JavaScript基礎

基本から応用まで

1. 変数宣言と定数

```
// 再代入可能な変数
let count = 0;
count = 1;

// 定数（再代入不可）
const PI = 3.14;
const MAX_VALUE = 100;

// var は非推奨
var oldWay = "legacy"; // 使用しない
```

2. データ型

```
// 数値
const num = 42;
const float = 3.14;

// 文字列
const str1 = "Hello";
const str2 = 'World';
const template = `Value: ${num}`; // テンプレート文字列

// 真偽値
const isTrue = true;
const isFalse = false;

// 配列
const array = [1, 2, 3];

// オブジェクト
const obj = {
    name: "John",
    age: 30
};
```

3. 条件分岐

```
// if文
if (condition) {
    // 処理
} else if (otherCondition) {
    // 別の処理
} else {
    // それ以外の処理
}

// switch文
switch (value) {
    case 1:
        // 処理1
        break;
    case 2:
        // 処理2
        break;
    default:
        // デフォルト処理
}
```

4. ループ処理

```
// for文
for (let i = 0; i < 5; i++) {
    console.log(i);
}

// while文
while (condition) {
    // 処理
}

// 配列のループ
const items = ['a', 'b', 'c'];
items.forEach(item => {
    console.log(item);
});

// for...of（イテレータブルなオブジェクト用）
for (const item of items) {
    console.log(item);
}
```

5. 関数定義

```
// 通常の関数
function add(a, b) {
    return a + b;
}

// アロー関数
const multiply = (a, b) => a * b;

// 複数行のアロー関数
const greet = (name) => {
    const message = `Hello, ${name}!`;
    return message;
};

// デフォルトパラメータ
const power = (base, exponent = 2) => {
    return Math.pow(base, exponent);
};
```

6. オブジェクト操作

```
// オブジェクトの作成
const person = {
    name: "Alice",
    age: 25,
    greet() {
        return `Hello, I'm ${this.name}`;
    }
};

// プロパティアクセス
console.log(person.name);
console.log(person["age"]);

// スプレッド演算子
const updatedPerson = {
    ...person,
    age: 26
};
```

7. 配列操作

```
const numbers = [1, 2, 3, 4, 5];

// map: 各要素を変換
const doubled = numbers.map(x => x * 2);

// filter: 条件に合う要素を抽出
const evens = numbers.filter(x => x % 2 === 0);

// reduce: 要素を集約
const sum = numbers.reduce((acc, cur) => acc + cur, 0);

// 配列の追加・削除
numbers.push(6);    // 末尾に追加
numbers.pop();      // 末尾から削除
numbers.unshift(0); // 先頭に追加
numbers.shift();    // 先頭から削除
```

8. 非同期処理

```
// Promise
const fetchData = () => {
    return new Promise((resolve, reject) => {
        // 非同期処理
        if (success) {
            resolve(data);
        } else {
            reject(error);
        }
    });
};

// async/await
async function getData() {
    try {
        const response = await fetch('https://api.example.com/data');
        const data = await response.json();
        return data;
    } catch (error) {
        console.error('Error:', error);
    }
}
```

9. モジュール

```
// エクスポート
export const utils = {
    formatDate(date) {
        return date.toISOString();
    }
};

export default class User {
    constructor(name) {
        this.name = name;
    }
}

// インポート
import { utils } from './utils.js';
import User from './User.js';
```

10. エラーハンドリング

```
try {
    // エラーが発生する可能性のある処理
    throw new Error('エラーが発生しました');
} catch (error) {
    // エラー処理
    console.error('エラー:', error.message);
} finally {
    // 必ず実行される処理
    console.log('処理完了');
}
```