Apunts de JavaScript - Conceptes Avançats

1. undefined, null i NaN

undefined

Valor que té una variable quan ha estat declarada però no se li ha assignat cap valor.

```javascript
let x;
console.log(x); // undefined
console.log(typeof undefined); // "undefined"
```

null

Representa l'absència intencionada d'un valor. S'ha d'assignar explícitament.

```javascript
let y = null;
console.log(y); // null
console.log(typeof null); // "object" (bug històric de JS)
```

NaN (Not a Number)

Resultat d'operacions matemàtiques invàlides.

```javascript
console.log(0 / 0); // NaN
console.log(parseInt("hola")); // NaN
console.log(isNaN(NaN)); // true
```

2. Funcions

Declaració de funcions

```javascript
// Funció declarada
function sumar(a, b) {
    return a + b;
}

// Funció expressió
const restar = function(a, b) {
    return a - b;
};
```

Paràmetres per defecte

```javascript
function saludar(nom = "Anònim") {
    return `Hola, ${nom}`;
}
```

3. Funcions de Fletxa (Arrow Functions)

Sintaxi més concisa per crear funcions.

```javascript
// Sintaxi completa
const multiplicar = (a, b) => {
    return a * b;
};

// Sintaxi curta (return implícit)
const dividir = (a, b) => a / b;

// Un sol paràmetre (sense parèntesis)
const quadrat = x => x * x;

// Sense paràmetres
const obtenerData = () => new Date();
```

Diferències amb funcions normals

- No tenen el seu propi `this`
- No es poden utilitzar com a constructors
- No tenen l'objecte `arguments`

4. Arrays

Creació i accés

```javascript
const fruits = ["poma", "plàtan", "taronja"];
console.log(fruits[0]); // "poma"
console.log(fruits.length); // 3
```

Mètodes principals

```javascript
// Afegir/eliminar elements
fruits.push("maduixa"); // Afegeix al final
fruits.pop(); // Elimina l'últim
fruits.unshift("kiwi"); // Afegeix al principi
fruits.shift(); // Elimina el primer

// Cercar elements
fruits.indexOf("plàtan"); // 1
fruits.includes("poma"); // true

// Transformacions
const numeros = [1, 2, 3, 4, 5];
const doblesNum = numeros.map(n => n * 2); // [2, 4, 6, 8, 10]
const parell = numeros.filter(n => n % 2 === 0); // [2, 4]
const suma = numeros.reduce((acc, n) => acc + n, 0); // 15

// Altres mètodes útils
fruits.slice(1, 3); // Retorna subcadena
fruits.splice(1, 1, "mango"); // Modifica l'array
fruits.forEach(fruit => console.log(fruit));
fruits.find(fruit => fruit === "poma");
fruits.some(fruit => fruit.length > 5);
fruits.every(fruit => typeof fruit === "string");
```

5. Objectes Literals

Creació i propietats

javascript

```javascript
const persona = {
    nom: "Joan",
    edat: 25,
    ciutat: "Barcelona",
    saludar: function() {
        return `Hola, sóc ${this.nom}`;
    }
};

// Accedir a propietats
console.log(persona.nom); // Notació de punt
console.log(persona["edat"]); // Notació de claudàtors

// Afegir/modificar propietats
persona.professio = "Programador";
persona.edat = 26;

// Eliminar propietats
delete persona.ciutat;
```

Mètodes d'objectes

javascript

```javascript
const claus = Object.keys(persona); // ["nom", "edat", "professio"]
const valors = Object.values(persona); // ["Joan", 26, "Programador"]
const entrades = Object.entries(persona); // [["nom", "Joan"], ...]

// Copiar objectes
const copia = Object.assign({}, persona);
const copia2 = { ...persona }; // Spread operator
```

Shorthand i computed properties

javascript

```javascript
const nom = "Maria";
const edat = 30;

// Property shorthand
const usuari = { nom, edat };

// Computed property names
const clau = "color";
const objecte = {
  [clau]: "blau",
  [`${clau}Secundari`]: "verd"
};
```

6. Maneig d'Errors

try...catch...finally

javascript

```javascript
try {
  // Codi que pot generar errors
  const resultat = funcioPerillosa();
  console.log(resultat);
} catch (error) {
  // Gestió de l'error
  console.error("S'ha produït un error:", error.message);
} finally {
  // S'executa sempre
  console.log("Procés finalitzat");
}
```

Llançar errors personalitzats

```javascript
function dividir(a, b) {
  if (b === 0) {
    throw new Error("No es pot dividir per zero");
  }
  return a / b;
}

try {
  dividir(10, 0);
} catch (e) {
  console.error(e.message);
}
```

7. break i continue

break

Surt del bucle completament.

```javascript
for (let i = 0; i < 10; i++) {
  if (i === 5) {
    break; // Surt del bucle
  }
  console.log(i); // 0, 1, 2, 3, 4
}
```

continue

Salta a la següent iteració.

```javascript
for (let i = 0; i < 5; i++) {
  if (i === 2) {
    continue; // Salta la iteració
  }
  console.log(i); // 0, 1, 3, 4
}
```

8. Desestructuració

Desestructuració d'arrays

```javascript
const colors = ["vermell", "verd", "blau"];
const [primer, segon, tercer] = colors;

// Saltar elements
const [, , tercerColor] = colors;

// Rest operator
const [cap, ...resta] = colors;
console.log(resta); // ["verd", "blau"]
```

Desestructuració d'objectes

```javascript
const persona = {
    nom: "Anna",
    edat: 28,
    ciutat: "València"
};

const { nom, edat } = persona;

// Renombrar variables
const { nom: nomComplet, edat: anys } = persona;

// Valors per defecte
const { pais = "Espanya" } = persona;

// Desestructuració niuada
const usuari = {
    id: 1,
    dades: {
        nom: "Pere",
        email: "pere@example.com"
    }
};

const { dades: { nom, email } } = usuari;
```

9. Objecte console

Mètodes útils per depurar.

javascript

```javascript
console.log("Missatge normal");
console.error("Missatge d'error");
console.warn("Advertència");
console.info("Informació");

// Taules
console.table([{nom: "Joan", edat: 25}, {nom: "Maria", edat: 30}]);

// Agrupació
console.group("Grup 1");
console.log("Dins del grup");
console.groupEnd();

// Temps d'execució
console.time("operacio");
// ... codi ...
console.timeEnd("operacio");

// Comptar
console.count("comptador");
console.count("comptador"); // comptador: 2

// Netejar consola
console.clear();
```

10. Objecte Date

Treballar amb dates i hores.

javascript

```javascript
// Crear dates
const ara = new Date();
const dataEspecifica = new Date(2024, 0, 15); // 15 de gener de 2024
const desdeString = new Date("2024-01-15");

// Obtenir components
ara.getFullYear(); // Any
ara.getMonth(); // Mes (0-11)
ara.getDate(); // Dia del mes
ara.getDay(); // Dia de la setmana (0-6)
ara.getHours();
ara.getMinutes();
ara.getSeconds();
ara.getTime(); // Millisegons des de 1970

// Modificar dates
ara.setFullYear(2025);
ara.setMonth(5);
ara.setDate(20);

// Formatar dates
ara.toLocaleDateString("ca-ES");
ara.toLocaleTimeString("ca-ES");
ara.toISOString();
```

11. Objecte Math

Operacions matemàtiques.

```javascript
// Constants
Math.PI; // 3.141592653589793
Math.E; // 2.718281828459045

// Arrodoniment
Math.round(4.7); // 5
Math.ceil(4.1); // 5 (cap amunt)
Math.floor(4.9); // 4 (cap avall)
Math.trunc(4.9); // 4 (elimina decimals)

// Operacions
Math.abs(-5); // 5
Math.pow(2, 3); // 8
Math.sqrt(16); // 4
Math.max(1, 5, 3); // 5
Math.min(1, 5, 3); // 1

// Aleatori
Math.random(); // Entre 0 i 1
Math.floor(Math.random() * 10); // Entre 0 i 9
Math.floor(Math.random() * (max - min + 1)) + min; // Entre min i max

// Trigonometria
Math.sin(Math.PI / 2);
Math.cos(0);
Math.tan(Math.PI / 4);
```

12. Operador de Curtcircuit

AND lògic (&&)

Retorna el primer valor falsy o l'últim valor.

```javascript
const resultat = valor1 && valor2;

// Exemple pràctic
const usuari = { nom: "Joan" };
const nomUsuari = usuari && usuari.nom; // "Joan"

// Execució condicional
isLoggedIn && mostrarPerfil();
```

OR lògic (||)

Retorna el primer valor truthy o l'últim valor.

```javascript
const nom = nomUsuari || "Anònim";
const port = process.env.PORT || 3000;
```

Nullish coalescing (??)

Similar a ||, però només considera null i undefined com falsy.

```javascript
const valor = 0;
const resultat1 = valor || 10; // 10
const resultat2 = valor ?? 10; // 0
```

13. alert, confirm i prompt

alert

Mostra un missatge a l'usuari.

```javascript
alert("Benvingut a la pàgina!");
```

confirm

Demana confirmació (OK/Cancel).

```javascript
const confirmacio = confirm("Estàs segur?");
if (confirmacio) {
  console.log("Usuari ha confirmat");
}
```

prompt

Sol·licita entrada de text.

```javascript
const nom = prompt("Quin és el teu nom?", "Valor per defecte");
if (nom !== null) {
  console.log(`Hola, ${nom}`);
}
```

14. Temporitzadors

setTimeout

Executa una funció després d'un temps.

```javascript
const tempId = setTimeout(() => {
    console.log("Executat després de 2 segons");
}, 2000);

// Cancel·lar
clearTimeout(tempId);
```

setInterval

Executa una funció repetidament.

```javascript
let comptador = 0;
const intervalId = setInterval(() => {
    comptador++;
    console.log(comptador);
    if (comptador === 5) {
        clearInterval(intervalId);
    }
}, 1000);
```

15. DOM - Introducció

El DOM (Document Object Model) és una representació en forma d'arbre dels elements HTML.

```
Document
└── html
    ├── head
    │   ├── title
    │   └── meta
    └── body
        ├── header
        │   └── h1
        ├── main
        │   ├── section
        │   └── article
        └── footer
```

Exemple bàsic

```javascript
// Accedir al document
console.log(document.title);
console.log(document.body);
console.log(document.documentElement); // <html>
```

16. DOM - Crear Elements

```javascript
// Crear element
const nouDiv = document.createElement("div");
const nouText = document.createTextNode("Contingut");
const nouFragment = document.createDocumentFragment();

// Exemple complet
const article = document.createElement("article");
article.className = "post";

const titol = document.createElement("h2");
titol.textContent = "Títol de l'article";

article.appendChild(titol);
document.body.appendChild(article);
```

17. DOM - Atributs

Afegir atributs

```javascript
const imatge = document.createElement("img");

// setAttribute
imatge.setAttribute("src", "imatge.jpg");
imatge.setAttribute("alt", "Descripció");
imatge.setAttribute("width", "300");

// Propietats directes
imatge.src = "imatge.jpg";
imatge.alt = "Descripció";
```

Accedir als atributs

```javascript
// getAttribute
const src = imatge.getAttribute("src");
const alt = imatge.getAttribute("alt");

// Propietats directes
const src2 = imatge.src;
const alt2 = imatge.alt;

// Comprovar existència
const teAlt = imatge.hasAttribute("alt"); // true
```

Eliminar atributs

```javascript
imatge.removeAttribute("width");
```

18. DOM - Afegir Contingut

innerText

Afegeix text pla (respecta el CSS visibility).

```javascript
const parrafo = document.createElement("p");
parrafo.innerText = "Aquest és el contingut de text";
```

textContent

Similar a innerText però més ràpid i no respecta CSS.

```javascript
parrafo.textContent = "Contingut de text";
```

innerHTML

Afegeix HTML (pot ser perillós amb dades d'usuari).

```javascript
const contenidor = document.getElementById("contenidor");
contenidor.innerHTML = "<h2>Títol</h2><p>Paràgraf</p>";

// PERILL: XSS vulnerability
const userInput = "<img src=x onerror=alert('XSS')>";
contenidor.innerHTML = userInput; // NO FER AIXÒ!
```

19. DOM - Manipulació de Nodes

removeChild

Elimina un fill d'un node.

```javascript
const pare = document.getElementById("contenidor");
const fill = document.getElementById("element");
pare.removeChild(fill);

// Mètode modern
fill.remove();
```

replaceChild

Substitueix un node fill per un altre.

```javascript
const nouElement = document.createElement("div");
nouElement.textContent = "Nou contingut";
pare.replaceChild(nouElement, fill);
```

insertBefore

Insereix un node abans d'un altre.

```javascript
const referencia = document.getElementById("referencia");
const nouNode = document.createElement("p");
pare.insertBefore(nouNode, referencia);
```

hasChildNodes

Comprova si un node té fills.

```javascript
if (pare.hasChildNodes()) {
    console.log("El node té fills");
}
```

20. DOM - Mètodes Moderns d'Inserció

prepend

Afegeix al principi del contenidor.

```javascript
const contenidor = document.getElementById("contenidor");
const nouElement = document.createElement("h1");
nouElement.textContent = "Títol principal";
contenidor.prepend(nouElement);
```

append

Afegeix al final del contenidor.

```javascript
contenidor.append(nouElement);
```

after

Afegeix després de l'element.

```javascript
const element = document.getElementById("element");
const nouElement = document.createElement("div");
element.after(nouElement);
```

before

Afegeix abans de l'element.

```javascript
element.before(nouElement);
```

insertAdjacentElement

Insereix element en posició específica.

```javascript
const element = document.getElementById("element");
const nou = document.createElement("div");

element.insertAdjacentElement("beforebegin", nou); // Abans de l'element
element.insertAdjacentElement("afterbegin", nou);  // Primer fill
element.insertAdjacentElement("beforeend", nou);   // Últim fill
element.insertAdjacentElement("afterend", nou);    // Després de l'element
```

insertAdjacentHTML

Insereix HTML en posició específica.

```javascript
element.insertAdjacentHTML("beforebegin", "<p>Abans</p>");
element.insertAdjacentHTML("afterbegin", "<p>Primer fill</p>");
element.insertAdjacentHTML("beforeend", "<p>Últim fill</p>");
element.insertAdjacentHTML("afterend", "<p>Després</p>");
```

insertAdjacentText

Insereix text en posició específica.

```javascript
element.insertAdjacentText("beforeend", "Text al final");
```

21. Emmagatzematge Local

localStorage

Emmagatzema dades sense data de caducitat.

javascript

```javascript
// Guardar dades
localStorage.setItem("usuari", "Joan");
localStorage.setItem("preferencies", JSON.stringify({ tema: "fosc" }));

// Llegir dades
const usuari = localStorage.getItem("usuari");
const prefs = JSON.parse(localStorage.getItem("preferencies"));

// Eliminar dades
localStorage.removeItem("usuari");
localStorage.clear(); // Elimina tot

// Comprovar existència
if (localStorage.getItem("usuari")) {
  console.log("Usuari trobat");
}
```

sessionStorage

Similar a localStorage però s'elimina en tancar la pestanya.

javascript

```javascript
sessionStorage.setItem("sessio", "12345");
const sessio = sessionStorage.getItem("sessio");
sessionStorage.removeItem("sessio");
sessionStorage.clear();
```

Cookies

Emmagatzemen dades amb més control (data de caducitat, path, domain).

javascript

```javascript
// Crear cookie
document.cookie = "usuari=Joan; max-age=3600; path=/";

// Llegir cookies
const cookies = document.cookie.split("; ");
cookies.forEach(cookie => {
  const [nom, valor] = cookie.split("=");
  console.log(`${nom}: ${valor}`);
});

// Eliminar cookie (establir data passada)
document.cookie = "usuari=; max-age=0";
```

22. Selectors d'Elements

querySelector

Retorna el primer element que coincideix.

```javascript
const element = document.querySelector(".classe");
const element2 = document.querySelector("#id");
const element3 = document.querySelector("div > p");
const element4 = document.querySelector('[data-atribut="valor"]');
```

querySelectorAll

Retorna tots els elements que coincideixen (NodeList).

```javascript
const elements = document.querySelectorAll(".classe");
elements.forEach(el => {
  console.log(el);
});

// Convertir a array
const arrayElements = Array.from(elements);
const arrayElements2 = [...elements];
```

getElementById

Retorna l'element amb l'ID especificat.

```javascript
const element = document.getElementById("meu-id");
// Més ràpid que querySelector("#meu-id")
```

getElementsByClassName

Retorna una HTMLCollection d'elements amb la classe.

```javascript
const elements = document.getElementsByClassName("classe");
// HTMLCollection (no és un array)
const arrayElements = Array.from(elements);
```

getElementsByTagName

Retorna elements pel nom de l'etiqueta.

```javascript
const paragrafs = document.getElementsByTagName("p");
const tots = document.getElementsByTagName("*"); // Tots els elements

// Recórrer
for (let p of paragrafs) {
  console.log(p.textContent);
}
```

Diferències entre NodeList i HTMLCollection

```javascript
// NodeList (querySelectorAll) - estàtica
const nodeList = document.querySelectorAll("p");
nodeList.forEach(node => console.log(node)); // Té forEach

// HTMLCollection (getElementsBy...) - dinàmica
const htmlCollection = document.getElementsByTagName("p");
// No té forEach directament
Array.from(htmlCollection).forEach(el => console.log(el));
```

23. Comunicació Asíncrona

Callbacks

Funció que es passa com a argument per executar-se després.

```javascript
function descarregarDades(callback) {
  setTimeout(() => {
    const dades = { nom: "Joan", edat: 25 };
    callback(dades);
  }, 1000);
}

descarregarDades(dades => {
  console.log(dades);
});
```

Promises

Objecte que representa una operació asíncrona.

javascript

```javascript
const promesa = new Promise((resolve, reject) => {
  setTimeout(() => {
    const exit = Math.random() > 0.5;
    if (exit) {
      resolve("Operació exitosa");
    } else {
      reject("Error en l'operació");
    }
  }, 1000);
});

promesa
  .then(resultat => console.log(resultat))
  .catch(error => console.error(error))
  .finally(() => console.log("Finalitzat"));
```

async/await

Sintaxi més neta per treballar amb Promises.

javascript

```javascript
async function obtenerDades() {
  try {
    const resposta = await fetch("https://api.example.com/dades");
    const dades = await resposta.json();
    return dades;
  } catch (error) {
    console.error("Error:", error);
  }
}

// Utilitzar
obtenerDades().then(dades => console.log(dades));
```

Fetch API

Mètode modern per fer peticions HTTP.

```javascript
// GET
fetch("https://api.example.com/usuaris")
  .then(res => res.json())
  .then(dades => console.log(dades))
  .catch(error => console.error(error));

// POST
fetch("https://api.example.com/usuaris", {
  method: "POST",
  headers: {
    "Content-Type": "application/json"
  },
  body: JSON.stringify({
    nom: "Joan",
    email: "joan@example.com"
  })
})
  .then(res => res.json())
  .then(dades => console.log(dades));

// Amb async/await
async function crearUsuari(usuari) {
  const resposta = await fetch("https://api.example.com/usuaris", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify(usuari)
  });
  return await resposta.json();
}
```

Promise.all

Espera que totes les promises es completin.

```javascript
const promesa1 = fetch("https://api.example.com/usuari/1");
const promesa2 = fetch("https://api.example.com/usuari/2");
const promesa3 = fetch("https://api.example.com/usuari/3");

Promise.all([promesa1, promesa2, promesa3])
  .then(respostes => Promise.all(respostes.map(r => r.json())))
  .then(dades => console.log(dades))
  .catch(error => console.error("Error en alguna petició"));
```

Promise.race

Retorna el resultat de la primera promise que es completa.

```javascript
const promesa1 = new Promise(resolve => setTimeout(() => resolve("Primera"), 500));
const promesa2 = new Promise(resolve => setTimeout(() => resolve("Segona"), 300));

Promise.race([promesa1, promesa2])
  .then(resultat => console.log(resultat)); // "Segona"
```

Aquests apunts cobreixen els conceptes fonamentals i avançats de JavaScript necessaris per desenvolupar aplicacions web modernes.