Tema 3. Arrays i funcions

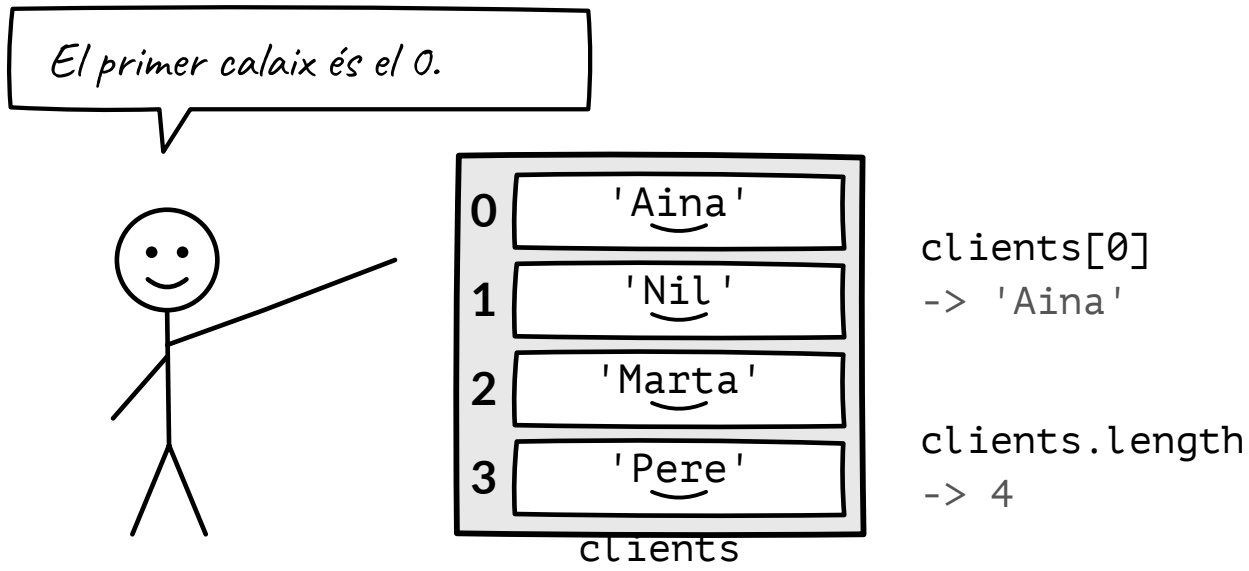


FIG 3.0 Un array és una fila de calaixos numerats. El primer és el 0.

Al tema 2 cada variable guardava un sol valor. Però una botiga no té un client: en té cent. Aquí aprendràs a guardar **llistes** de dades, a recórrer-les amb els bucles que ja coneixes i a agrupar codi en **funcions**. Al final del tema, un bucle pintarà la llista de clients **en una pàgina web**.

Què aprendràs

- Crear arrays, llegir-los per índex i afegir-hi o treure'n elements.
- Recórrer un array amb `for` i `for...of` per sumar, comptar i buscar.
- Definir funcions amb paràmetres i `return`, i escriure-les en forma de fletxa.
- Guardar les dades d'un client en un objecte i fer arrays d'objectes.
- Filtrar, transformar i sumar llistes amb `filter`, `map`, `find` i `reduce`.
- Pintar una llista a la pàgina amb `createElement` i `appendChild`.

La teva xuleta per a tot el tema

Copia-la o deixa aquesta pàgina marcada. **Fer-la servir no és fer trampa**: els professionals també tenen les seves.

Vull...	Escric...
Crear un array	`const noms = ['Aina', 'Nil'];`
El primer element	`noms[0]`
Quants elements té	`noms.length`
Afegir al final	`noms.push('Marta');`
Treure l'últim	`noms.pop()`
Recórrer-lo	`for (const nom of noms) { }`
Definir una funció	`function doble(n) { return n * 2; }`
La mateixa, en fletxa	`const doble = (n) => n * 2;`
Una fitxa de dades	`const client = { nom: 'Aina', saldo: 120 };`
Llegir-ne un camp	`client.nom`
Quedar-me amb uns quants	`llista.filter((c) => c.saldo < 0)`
Transformar-los tots	`llista.map((c) => c.nom)`
Crear un element a la pàgina	`document.createElement('li')`
Posar-lo dins d'un altre	`pare.appendChild(fill)`

---

1. Arrays

Un **array** és una llista de valors guardada en una sola variable. Els valors van entre claudàtors `[ ]` i separats per comes.

arrays.js

```
const clients = ['Aina', 'Nil', 'Marta'];
console.log(clients[0]);
console.log(clients[2]);
console.log(clients.length);
console.log(clients[3]);
```

```
Aina
Marta
3
undefined
```

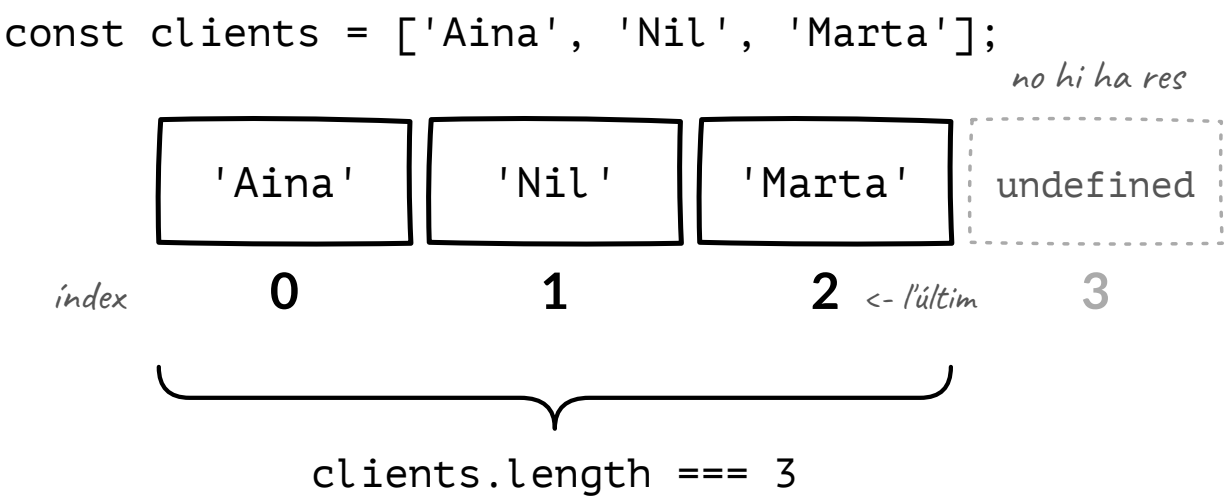


FIG 3.1 Cada element té un índex. El primer és el 0 i l'últim és `length - 1`.

- Cada element té una posició, l'**índex** (*index*). **El primer és el 0**, no l'1.
- `clients[0]` llegeix l'element de la posició 0.
- `length` diu quants elements hi ha. Aquí 3, i per tant l'últim índex és el **2**.
- Si demanes una posició que no existeix, JavaScript no s'atura: dona `undefined`.

Regla ràpida: **l'últim índex sempre és** `length - 1`.

Canviar un array

compra.js

```
const compra = ['pa', 'llet'];
compra.push('ous');
console.log(compra);
compra[0] = 'pa integral';
console.log(compra);
const darrer = compra.pop();
console.log(darrer, compra.length);
```

```
[ 'pa', 'llet', 'ous' ]
[ 'pa integral', 'llet', 'ous' ]
ous 2
```

- `push` afegeix al final.
- `compra[0] = ...` canvia l'element d'una posició.
- `pop` treu l'últim i te'l torna. Per això el pots guardar a `darrer`.

Fixa't que `compra` és `const` i l'hem canviada. `const` **vol dir que la variable no pot apuntar a un altre array**, però el contingut de l'array sí que es pot canviar. Si escrius `compra = ['cafè'];`, surt `TypeError: Assignment to constant variable.`, com al tema 2.

Practica 3.1

Taula de traça. Escriu com queda l'array després de cada línia.

pila.js

```
const pila = ['A', 'B'];
pila.push('C');
pila[1] = 'X';
pila.pop();
pila.push('D');
```

Línia	pila	pila.length
1	['A', 'B']	2
2		
3		
4		
5		

2. Recórrer un array

Amb un `for` del tema 2 pots passar per tots els índexs, del 0 a `length - 1`:

recorrer.js

```
const notes = [7, 4.5, 9, 6];
for (let i = 0; i < notes.length; i++) {
  console.log(`Nota ${i + 1}: ${notes[i]}`);
}
```

```
Nota 1: 7
Nota 2: 4.5
Nota 3: 9
Nota 4: 6
```

Fixa't en la condició: `i < notes.length`, **amb `<` i no amb `<=`**. Si posessis `<=`, l'última volta llegiria `notes[4]`, que és `undefined`.

Quan no et cal l'índex, hi ha una forma més curta i més clara, el `for...of`. A cada volta, la variable `nota` val el següent element:

mitjana.js

```
const notes = [7, 4.5, 9, 6];
let suma = 0;
for (const nota of notes) {
  suma = suma + nota;
}
console.log('Mitjana:', suma / notes.length);
```

```
Mitjana: 6.625
```

La variable del `for...of` pot ser `const`: a cada volta se'n crea una de nova.

Exemple resolt. Buscar el màxim

maxima.js

```
const temperatures = [18, 23, 21, 25, 19];
let maxima = temperatures[0];
for (const t of temperatures) {
  if (t > maxima) {
    maxima = t;
  }
}
console.log(maxima);
```

Volta	`t`	`t > maxima`?	`maxima` després
—	—	—	18 (el primer)
1	18	`false`	18
2	23	`true`	23
3	21	`false`	23
4	25	`true`	25
5	19	`false`	25

Surt 25. **La idea: comences amb el primer element i el canvies cada vegada que en trobes un de més gran.**

Guiat 3.2

Completar codi. El programa ha de comptar quantes notes són aprovats. Omple els buits.

aprovats.js

```
const notes = [7, 4.5, 9, 3, 6];
let aprovats = ______;
for (const nota ______ notes) {
  if (nota ______ 5) {
    aprovats______;
  }
}
console.log(`${aprovats} aprovats de ${notes.length}`);
```

Surt: .....

Practica 3.3

Caçar l'error. Aquest programa hauria de mostrar els tres noms. Mostra els tres i, després, `undefined`.

noms.js

```
const noms = ['Aina', 'Nil', 'Marta'];
for (let i = 0; i <= noms.length; i++) {
  console.log(noms[i]);
}
```

Quin caràcter sobra i per què?

3. Funcions

Una **funció** (*function*) és un tros de codi amb nom que pots fer servir tantes vegades com vulguis. La **defineixes** una vegada i la **crides** quan la necessites.

saluda.js

```
function saluda(nom) {
  console.log(`Hola, ${nom}!`);
}

saluda('Aina');
saluda('Nil');
```

```
Hola, Aina!
Hola, Nil!
```

- `function saluda(nom)` defineix la funció. **Encara no fa res.**
- `nom` és un **paràmetre** (*parameter*): una variable que val el que li passis en cridar-la.
- `saluda('Aina')` crida la funció. Ara sí que s'executa, amb `nom` valent `'Aina'`.

`return`: la funció et torna un valor

La majoria de funcions no mostren res: **calculen un valor i te'l tornen** amb `return`. Qui l'ha cridada decideix què en fa.

iva.js

```
function preuAmbIva(preu) {
  return preu * 1.21;
}

const total = preuAmbIva(100);
console.log(total);
console.log(preuAmbIva(50) + 10);
```

```
121
70.5
```

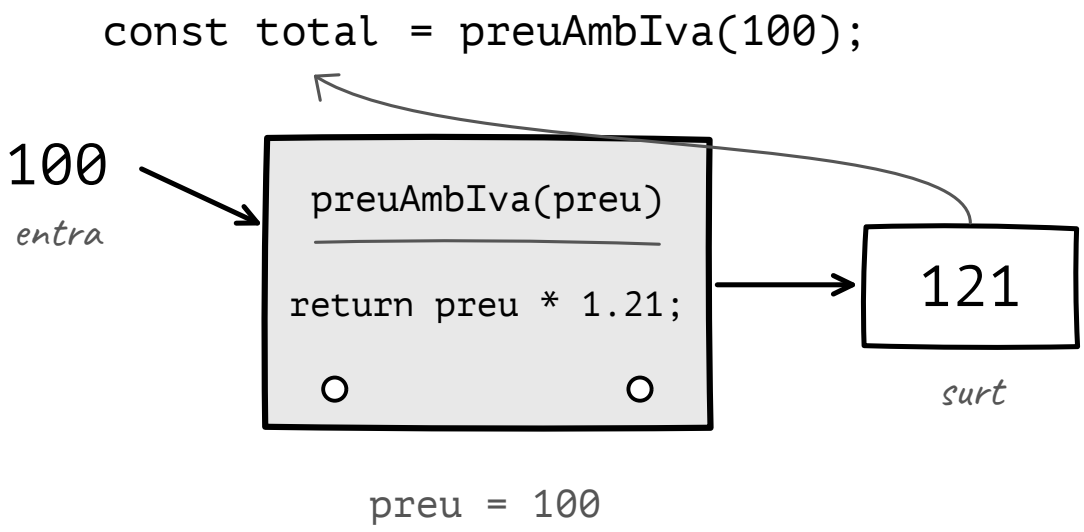


FIG 3.2 Una funció és com una màquina: hi entra el paràmetre i en surt el `return`.

On escrius `preuAmbIva(50)`, és com si hi posessis el número que torna: `60.5`. Per això el pots sumar, guardar o passar a una altra funció. **Quan s'executa `return`, la funció s'acaba**: les línies de sota ja no s'executen.

`console.log` no és `return`

mal.js

```
function suma(a, b) {
  console.log(a + b);
}

const resultat = suma(2, 3);
console.log('resultat val', resultat);
```

```
5
resultat val undefined
```

La funció **mostra** el 5, però no el **torna**. Una funció sense `return` torna `undefined`. És l'error de funcions més habitual.

Funcions fletxa

Hi ha una forma curta d'escriure funcions, la **funció fletxa** (*arrow function*). Aquestes dues fan el mateix:

fletxa.js

```
function doble(n) {
  return n * 2;
}

const dobleFletxa = (n) => n * 2;

console.log(doble(4), dobleFletxa(4));
```

```
8 8
```

Quan el cos de la fletxa és una sola expressió, **no porta** `return` **ni claus**: torna el resultat directament. La faràs servir molt a l'apartat 5.

Practica 3.4

Predir abans d'executar. Escriu què mostra cada línia.

prediu.js

```
function area(base, altura) {
  return base * altura;
}
const triple = (n) => n * 3;

console.log(area(4, 5));
console.log(triple(area(2, 3)));
console.log(area(2));
```

Línia	Surt
6	`20`
7	
8	

Per què la línia 8 dona això? .....

Practica 3.5

Escriure codi a mà. Escriu una funció `esParell` que rebi un número i torni `true` si és parell i `false` si no. Després, la versió en fletxa.

```
1
2
3
4
```

Comprovació: `esParell(4)` ha de tornar `true` i `esParell(7)`, `false`.

---

4. Objectes: fitxes de dades

Un client no és només un nom. També té una ciutat i un saldo. Podries fer tres arrays, però és fàcil que es desquadrin. Millor posar les dades de cada client **juntes**, en un **objecte** (*object*):

client.js

```
1 const client = { nom: 'Aina', ciutat: 'Palma', saldo: 120 };
2 console.log(client.nom);
3 console.log(client.saldo + 30);
4 client.saldo = 200;
5 console.log(client);
```

```
Aina
150
{ nom: 'Aina', ciutat: 'Palma', saldo: 200 }
```

- Un objecte va entre claus `{ }`. Cada dada és una parella **clau: valor**. La clau és el nom del camp; es diu **propietat** (*property*).
- Es llegeix i es canvia amb un **punt**: `client.nom`, `client.saldo = 200`.

Pensa-hi com en una **fitxa**: una targeta amb uns quants camps omplerts. En aquest llibre els objectes són això, fitxes de dades.

Arrays d'objectes

Una llista de fitxes és un array d'objectes. És la forma que tenen gairebé totes les dades reals: clients, productes, alumnes, comandes.

clients.js

```
const clients = [
  { nom: 'Aina', ciutat: 'Palma', saldo: 120 },
  { nom: 'Nil', ciutat: 'Inca', saldo: -40 },
  { nom: 'Marta', ciutat: 'Palma', saldo: 75 },
];

for (const client of clients) {
  console.log(`${client.nom} (${client.ciutat}): ${client.saldo} €`);
}
```

```
Aina (Palma): 120 €
Nil (Inca): -40 €
Marta (Palma): 75 €
```

Fixa't en la coma després de l'últim objecte. JavaScript l'accepta, i així, si afegeixes un client, no t'oblides de posar-la.

Si demanes un client que no hi és, l'error és aquest:

fora.js

```
const clients = [{ nom: 'Aina' }];
console.log(clients[1].nom);
```

```
/home/alumne/projectes/tema3/fora.js:2
console.log(clients[1].nom);
                       ^

TypeError: Cannot read properties of undefined (reading 'nom')

Node.js v24.21.0
```

Es llegeix així: `clients[1]` és `undefined`, i **d'un** `undefined` **no pots llegir** `nom`. Quan vegis *Cannot read properties of undefined*, mira l'índex o el nom de la variable que hi ha just abans del punt.

Practica 3.6

Fes servir l'array `clients` de `clients.js`. Escriu el resultat.

Expressió	Resultat
`clients[0].nom`	`'Aina'`
`clients[1].saldo`	
`clients.length`	
`clients[2].ciutat`	
`clients[0].saldo + clients[2].saldo`	
`clients[3].nom`	

Practica 3.7

Escriure codi a mà. Escriu un bucle que mostri **només** els clients amb saldo negatiu, amb aquest format: `Nil deu 40 €`. Pista: un `-` davant canvia el signe, i si `client.saldo` val `-40`, `-client.saldo` val `40`.

```
1
2
3
4
5
```

5. Filtrar, transformar i sumar llistes

Els arrays porten **mètodes** que fan les feines de sempre sense escriure el bucle. Tots reben una **funció** (normalment una fletxa) que diu què s'ha de fer amb cada element.

metodes.js

```
const clients = [
  { nom: 'Aina', ciutat: 'Palma', saldo: 120 },
  { nom: 'Nil', ciutat: 'Inca', saldo: -40 },
  { nom: 'Marta', ciutat: 'Palma', saldo: 75 },
];

const deutors = clients.filter((c) => c.saldo < 0);
console.log(deutors);

const noms = clients.map((c) => c.nom);
console.log(noms);

const marta = clients.find((c) => c.nom === 'Marta');
console.log(marta);

const total = clients.reduce((suma, c) => suma + c.saldo, 0);
console.log(total);
```

```
[ { nom: 'Nil', ciutat: 'Inca', saldo: -40 } ]
[ 'Aina', 'Nil', 'Marta' ]
{ nom: 'Marta', ciutat: 'Palma', saldo: 75 }
155
```

Mètode	La funció respon...	Torna
`filter`	Aquest element, **el vull**? `true` o `false`	Un array **nou** amb els que han dit `true`
`map`	**Què poso** al lloc d'aquest element?	Un array nou **de la mateixa llargada**
`find`	**És aquest**?	El **primer** element que diu `true`, o `undefined`
`reduce`	Com **sumo** aquest element al que porto?	**Un sol valor**

`reduce` té dues coses dins: la funció, que rep el que portes acumulat (`suma`) i l'element (`c`), i el valor inicial, el `0` del final. És el mateix que el bucle de la mitjana de l'apartat 2.

Cap d'aquests mètodes canvia l'array original. `clients` continua tenint els tres clients.

Practica 3.8

Predir abans d'executar. Fes servir l'array `clients` de `metodes.js`.

a) `clients.filter((c) => c.ciutat === 'Palma').length` .....

b) `clients.map((c) => c.saldo * 2)` .....

c) `clients.find((c) => c.saldo > 100).nom` .....

d) `clients.find((c) => c.nom === 'Pere')` .....

Practica 3.9

Completar codi. Omple els buits perquè el programa mostri `Palma: 195 €`.

palma.js

```
const dePalma = clients._____((c) => c.ciutat === 'Palma');
const saldoPalma = dePalma._____((suma, c) => suma + c._____, ______);
console.log(`Palma: ${saldoPalma} €`);
```

---

6. De la llista a la pàgina

Fins ara la llista sortia al terminal. Ara la pintaràs **en una pàgina web**, com al tema 1. Crea la carpeta i copia-hi el servidor d'aquell tema:

```
mkdir -p ~/projectes/tema3
cp ~/projectes/hola/servidor.js ~/projectes/tema3/
cd ~/projectes/tema3
```

Aquest `servidor.js` serveix `index.html` i `app.js`. Fes la pàgina. Fixa't que la llista `<ul>` és **buida**: la omplirà el JavaScript.

index.html

```
<!DOCTYPE html>
<html lang="ca">
<head>
  <meta charset="utf-8">
  <title>Clients</title>
</head>
<body>
  <h1>Clients</h1>
  <p id="resum"></p>
  <ul id="llista"></ul>
  <script src="app.js"></script>
</body>
</html>
```

I el JavaScript, que ara s'executa **al navegador**:

app.js

```
const clients = [
  { nom: 'Aina', ciutat: 'Palma', saldo: 120 },
  { nom: 'Nil', ciutat: 'Inca', saldo: -40 },
  { nom: 'Marta', ciutat: 'Palma', saldo: 75 },
];

const llista = document.getElementById('llista');

for (const client of clients) {
  const element = document.createElement('li');
  element.textContent = `${client.nom} (${client.ciutat}): ${client.saldo} €`;
  llista.appendChild(element);
}

const resum = document.getElementById('resum');
resum.textContent = `${clients.length} clients`;
```

Engega `node servidor.js`. Si a la pestanya **Ports** de VS Code no hi ha el 3000, reenvia'l com al tema 1, i obre `http://localhost:3000` al navegador del portàtil. Surt el títol, `3 clients` i una llista amb tres punts.

El navegador guarda la pàgina com un **arbre** d'elements, el **DOM** (*Document Object Model*). `document` és la porta d'entrada a aquest arbre. Tres ordres fan tota la feina:

Ordre	Què fa
`document.getElementById('llista')`	Busca l'element que té `id="llista"`
`document.createElement('li')`	Crea un element `<li>` nou. **Encara no es veu**: no és a l'arbre
`llista.appendChild(element)`	El posa a dins de `llista`, al final. **Ara sí que es veu**

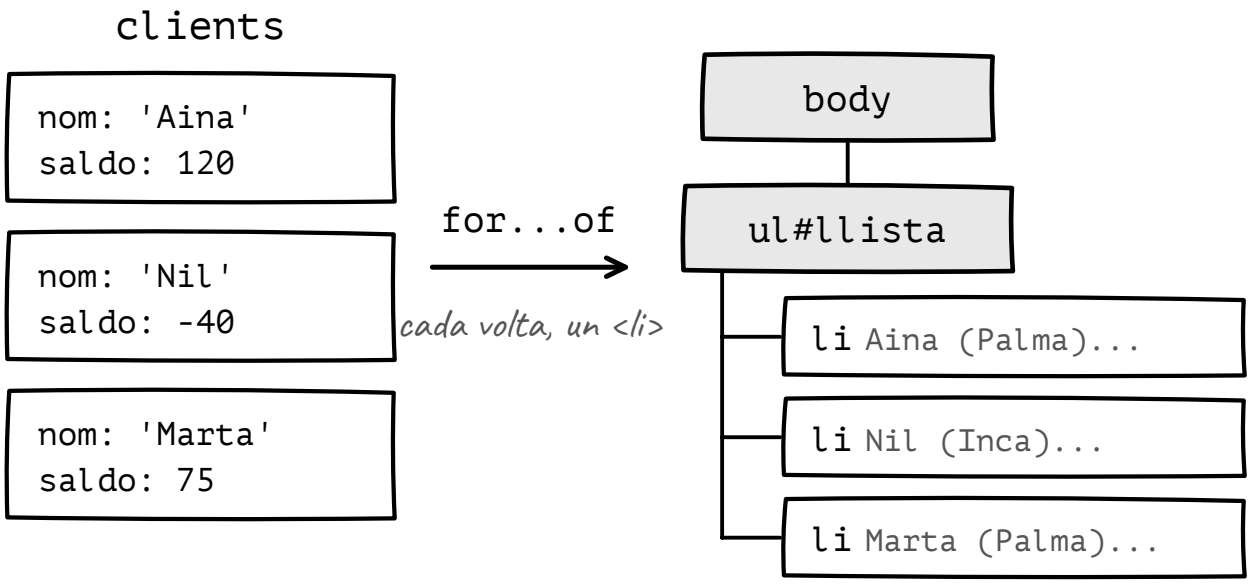


FIG 3.3 Cada volta del bucle crea un `<li>` i el penja de la llista.

Mira la pàgina amb F12, a la pestanya **Elements**. Dins del `<ul id="llista">` hi ha tres `<li>`: **no són a l'**`index.html`, els ha posat el bucle.

Practica 3.10

Predir abans d'executar. Afegeix un quart client a l'array: `{ nom: 'Pere', ciutat: 'Sóller', saldo: 0 },`. Sense tocar res més:

a) Quants `<li>` hi haurà dins la llista? .....

b) Què posarà el paràgraf `resum`? .....

c) Què diu el quart `<li>`? .....

Practica 3.11

Arbre DOM. Dibuixa com queda l'arbre del `<body>` **després** d'executar `app.js`, amb els tres clients originals. Posa a cada `<li>` el seu text.

Practica 3.12

Escriure codi a mà. Canvia el bucle d'`app.js` perquè només pinti els clients **de Palma**, i el resum digui `2 clients de Palma`. Fes servir `filter`.

```
1
2
3
4
5
6
7
8
```

Guiat 3.13

Una funció per pintar. Quan un tros de codi fa una feina concreta, posa-li nom. Completa la funció que pinta **un** client i la crida per a cadascun.

app.js

```
function pintaClient(client) {
  const element = document._____________('li');
  element.textContent = `${client.nom}: ${client.saldo} €`;
  if (client.saldo < 0) {
    element.textContent = element.textContent + ' (deutor)';
  }
  llista._____________(element);
}

for (const client of clients) {
  _____________________;
}
```

Amb els tres clients, què diu el segon `<li>`? .....

El que has de saber sí o sí

- Un **array** és una llista. El **primer índex és el 0** i l'últim, `length - 1`.
- `push` afegeix al final i `pop` treu l'últim. Un array `const` es pot canviar per dins.
- Per recórrer: `for (let i = 0; i < llista.length; i++)` si et cal l'índex, `for...of` si no.
- Una **funció** es defineix una vegada i es crida moltes. El **paràmetre** val el que li passes.
- `return` **torna un valor** i acaba la funció. `console.log` només el mostra. Sense `return`, torna `undefined`.
- Una fletxa curta, `(n) => n * 2`, torna el resultat sense escriure `return`.

- Un **objecte** és una fitxa de dades: `{ nom: 'Aina', saldo: 120 }`. Es llegeix amb punt: `client.nom`.
- *Cannot read properties of undefined*: el que hi ha abans del punt no existeix.
- `filter` es queda amb uns quants, `map` els transforma tots, `find` en busca un i `reduce` els suma en un valor.
- A la pàgina: `getElementById` busca, `createElement` crea i `appendChild` **el fa aparèixer**.

Paraules noves

Paraula	Què és	Les meves notes
Array	Llista de valors en una sola variable.	
Índex *(index)*	Posició d'un element dins d'un array. Comença pel 0.	
Funció *(function)*	Tros de codi amb nom que es pot cridar moltes vegades.	
Paràmetre *(parameter)*	Variable d'una funció que val el que li passes en cridar-la.	
Valor de retorn	El que torna una funció amb `return`.	
Funció fletxa *(arrow function)*	Forma curta d'escriure una funció: `(n) => n * 2`.	
Objecte *(object)*	Conjunt de parelles clau: valor. Aquí, una fitxa de dades.	
Propietat *(property)*	Cada camp d'un objecte: `nom`, `saldo`…	
Mètode	Funció que porta un valor: `push`, `filter`, `map`…	
DOM *(Document Object Model)*	L'arbre d'elements de la pàgina que JavaScript pot llegir i canviar.	

Repàs del tema

A. Vertader o fals

Frase	V o F	Per què
El primer element d'un array és `llista[1]`		
Si un array té `length` 5, l'últim és `llista[5]`		
A un array `const` no s'hi pot fer `push`		
Una funció sense `return` torna `undefined`		
`filter` canvia l'array original		
Un `<li>` fet amb `createElement` es veu abans de fer `appendChild`		

B. Preguntes curtes

1. Quina diferència hi ha entre `map` i `filter`? Posa'n un exemple de cada.

2. Per què és millor guardar un client en un objecte que en tres arrays separats de noms, ciutats i saldos?

3. Obres la pàgina i la llista surt buida. A la consola hi ha *Cannot read properties of null (reading 'appendChild')*. Què pot passar?

C. Problemes

Problema 1. Escriu una funció `totalCarro` que rebi un array de productes com aquest i torni el total a pagar. Fes servir `reduce`.

carro.js

```
const carro = [
  { nom: 'Samarreta', preu: 15, quantitat: 2 },
  { nom: 'Gorra', preu: 8, quantitat: 1 },
];
```

```

```

Comprovació: amb aquest carro ha de tornar `38`.

Problema 2. Aquest `alumnes.js`, carregat des d'una pàgina amb un `<ul id="llista">`, hauria de pintar tres alumnes. La pàgina surt sense cap nom i a la consola no hi ha cap error. Troba què falta i digues on va.

alumnes.js

```
const alumnes = ['Aina', 'Nil', 'Marta'];
const llista = document.getElementById('llista');
for (const alumne of alumnes) {
  const element = document.createElement('li');
  element.textContent = alumne;
}
```

Les meves notes