Gatekeeper: Si hi ha errors, parem aquí.

Asincronia i Promeses

Comparativa: Síncron vs. Asíncron



Pending
(Pendent - Esperant resposta)
⌚

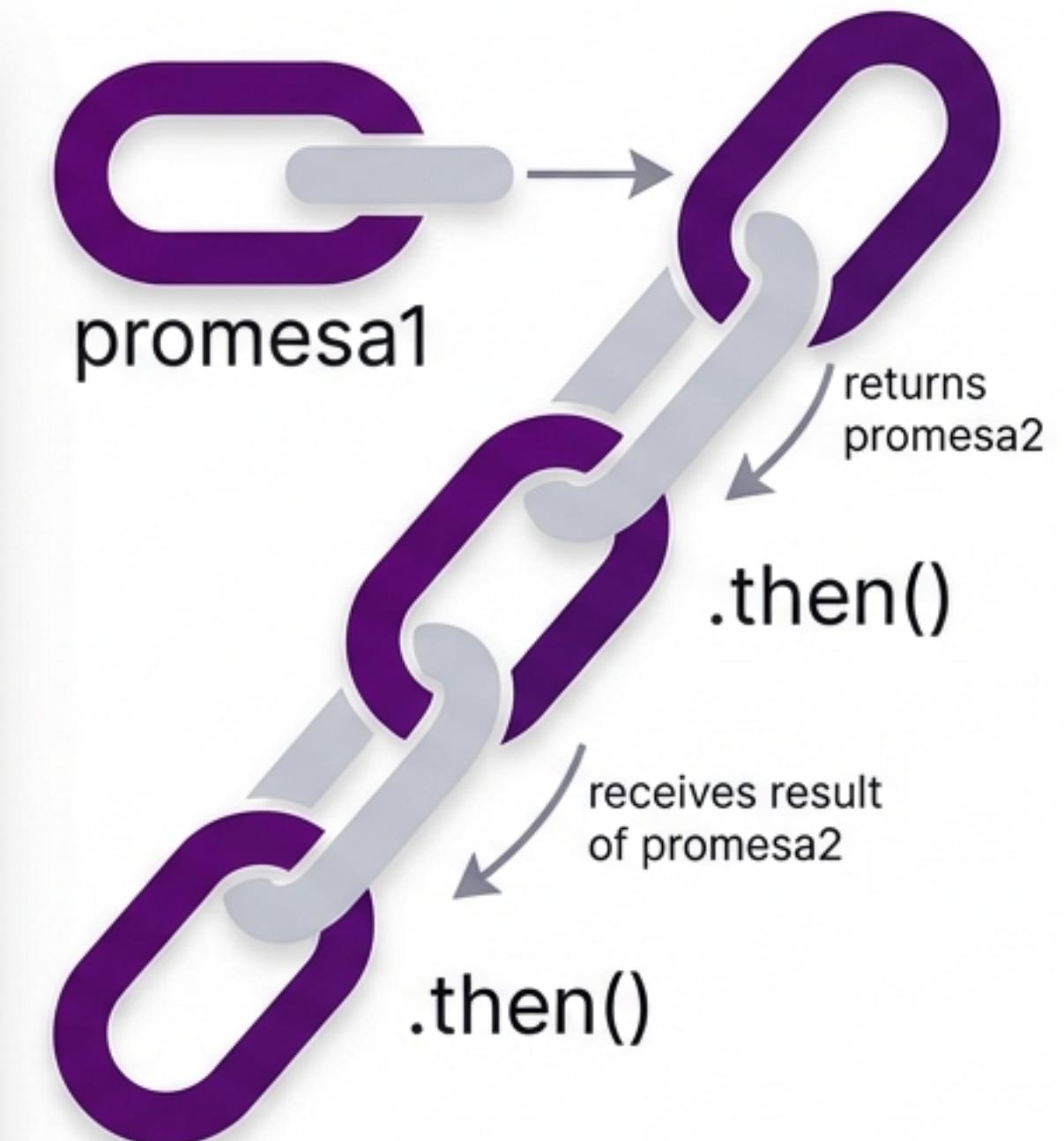
Fulfilled
(Èxit - ` .then()`)
✓

Rejected
(Error - ` .catch()`)
✗

Idea Clau: JavaScript no espera automàticament. Les promeses són el mecanisme per controlar el flux.

Promeses en Acció (Encadenament)

```
1 let promesa1 = new Promise((res, rej) => { res(12); });
2 let promesa2 = new Promise((res, rej) => { res(15); });
3
4 promesa1.then(valor => {
5     if (valor > 10) {
6         return promesa2; // Retornem una nova promesa
7     }
8 }).then((valor2) => {
9     console.log(valor2); // 15
10    return valor2;
11 })
12 .catch(e => console.log(e))
13 .finally(() => console.log('estem a finally!'))
```



Async / Await (Sintaxi Moderna)

Vella Manera (Old Way)

```
.then(data => {  
    ...  
})
```

Evolution / Sugar Syntax

Nova Manera (New Way)

```
let data = await ...
```

- ✓ **await** només funciona dins d'una funció **async**.
- ✓ **await** atura l'execució de la funció fins que la promesa es resol.
- ✓ Es gestionen els errors amb blocs **try / catch**.

Idea Clau

Fa el codi més lleigible (sempbla síncron), però continua sent asíncron per sota.

Fetch (Consum d'APIs)



Recorda: `fetch` no falla en errors 404. Cal comprovar `response.ok`.

Exemple de Flux Asíncron

```
1 function transformarText(text) {  
2     return new Promise((resolve, reject) => {  
3         if (!text || typeof text !== 'string') reject('Text no valid');  
4         else resolve(text.toUpperCase());  
5     });  
6 }  
7  
8 transformarText('Hi Everybody')  
9     .then(resultat => afegirInfo(resultat)) // Processament encadenat  
10    .then(final => console.log(final))  
11    .catch(error => console.error('Error:', error)) ← Captura errors de  
12    .finally(() => console.log('Procés finalitzat'));
```

Cada pas espera l'anterior.

Captura errors de qualsevol pas anterior.

Errors Habituals (Checklist)



Oblidar `await`



El codi continuarà executant-se sense tenir les dades.

Resultat: `undefined` o Promesa pendent.

Oblidar `await response.json()`



La resposta HTTP no és usable directament. Cal analitzar el body.

Errors HTTP (404/500)



Pensar que el `catch` captura un 404. Fetch només falla per errors de xarxa.

Formularis sense control



No prevenir el comportament per defecte (`e.preventDefault()`).

La pàgina es recarrega.

Renderitzat prematur



Intentar pintar dades al DOM abans que arribin del servidor.

Estratègia per a l'Examen

1. Llegir amb calma

Identificar si el codi és síncron o asíncron.

2. Identificar retorna

Què retorna cada funció? Un valor simple? Una promesa?

3. Pas a pas

Escriure el codi seguint el flux lògic, no tot de cop.

4. Patró Fetch + DOM

Carregar dades -> Recórrer array -> Crear elements -> Append.

Entendre el flux és més important que escriure molt codi. Molta sort!