Natuurlijk! Hier is een spoedcursus JavaScript die je in een korte tijd kan helpen de basisprincipes van de taal te begrijpen. Ik zal de belangrijkste concepten snel en duidelijk uitleggen, met voorbeelden. Laten we beginnen!

1. Wat is JavaScript?

JavaScript is een programmeertaal die voornamelijk wordt gebruikt voor interactieve elementen op webpagina's. Het maakt het mogelijk om dynamische en interactieve webpagina's te maken, zoals formulieren, animaties, en de interactie met servers.

2. Basisstructuur van een JavaScript-programma

JavaScript-code kan in een webpagina worden geplaatst tussen `<script>` tags:

```
html
Code kopiëren
<!DOCTYPE html>
<html>
<head>
    <title>JavaScript Cursus</title>
</head>
<body>
    <script>
        console.log("Hallo, wereld!");
    </script>
</body>
</html>
```

De bovenstaande code toont de tekst "Hallo, wereld!" in de browserconsole.

3. Variabelen

In JavaScript kun je variabelen declareren met `let`, `const` of `var`. Het belangrijkste verschil is de scope (bereik) van de variabelen en hun herbruikbaarheid.

- **`let`**: Kan opnieuw worden toegewezen, en heeft block-scoping.
- **`const`**: Kan niet opnieuw worden toegewezen.
- **`var`**: Oudere manier, met function-scoping (meestal wordt `let` of `const` aanbevolen).

Voorbeelden:

```
javascript
Code kopiëren
let naam = "John";
const leeftijd = 30;

console.log(naam);     // Output: John
console.log(leeftijd); // Output: 30

naam = "Jane";         // Mag omdat 'let' wordt gebruikt
console.log(naam);     // Output: Jane

// leeftijd = 31;      // Fout! 'const' kan niet opnieuw worden toegewezen
```

4. Datatypes

JavaScript heeft verschillende datatypes:

- **String**: Een tekstwaarde, bijvoorbeeld `"Hallo"`.
- **Number**: Numerieke waarden, bijvoorbeeld `123` of `3.14`.
- **Boolean**: Waar (`true`) of niet waar (`false`).
- **Object**: Een verzameling van sleutel-waarde paren.
- **Array**: Een lijst van waarden.
- **Null**: Een expliciete lege waarde.
- **Undefined**: Wanneer een variabele geen waarde heeft.

Voorbeelden:

```
javascript
Code kopiëren
let naam = "Jane";        // String
let leeftijd = 30;        // Number
let isStudent = true;     // Boolean
let persoon = { naam: "John", leeftijd: 30 };  // Object
let nummers = [1, 2, 3];  // Array
let niets = null;         // Null
let onbekend;             // Undefined
```

5. Operatoren

JavaScript ondersteunt verschillende operatoren:

- **Rekenkundige operatoren**: `+`, `-`, `*`, `/`, `%` (modulo)
- **Vergelijkingsoperatoren**: `==`, `===` (strikt gelijk), `!=`, `<`, `>`, `<=`, `>=`
- **Logische operatoren**: `&&` (EN), `||` (OF), `!` (NIET)

Voorbeeld:

```
javascript
Code kopiëren
let x = 10;
let y = 5;

console.log(x + y); // Output: 15
console.log(x === y); // Output: false (strikt gelijk)
console.log(x > y && y < 10); // Output: true (EN operator)
```

6. Functies

Functies worden gebruikt om blokken code te hergebruiken. Je definieert een functie met `function`.

Voorbeeld:

```
javascript
Code kopiëren
function greet(naam) {
    console.log("Hallo, " + naam + "!");
}
```

```
greet("John");  // Output: Hallo, John!
```

Er zijn ook **pijl-functies** (arrow functions), die een kortere syntaxis hebben:

```
javascript
Code kopiëren
const greet = (naam) => {
    console.log("Hallo, " + naam + "!");
};

greet("Jane");  // Output: Hallo, Jane!
```

7. Voorwaardelijke logica (if, else)

Met `if`, `else if`, en `else` kun je code op basis van een voorwaarde uitvoeren.

Voorbeeld:

```
javascript
Code kopiëren
let leeftijd = 18;

if (leeftijd >= 18) {
    console.log("Je bent volwassen.");
} else {
    console.log("Je bent minderjarig.");
}
```

8. Loops (lussen)

Met lussen kun je herhaaldelijk dezelfde code uitvoeren.

- **for-loop**:

```
javascript
Code kopiëren
for (let i = 0; i < 5; i++) {
    console.log(i);  // Output: 0, 1, 2, 3, 4
}
```

- **while-loop**:

```
javascript
Code kopiëren
let i = 0;
while (i < 5) {
    console.log(i);  // Output: 0, 1, 2, 3, 4
    i++;
}
```

9. Objecten

Objecten zijn verzamelingen van sleutel-waarde paren. Ze worden gebruikt om gegevens te groeperen.

Voorbeeld:

```
javascript
Code kopiëren
let persoon = {
    naam: "John",
    leeftijd: 30,
    adres: "Amsterdam"
};

console.log(persoon.naam);  // Output: John
console.log(persoon["leeftijd"]);  // Output: 30
```

10. Arrays

Arrays zijn lijsten van gegevens. Je kunt items ophalen via hun index (0-gebaseerd).

Voorbeeld:

```
javascript
Code kopiëren
let fruits = ["appel", "banaan", "kers"];

console.log(fruits[0]);  // Output: appel
console.log(fruits.length);  // Output: 3

fruits.push("druif");  // Voegt een item toe aan het einde
console.log(fruits);  // Output: ["appel", "banaan", "kers", "druif"]
```

11. Asynchrone JavaScript (setTimeout, Promises, async/await)

Asynchrone code is essentieel voor het werken met bijvoorbeeld API-aanroepen of timers.

- **setTimeout**: Voert een functie uit na een vertraging.

```
javascript
Code kopiëren
setTimeout(() => {
    console.log("Dit gebeurt na 2 seconden");
}, 2000);
```

- **Promises**: Gebruikt voor asynchrone operaties, zoals API-aanroepen.

```
javascript
Code kopiëren
let promise = new Promise((resolve, reject) => {
    let success = true;
    if (success) {
        resolve("Sucess!");
    } else {
        reject("Fout!");
    }
});

promise
    .then(result => console.log(result))   // Output: Success!
    .catch(error => console.log(error));
```

- **async/await**: Biedt een makkelijker manier om met asynchrone code te werken.

```
javascript
Code kopiëren
async function fetchData() {
    let response = await fetch("https://api.example.com/data");
    let data = await response.json();
    console.log(data);
}

fetchData();
```

---

12. Dom Manipulatie (Document Object Model)

Met JavaScript kun je de HTML en CSS van een pagina dynamisch aanpassen.

Voorbeeld:

```
html
Code kopiëren
<!DOCTYPE html>
<html>
<head>
    <title>DOM Manipulatie</title>
</head>
<body>
    <h1 id="mijnTitel">Hallo!</h1>
    <script>
        document.getElementById("mijnTitel").innerText = "Hallo, JavaScript!";
    </script>
</body>
</html>
```

Samenvatting

Deze spoedcursus heeft je door de basisconcepten van JavaScript geleid, zoals variabelen, functies, objecten, arrays, lussen, en DOM-manipulatie. Met deze kennis kun je beginnen met het schrijven van dynamische webpagina’s. Er is veel meer te leren, zoals geavanceerde onderwerpen zoals modules, closures, en het werken met API's, maar deze basis is een goede start.

Wil je verder gaan met leren of specifieke onderwerpen dieper verkennen? Laat het me weten!

Wat is het DOM?

DOM staat voor **Document Object Model**. Het is een programmeerinterface voor HTML- en XML-documenten. Het DOM beschrijft de structuur van een webpagina als een boom van objecten die representeren:

- HTML-elementen (zoals `<div>`, `<p>`, `<button>`, enz.)
- Attributen van deze elementen (zoals `id`, `class`, `src`, enz.)
- De tekst tussen de HTML-elementen (zoals de inhoud binnen een `<p>`-tag)

Wanneer je een webpagina laadt, maakt de browser een representatie van de pagina aan in het geheugen (de DOM). JavaScript kan dan de DOM manipuleren, oftewel de inhoud, structuur en stijl van de webpagina wijzigen.

Waarom is DOM-manipulatie belangrijk?

DOM-manipulatie stelt je in staat om:

1. **Inhoud van een pagina aan te passen** (zoals tekst in een paragraaf of afbeelding).
2. **Elementen toe te voegen of te verwijderen** van de pagina (zoals knoppen, lijsten, enz.).
3. **Stijlen te wijzigen** (zoals het veranderen van de kleur van een tekst of het aanpassen van de layout).
4. **Gebeurtenissen te behandelen**, zoals klikken, toetsenbordinvoer en meer.

Hoe manipuleer je het DOM met JavaScript?

Je gebruikt JavaScript om het DOM te benaderen en te wijzigen via verschillende methoden. Hier zijn enkele veelvoorkomende taken en de bijbehorende JavaScript-methoden:

1. Elementen selecteren

Om elementen in de DOM te selecteren en ermee te werken, kun je verschillende methoden gebruiken:

- **`getElementById()`**: Selecteert een element op basis van zijn `id`.
- **`getElementsByClassName()`**: Selecteert elementen op basis van hun `class`.
- **`getElementsByTagName()`**: Selecteert elementen op basis van hun HTML-tagnaam.
- **`querySelector()`**: Selecteert het eerste element dat overeenkomt met een CSS-selector.
- **`querySelectorAll()`**: Selecteert alle elementen die overeenkomen met een CSS-selector.

Voorbeelden:

```
// Selecteren op id
let titel = document.getElementById('mijnTitel');

// Selecteren op class
let paragrafen = document.getElementsByClassName('mijnParagraaf');

// Selecteren met een CSS-selector
let knop = document.querySelector('button');
```

2. Inhoud wijzigen

Na het selecteren van een element, kun je de inhoud ervan aanpassen.

- **`innerText`**: Wijzigt de zichtbare tekst van een element.
- **`innerHTML`**: Wijzigt de HTML-inhoud (inclusief eventuele tags) van een element.

Voorbeeld:

```
let titel = document.getElementById('mijnTitel');
titel.innerText = "Nieuwe Titel!"; // Wijzigt de tekst

let div = document.getElementById('mijnDiv');
div.innerHTML = "<p>Nieuwe <strong>HTML</strong> inhoud</p>"; // Wijzigt de HTML-
inhoud
```

3. Elementen toevoegen of verwijderen

Met JavaScript kun je elementen toevoegen of verwijderen uit de DOM.

- **`createElement()`**: Maak een nieuw element aan.
- **`appendChild()`**: Voeg een nieuw element toe aan een ander element.
- **`removeChild()`**: Verwijder een element uit de DOM.
- **`remove()`**: Verwijder een element zelf (deze is handiger dan `removeChild()`).

Voorbeeld:

```
// Een nieuw element aanmaken
let nieuweParagraaf = document.createElement('p');
nieuweParagraaf.innerText = "Dit is een nieuwe paragraaf.";

// Het nieuwe element toevoegen aan de body
document.body.appendChild(nieuweParagraaf);

// Verwijderen van een element
let oudElement = document.getElementById('oudElement');
oudElement.remove();  // Verwijdert het element zelf
```

4. Stijlen wijzigen

Je kunt de stijl van elementen wijzigen door toegang te krijgen tot hun `style`-eigenschappen.

Voorbeeld:

```
let titel = document.getElementById('mijnTitel');
titel.style.color = "blue";  // Verandert de tekstkleur naar blauw
titel.style.fontSize = "30px";  // Verandert de fontgrootte naar 30px
```

5. Gebeurtenissen (Events) behandelen

Je kunt evenementen zoals klikken, muisbewegingen, of het indrukken van toetsen behandelen door eventlisteners toe te voegen aan elementen.

Voorbeeld:

```
// Voeg een click event listener toe aan een knop
let knop = document.querySelector('button');
knop.addEventListener('click', function() {
    alert('De knop is geklikt!');
});
```

Je kunt ook andere events gebruiken zoals `mouseover`, `keydown`, `submit`, enzovoort.

Voorbeeld van DOM-manipulatie

Hier is een voorbeeld van een eenvoudige webpagina die JavaScript gebruikt om de DOM te manipuleren:

```
<!DOCTYPE html>
<html lang="nl">
<head>
    <meta charset="UTF-8">
    <title>DOM Manipulatie Voorbeeld</title>
</head>
<body>

    <h1 id="mijnTitel">Oude Titel</h1>
    <p class="mijnParagraaf">Dit is een paragraaf.</p>
    <button id="mijnKnop">Klik mij!</button>

    <script>
        // Wijzig de tekst van de titel
        let titel = document.getElementById('mijnTitel');
        titel.innerText = "Nieuwe Titel!";

        // Voeg een nieuwe paragraaf toe
        let nieuweParagraaf = document.createElement('p');
        nieuweParagraaf.innerText = "Dit is een nieuwe paragraaf die ik heb
toegevoegd!";
        document.body.appendChild(nieuweParagraaf);

        // Verander de stijl van de eerste paragraaf
        let eersteParagraaf = document.querySelector('.mijnParagraaf');
        eersteParagraaf.style.color = "red";

        // Voeg een click event toe aan de knop
        let knop = document.getElementById('mijnKnop');
        knop.addEventListener('click', function() {
            alert('Knop geklikt!');
        });
    </script>

</body>
</html>
```

Samenvatting

- **DOM-manipulatie** is het proces van het dynamisch aanpassen van de inhoud, structuur en stijl van een webpagina met JavaScript.
- Je kunt elementen **selecteren**, hun **inhoud** wijzigen, nieuwe **elementen toevoegen** of bestaande **verwijderen**, en **stijlen** aanpassen.

- Je kunt **gebeurtenissen** afhandelen, zoals het klikken op knoppen of het invoeren van tekst.

Door DOM-manipulatie kun je interactieve en dynamische webpagina's maken die reageren op de acties van de gebruiker. Het is een van de kernvaardigheden die je nodig hebt voor webontwikkeling!

Node.js is een runtime-omgeving voor JavaScript die het mogelijk maakt om JavaScript-code buiten een webbrowser uit te voeren, bijvoorbeeld op een server. Het is gebaseerd op de V8 JavaScript-engine van Google Chrome en is vooral handig voor het bouwen van schaalbare netwerkapplicaties zoals webservers.

npm (Node Package Manager) is een pakketbeheerder die samenwerkt met Node.js. Het stelt ontwikkelaars in staat om externe bibliotheken (pakketten) te installeren, beheren en delen. Deze bibliotheken kunnen gebruikt worden om de ontwikkeling van applicaties te versnellen door herbruikbare code toe te voegen.

Samengevat:

- **Node.js** draait JavaScript op de server.
- **npm** helpt om JavaScript-pakketten en afhankelijkheden te beheren.

Ja, er zijn een aantal belangrijke dingen die je moet weten over de werking van **Node.js** en **npm** om effectief met deze technologieën te werken. Hier zijn enkele kernpunten:

1. Asynchrone werking in Node.js

- **Node.js** is ontworpen voor asynchrone, niet-blokkerende I/O (input/output). Dit betekent dat Node.js verschillende taken tegelijkertijd kan uitvoeren zonder te wachten totdat één taak is voltooid, wat het uitermate geschikt maakt voor toepassingen die veel netwerkverkeer of bestandstoegang vereisen, zoals webservers.
- **Callbacks, Promises en Async/Await** worden veel gebruikt in Node.js om asynchrone operaties af te handelen.

2. Event Loop

- Node.js werkt met een **event loop**: het verwerkt taken zoals HTTP-verzoeken door een enkele thread te gebruiken. Wanneer een taak een asynchrone operatie uitvoert (bijv. lezen van een bestand of ophalen van data via een API), blokkeert Node.js niet en kan het door andere taken gaan totdat de asynchrone taak voltooid is.
- Het begrijpen van hoe de **event loop** werkt is belangrijk om efficiënt met Node.js te kunnen ontwikkelen.

3. Single-threaded

- **Node.js** is single-threaded, wat betekent dat het slechts één thread gebruikt voor het uitvoeren van de JavaScript-code. Dit kan zowel een voordeel als een uitdaging zijn. Het is snel bij I/O-intensieve toepassingen, maar voor CPU-intensieve taken kan het prestaties beïnvloeden, omdat die de enige thread kunnen blokkeren.
- Voor CPU-intensieve operaties kun je bijvoorbeeld **Worker Threads** of **Child Processes** gebruiken om de werkbelasting te verdelen.

4. npm en versiebeheer

- **npm** beheert niet alleen pakketten, maar ook versiebeheer van afhankelijkheden. Dit zorgt ervoor dat je specifieke versies van bibliotheken kunt gebruiken die compatibel zijn met je project.
- Het bestand **package.json** is essentieel voor versiebeheer en voor het bijhouden van de afhankelijkheden van je project. Het bepaalt welke versies van pakketten geïnstalleerd moeten worden en biedt ook scripts die je kunt gebruiken om bepaalde taken automatisch uit te voeren.
- **Semver** (Semantisch versiebeheer) wordt gebruikt om versie-informatie van pakketten te structureren. Pakketten hebben versies zoals `1.2.3`, waarbij elk nummer een specifiek type wijziging aangeeft:
    - **Major** (1): Grote wijzigingen die niet compatibel zijn met oudere versies.
    - **Minor** (2): Kleine nieuwe functies die compatibel blijven met oudere versies.
    - **Patch** (3): Bugfixes en kleinere aanpassingen.

5. Global vs. Local Installatie

- Pakketten kunnen **lokaal** (alleen voor je project) of **globaal** (voor het hele systeem) worden geïnstalleerd. Lokale installaties worden in de `node_modules`-map geplaatst en zijn alleen toegankelijk binnen dat specifieke project.
- Bij globale installatie worden de pakketten vaak gebruikt voor tools of commando's die overal op je systeem beschikbaar moeten zijn, zoals `nodemon` of `typescript`.

6. Beheer van afhankelijkheden met npm

- **package.json** en **package-lock.json**:
    - Het **package.json** bestand houdt de informatie bij over de projectafhankelijkheden.
    - Het **package-lock.json** bestand legt exact vast welke versies van de afhankelijkheden zijn geïnstalleerd om ervoor te zorgen dat het project consistent blijft, zelfs als iemand anders het project kloneert en `npm install` uitvoert.
- **DevDependencies** vs. **Dependencies**:
    - **dependencies** zijn bibliotheken die je project nodig heeft om te draaien (zoals Express voor een server).
    - **devDependencies** zijn tools die alleen tijdens de ontwikkeling nodig zijn (zoals linters of testtools).

7. npm Scripts

- **npm scripts** zijn commando’s die je in de `package.json` kunt definiëren om veelvoorkomende taken te automatiseren, zoals het starten van de server of het uitvoeren van tests. Bijvoorbeeld:

```
"scripts": {
  "start": "node app.js",
  "test": "mocha"
}
```

Vervolgens kun je deze scripts uitvoeren met `npm run start` of `npm run test`.

8. Veiligheid en Best Practices

- **npm audit**: npm biedt de mogelijkheid om je afhankelijkheden te scannen op bekende beveiligingsproblemen met het commando `npm audit`.
- Het is belangrijk om verouderde of onveilige pakketten regelmatig bij te werken en afhankelijkheden goed te beheren.
- Zorg ervoor dat je geen gevoelige gegevens (zoals wachtwoorden of API-sleutels) opslaat in je `package.json` of versiebeheer.

9. Gebruik van externe pakketten

- Node.js heeft een enorme ecosystem van **npm-pakketten** die je kunt gebruiken, van webframeworks (zoals Express) tot databases, tools voor testen, en zelfs command-line utilities.
- Zorg ervoor dat je goed begrijpt wat een extern pakket doet, en kies alleen betrouwbare en goed onderhouden bibliotheken om je project robuust te houden.

Samenvatting:

- **Node.js** is een event-driven, non-blocking runtime-omgeving die JavaScript buiten de browser uitvoert, ideaal voor schaalbare netwerkapplicaties.
- **npm** is de pakketbeheerder voor Node.js en helpt bij het installeren, beheren en delen van afhankelijkheden.
- Het is belangrijk om de concepten van asynchrone programmering, de event loop en versiebeheer te begrijpen, evenals het efficiënt gebruiken van npm voor afhankelijkheden en scripts.

Door deze basisprincipes te begrijpen, kun je meer effectieve en onderhoudbare Node.js-applicaties ontwikkelen.