

Introduktion till ES6

Vad är ES6?

- ES6 (ECMAScript 2015) är en stor uppdatering av JavaScript som introducerade nya funktioner och förbättringar.
- Gör koden mer läsbar, effektiv och lättare att underhålla.
- Stöd för modern syntax som gör utveckling i React och TypeScript smidigare.

Viktiga nyheter i ES6

let och const

- **let** : Blockscope-variabel som kan om-assignas.
- **const** : Blockscope-variabel som inte kan om-assignas.

```
let name = "Lisa";  
name = "Anna"; // Ok
```

```
const age = 25;  
age = 30; // ✗ Error
```

Arrow Functions

- Kortare syntax för funktioner.
- **Implicit return** vid enkel logik.

```
const sum = (a, b) => a + b;  
const sayHello = (name) => `Hej ${name}`;
```

Template Literals

- Använd **backticks** (`) för stränginterpolation och flerradiga strängar.

```
const name = "Lisa";  
const greeting = `Hej, ${name}!`; // "Hej, Lisa!"  
const message = `Hej!  
Det här är en sträng  
som sträcker sig över flera rader.`;
```

Destructuring

- Extrahera värden från objekt och arrayer på ett smidigt sätt.

```
const cat = { name: "Misse", birthyear: 2020 };  
const { name, birthyear } = cat;  
console.log(name); // "Misse"
```

Default Parameters

- Ange standardvärden för funktionens parametrar.

```
const greet = (name = "Gäst") => `Hej, ${name}!`;
console.log(greet()); // "Hej, Gäst!"
console.log(greet("Lisa")); // "Hej, Lisa!"
```


Reduce

- `reduce()` används för att "sammanfatta" alla värden i en array till ett enda värde.
- Exempel: summera alla tal i en array

```
const numbers = [1, 2, 3, 4, 5];  
const sum = numbers.reduce((acc, num) => acc + num, 0);  
console.log(sum); // 15
```

Reduce

- Hitta största talet i en array

```
const numbers = [10, 25, 8, 42];  
const max = numbers.reduce((acc, num) => (num > acc ? num : acc), numbers[0]);  
console.log(max); // 42
```

Rest och Spread Operators

- Rest (`...`) samlar ihop argument i en array.
- Spread (`...`) sprider ut array-element eller objekt.

```
const sum = (...numbers) => numbers.reduce((acc, num) => acc + num, 0);  
console.log(sum(1, 2, 3)); // 6
```

```
const arr1 = [1, 2, 3];  
const arr2 = [...arr1, 4, 5];  
console.log(arr2); // [1, 2, 3, 4, 5]
```

map(): Transformera varje element

- `map()` skapar en **ny array** genom att applicera en funktion på varje element.

```
const cats = [  
  { name: "Misse", birthyear: 2020 },  
  { name: "John", birthyear: 2013 },  
  { name: "Monica", birthyear: 2017 },  
];  
  
const catDescriptions = cats.map(  
  (cat) => `${cat.name} är född ${cat.birthyear}`  
);  
console.log(catDescriptions);  
// ["Misse är född 2020", "John är född 2013", "Monica är född 2017"]
```

filter(): Filtrera ut element som uppfyller ett villkor

- `filter()` returnerar en **ny array** med de element som uppfyller ett visst villkor.
- Exempel: Hämta ut katter födda 2017 eller senare

```
const cats = [  
  { name: "Misse", birthyear: 2020 },  
  { name: "John", birthyear: 2013 },  
  { name: "Monica", birthyear: 2017 },  
];  
const youngCats = cats.filter((cat) => cat.birthyear >= 2017);  
console.log(youngCats);  
// [{ name: "Misse", birthyear: 2020 }, { name: "Monica", birthyear: 2017 }]
```

sort(): Sortera en array

- `sort()` sorterar elementen i en array.
- Skapar inte en ny array
- Exempel: Sortera katter från äldst till yngst

```
const sortedCats = cats.sort((a, b) => a.birthyear - b.birthyear);
console.log(sortedCats);
// [
//   { name: "John", birthyear: 2013 },
//   { name: "Monica", birthyear: 2017 },
//   { name: "Misse", birthyear: 2020 }
// ]
```

Export och Import i ES6

- ES6 gör det möjligt att **exportera och importera kod** mellan filer.
- Hjälper till att strukturera och återanvända kod.

Named Exports

- Används för att **exportera flera funktioner eller variabler** från en fil.
- Importen måste använda exakt samma namn som exporten.

Exempel: Named Export

```
// utils.js
export const sum = (a, b) => a + b;
export const multiply = (a, b) => a * b;
```

Exempel: Named Import

```
// main.js
import { sum, multiply } from "./utils.js";

console.log(sum(2, 3)); // 5
console.log(multiply(2, 3)); // 6
```


Default Export

- En fil kan ha en default-export.
- Vid import behöver man inte använda `{}`.

Exempel: Default Export

```
// math.js
export default function divide(a, b) {
  return a / b;
}
```

Exempel: Default Import

```
// main.js
import divide from "./math.js";

console.log(divide(10, 2)); // 5
```

När ska man använda Named vs. Default Export?

Typ	Fördelar	När ska man använda?
Named Export	Exporterar flera funktioner.	När modulen innehåller flera funktioner eller variabler .
Default Export	Enklare import, inget <code>{}</code> .	När modulen bara har en huvudfunktion eller klass .

Promises i JavaScript

- **Promises** används för att hantera asynkrona operationer.
- Ett **Promise** kan ha tre tillstånd:
 - **Pending** (väntande)
 - **Resolved** (uppfyllt, lyckades)
 - **Rejected** (avvisat, misslyckades)

```
const fetchData = new Promise((resolve, reject) => {
  setTimeout(() => {
    const success = true;
    if (success) {
      resolve("Data hämtad!");
    } else {
      reject("Fel vid hämtning av data!");
    }
  }, 2000);
});
```

```
console.log(fetchData); // Pending (väntande)
```

Använda `then` och `catch` för att hantera Promises

- `then()` körs om Promiset lyckas (**fulfilled**).
- `catch()` körs om Promiset misslyckas (**rejected**).
- Exempel: Använda `then()` och `catch()`

```
fetchData
  .then((response) => {
    console.log(response); // "Data hämtad!"
  })
  .catch((error) => {
    console.error(error); // "Fel vid hämtning av data!"
  });
```

ES8: `async/await` – Enklare syntax för att hantera Promises

- `async` gör att en funktion returnerar ett Promise.
- `await` väntar på att ett Promise ska bli klart innan koden fortsätter.
- Exempel: Samma Promise med `async/await`

```
async function getData() {  
  try {  
    const response = await fetchData;  
    console.log(response); // "Data hämtad!"  
  } catch (error) {  
    console.error(error); // "Fel vid hämtning av data!"  
  }  
}  
  
getData();
```

Exempel: Hämta data från ett API med `fetch()`

- `fetch()` returnerar ett Promise som löser sig när data är hämtad.

```
async function fetchUser() {  
  try {  
    const response = await fetch(  
      "https://jsonplaceholder.typicode.com/users/1"  
    );  
    const user = await response.json();  
    console.log(user); // Visar användarens data  
  } catch (error) {  
    console.error("Fel vid hämtning:", error);  
  }  
}  
  
fetchUser();
```

Allmänt om DOM-manipulation i JavaScript

Ändra element

```
// Ändra textinnehåll
document.getElementById("demo").textContent = "Ny text!";

// Ändra stil
document.getElementById("demo").style.color = "blue";
```

Lägga till en Event Listener

```
// Lägg till en klickhändelse
const button = document.getElementById("myButton");
button.addEventListener("click", () => {
    alert("Knappen klickades!");
});
```