第1章: JavaScriptとは - 言語特性と最初の一歩

前回は、JavaScriptの冒険に出るための準備として、開発環境を整えたり、GitやGitHubといった便利なツールの使い方を学んだりしましたね。いよいよ今回からは、冒険の主役であるJavaScriptそのものについて深く探求していきます！

JavaScriptは、ウェブページに命を吹き込み、ユーザーの操作に応じて見た目が変わったり、アニメーションしたりと、インタラクティブな体験を作り出すための強力なプログラミング言語です。まるで魔法の杖のように、ウェブの世界をより豊かで楽しいものに変えることができるんですよ。

この章では、まず「JavaScriptって一体何者なの？」という基本的なところから始め、HTMLやCSSといったウェブ技術との関係性、そしてJavaScriptが持つユニークな特徴、さらにはプログラムを書く上での最初のお作法について一緒に学んでいきましょう。

最初は覚えることが多く感じるかもしれませんが、一つ一つ丁寧に解説していきますので、安心してくださいね。さあ、JavaScriptの魔法の世界への第一歩を踏み出しましょう！

目次

---

1. JavaScriptの概要

まずは、JavaScriptがウェブの世界でどんな役割を担っているのか、そしてどうやってHTMLファイルと連携するのかを見ていきましょう。

HTML/CSSとの関係：ウェブページの役割分担

ウェブページを作るとき、主に3つの技術が使われます。それぞれ役割が違うんですよ。

- **HTML (HyperText Markup Language)**: ウェブページの**骨組みや構造**を担当します。例えば、「ここが見出しで、ここが段落、ここに画像があって…」といった情報を定義します。人間でいうと骨格みたいなものですね。
- **CSS (Cascading Style Sheets)**: ウェブページの**見た目や装飾**を担当します。文字の色や大きさ、背景色、レイアウトなどを指定して、ページを美しく整えます。人間でいうと服装や髪型、メイクといったところでしょうか。
- **JavaScript (JS)**: ウェブページに**動きや対話性**を加えるのがJavaScriptの役割です。ボタンをクリックしたらメッセージが表示されたり、入力内容に応じて表示が変わったり、アニメーションしたり…。人間でいうと、話したり動いたりする能力に近いです。

この3つが協力し合うことで、私たちが普段見ているような、情報が整理されていて、見た目も綺麗で、操作もできるウェブページが作られているんです。

JavaScriptファイルの連携方法

JavaScriptのコードをHTMLファイルに組み込む方法はいくつかありますが、主に2つの方法がよく使われます。

1. **外部ファイルとして読み込む**: JavaScriptのコードを別のファイル（拡張子は `.js`、例えば `app.js`）に書いておき、HTMLファイルからそれを読み込む方法です。これが一番一般的で、コードの管理がしやすくなるのでおすすめです。

```
<!-- filepath: (任意のフォルダ)/index.html -->
<!DOCTYPE html>
<html>
<head>
    <title>JS連携テスト</title>
</head>
<body>
    <h1>JavaScript連携テスト</h1>
    <!-- scriptタグのsrc属性でJSファイルを指定 -->
    <script src="app.js"></script>
</body>
</html>
```

```
// filepath: (index.htmlと同じフォルダ)/app.js
// このファイルにJavaScriptのコードを書いていきます
console.log("app.jsが読み込まれました！");
```

   ポイント: `script` タグは、HTMLファイルの `</body>` タグの閉じタグの直前に書くのが一般的です。これは、HTMLの構造が先に読み込まれてからJavaScriptが実行されるようにするためで、JavaScriptがHTML要素を操作しようとしたときに、まだその要素が存在しない、といったエラーを防ぐためです。

2. **HTMLファイル内に直接書き込む（インライン）**: 短いコードの場合や、ちょっとしたテストをしたい場合には、HTMLファイル内の `<script>` タグの間に直接JavaScriptコードを書くこともできます。

```
<!-- filepath: (任意のフォルダ)/inline_script.html -->
<!DOCTYPE html>
<html>
<head>
    <title>インラインJSテスト</title>
</head>
<body>
    <h1>インラインJavaScriptテスト</h1>
    <script>
        // ここに直接JavaScriptコードを書きます
        alert("こんにちは！インラインJavaScriptです。");
    </script>
</body>
</html>
```

ただ、コードが長くなってくるとHTMLファイルが見づらくなってしまうので、基本的には外部ファイルに分ける方法を使いましょうね。

実行環境：どこで動くの？

JavaScriptが実行される主な場所は2つあります。

- **ブラウザ（フロントエンド）**: これがJavaScriptの最も伝統的で主要な活躍場所です。Google Chrome, Microsoft Edge, Firefoxといったウェブブラウザには、JavaScriptを実行するための「エンジン」が内蔵されています。ユーザーが見ているウェブページ上で、HTMLやCSSと連携しながら動作し、インタラクティブな体験を提供します。私たちがこれから主に学んでいくのは、このブラウザ上で動くJavaScriptです。
- **Node.js（サーバーサイド、ツールなど）**: 「序章」でも少し触れましたが、Node.jsという環境を使うと、JavaScriptをブラウザの外、つまり自分のパソコンやサーバー上でも動かすことができます。これにより、ウェブサイトの裏側の処理（データベースとの連携など）をJavaScriptで書いたり、開発を助ける便利なツール（例えば、コードをチェックしたり、ファイルをまとめたりするもの）を実行したりできるようになります。Node.jsについては、また後の章で詳しく触れる機会がありますので、今は「ブラウザ以外でも動かせるんだな」くらいに覚えておけば大丈夫です。

---

2. JavaScriptの主な言語特性

JavaScriptには、他のプログラミング言語と比べていくつかのユニークな特徴があります。これらを理解しておくと、学習を進める上で役立ちますよ。

ただし、プログラミングを初めて学ぶ皆には難しいかもしれないので、「へー」って感じで聞き流してほしいです

動的型付け：柔軟だけど注意も必要

JavaScriptは**動的型付け言語**（Dynamically Typed Language）です。これは、変数にどんな種類のデータ（数値、文字列、真偽値など）でも代入でき、実行時に型が決まるという特徴です。

例えば、こんなことができます。

```
let myData = 100; // 最初は数値
console.log(myData); // 100 と表示

myData = ”こんにちは！”; // 次に文字列を代入
console.log(myData); // こんにちは！ と表示

myData = true; // さらに真偽値を代入
console.log(myData); // true と表示
```

これは非常に柔軟で書きやすい反面、意図しない型で処理を進めてしまってエラーになったり、バグの原因になったりすることもあります。 「あれ、この変数には数値が入っているつもりだったのに、いつの間にか文字列に変わってた！」なんてことも。

メリット:

- コードを素早く書ける。
- 型の宣言が不要なので、記述量が減る。

デメリット:

- 実行するまで型に関するエラーが分かりにくい。
- 大規模な開発では、型の管理が難しくなることがある。

このデメリットを補うために、JavaScriptに静的な型付けの機能を追加した**TypeScript**という言語も人気があります。TypeScriptについては、この学習プランの後半で触れる予定ですので、お楽しみに！

プロトタイプベースのオブジェクト指向：ちょっと変わった仕組み

多くのプログラミング言語（JavaやC#など）では、「クラス」という設計図をもとにオブジェクト（モノ）を作る「クラスベースのオブジェクト指向」が採用されています。

一方、JavaScriptは**プロトタイプベースのオブジェクト指向**という、少し変わった仕組みを持っています。これは、既存のオブジェクトを「プロトタイプ」（原型）として、それをコピーしたり拡張したりして新しいオブジェクトを作っていく考え方です。

歴史的経緯: 実は、JavaScriptが生まれた背景には面白い話があります。1990年代半ば、ウェブページに手軽に動きを加えられる軽量なスクリプト言語が求められていました。当時主流だったJavaアプレットは高機能でしたが、少し重厚長大でした。そこで、ネットスケープ社のブレンダン・アイク氏が、なんとわずか10日間でJavaScriptの最初のバージョンを設計したと言われています。その際、Selfというプログラミング言語などの影響を受け、クラスではなくプロトタイプをベースとした、より柔軟なオブジェクトモデルが採用されたのです。

このプロトタイプという考え方は、最初は少しとっつきにくいかもしれませんが、JavaScriptの柔軟性を支える重要な要素の一つです。オブジェクトやプロトタイプについては、後の章でじっくり解説しますので、今は「JavaScriptのオブジェクトの作り方は、他の言語とちょっと違うんだな」くらいに思っておいてください。

関数型プログラミングの側面：関数は働き者

JavaScriptでは、関数が非常に重要な役割を果たします。そして、関数は「**第一級オブジェクト**」として扱われます。これは、関数を他のデータ（数値や文字列など）と同じように、変数に代入したり、他の関数の引数として渡したり、関数の戻り値として返したりできる、という意味です。

この性質を利用すると、関数型プログラミングと呼ばれるスタイルでコードを書くことができます。例えば、ある処理を行う関数を別の関数に渡して、その処理をカスタマイズする、なんてこともできます。

ちょっと難しいかもしれませんが、簡単な「高階関数もどき」の例を見てみましょう。（高階関数とは、関数を引数に取ったり、関数を返したりする関数のことです）

```
function operateOnArray(array, operation) {
  const resultArray = []; // 結果を格納する新しい配列
  for (let i = 0; i < array.length; i++) {
    // 配列の各要素に対して、渡されたoperation関数を実行し、結果を新しい配列に追加
    resultArray.push(operation(array[i]));
  }
  return resultArray;
}

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

// 各要素を2倍にする操作
const doubledNumbers = operateOnArray(numbers, function(num) {
  return num * 2;
});
console.log(doubledNumbers); // [2, 4, 6, 8, 10] と表示されます

// 各要素に10を加える操作
const addedTenNumbers = operateOnArray(numbers, function(num) {
  return num + 10;
});
console.log(addedTenNumbers); // [11, 12, 13, 14, 15] と表示されます
```

この例では、`operateOnArray` という関数が、配列（`array`）ともう一つの関数（`operation`）を引数に取っています。そして、`operation` に渡す関数を変えるだけで、配列の各要素に対する処理内容を自由に変えられていますよね。これが関数型プログラミングの面白いところの一つです。

関数については、後の章でさらに詳しく学びますので、ここでは「JavaScriptの関数って、ただの処理の塊じゃなくて、もっと色々なことができるパワフルなやつなんだな」と感じてもらえれば十分です。

イベント駆動型プログラミング：何かが起きたら動く

JavaScriptは、特にブラウザ上で動く場合、**イベント駆動型プログラミング**というスタイルが中心になります。「イベント」とは、ユーザーの操作（ボタンのクリック、マウスの移動、キーボード入力など）や、システムの出来事（ページの読み込み完了、タイマーの時間が来たなど）のことです。

イベント駆動型プログラミングでは、「もし〇〇というイベントが起きたら、××という処理を実行する」というように、あらかじめイベントとそれに対応する処理を結びつけておきます。そして、実際にそのイベントが発生したときに、対応する処理が呼び出される、という流れになります。

例えば、「ユーザーがボタンをクリックしたら、メッセージを表示する」といった処理は、まさにイベント駆動の考え方ですね。このイベント処理についても、後の章で詳しく見ていきます。

---

3. JavaScriptプログラミングの基本作法

さあ、いよいよJavaScriptのコードを実際に書いていく上での基本的なルール、お作法について学んでいきましょう。

Hello, World!：最初のご挨拶

プログラミング学習の伝統として、最初に「Hello, World!」という文字列を表示するプログラムを作ることが多いです。JavaScriptでもやってみましょう！

1. ブラウザのコンソールに表示する

ブラウザの開発者ツールにある「コンソール」は、JavaScriptのメッセージを表示したり、簡単なテストコードを実行したりするのに非常に便利な場所です。`console.log()` という命令を使うと、コンソールに好きなメッセージを表示できます。

```
console.log(”Hello, World! from console!”);
```

この `hello_console.js` ファイルをHTMLファイルから読み込んでブラウザで表示し、開発者ツールのコンソールを開いてみてください。（序章で言及した、ブラウザでF12を押すと開くやつです）「Hello, World! from console!」と表示されていれば成功です！ `console.log()` は、プログラムが正しく動いているか確認したり、変数の値を見たりするのに、これから本当によく使うことになるので、ぜひ覚えておいてくださいね。名探偵の虫眼鏡のようなものです！

2. ブラウザの画面にアラートとして表示する

`alert()` という命令を使うと、ブラウザの画面に小さなポップアップウィンドウ（アラートダイアログ）を表示して、メッセージを伝えることができます。

```
alert(”Hello, World! from alert!”);
```

この `hello_alert.js` をHTMLから読み込むと、ページを開いたときに「Hello, World! from alert!」というメッセージのアラートが表示されるはずです。

3. HTML要素の内容を変更して表示する（DOM操作）

JavaScriptを使って、HTML要素の内容を直接書き換えることもできます。これは「DOM操作」と呼ばれるテクニックで、ウェブページを動的に変化させるための基本です。

まずは、こんなHTMLファイルを用意しましょう。

```
<!DOCTYPE html>
<html>
<head>
    <title>JS DOM Test</title>
```

```
</head>
<body>
    <p id="messageArea">ここにメッセージが表示されます。</p>

    <button onclick="displayMessage()">メッセージ表示</button>

    <script>
        function displayMessage() {
            // idが "messageArea" のHTML要素を取得
            const messageElement = document.getElementById('messageArea');
            // その要素のテキスト内容を書き換える
            messageElement.textContent = 'Hello, World! from JavaScript DOM manipulation!';
        }
    </script>
</body>
</html>
```

このHTMLファイルをブラウザで開いて、「メッセージ表示」ボタンをクリックしてみてください。`<p>`タグで囲まれた部分のテキストが「Hello, World! from JavaScript DOM manipulation!」に変われば大成功です！

ここでは、`document.getElementById('messageArea')`で特定のHTML要素を見つけ出し、`textContent`というプロパティを使ってその中身を書き換えています。「え、`document`って何？`getElementById`って？`textContent`って？？」と、たくさんの「？」が浮かんでいるかもしれませんが、心配しないでください！これらのDOM操作に関する詳しいことは、後の「DOMの探求」の章でじっくり解説します。今は、「JavaScriptって、HTMLの中身も変えられるんだな、すごい！」と感じてもらえればOKです。

コメント：未来の自分へのメッセージ

プログラムの中に、処理内容を説明するためのメモ書きを残すことができます。これを**コメント**と言います。コメントは、プログラムの実行には影響しませんが、コードを後から見返したときや、他の人がコードを読むときに、処理内容を理解しやすくするために非常に重要です。

JavaScriptのコメントには、主に2種類あります。

- **一行コメント**: から行末までがコメントになります。

```
// filepath: (どこかのフォルダ)/comments_example.js
// これは一行コメントです
let userAge = 20; // ユーザーの年齢を格納する変数（これもコメント）
```

- **複数行コメント**: `/*` と `*/` で囲まれた範囲がコメントになります。

```
/*
  これは複数行コメントです。
  何行にもわたって書くことができます。
  主に、関数の説明や、少し複雑な処理の概要を書くのに使います。
*/
function calculatePrice(quantity, unitPrice) {
  // 商品の数量と単価から合計金額を計算する
```

```
  return quantity * unitPrice;
}
```

個人的には、コメントは未来の自分へのメッセージであり、チームメイトへの思いやりだと思っています。「このコード、何でこんな書き方したんだっけ…？」と数ヶ月後に悩むことは本当によくあります（笑）。面倒くさがらずに、分かりやすいコメントを書く習慣をつけると、後で絶対に自分を褒めてあげたくなりますよ。これは経験上、間違いありません！

セミコロン：文の終わり、どうする？

JavaScriptでは、文（命令の単位）の終わりに**セミコロン（`;`）**を付けるのが一般的です。

```
let name = ”高専太郎”;
console.log(name);
alert(”処理が完了しました。”);
```

実は、JavaScriptには **ASI (Automatic Semicolon Insertion)** という、セミコロンが省略されていても、解釈できる場所では自動的にセミコロンを挿入してくれる機能があります。そのため、セミコロンを省略しても動く場合があります。

しかし、このASIは時々、意図しない場所でセミコロンを挿入してしまい、バグの原因になることがあります。例えば、こんなコードを見てください。

```
function getObject() {
  return // ASIにより、ここでセミコロンが勝手に挿入されてしまう！
  {
    name: ”ASI Trap”
  };
}

let myObject = getObject();
console.log(myObject); // undefined と表示されてしまう！
```

この例では、`return` の直後で改行しているため、ASIがそこで文の終わりだと判断してセミコロンを挿入してしまいます。その結果、`getObject` 関数は何も返さない（`undefined` を返す）ことになってしまうのです。

このような予期せぬ挙動を避けるため、そしてコードの意図を明確にするために、**文の終わりにはセミコロンを付けることを強く推奨します。** 多くの開発チームでは、セミコロンを付けることをコーディングスタイルとして定めています。私たちも、セミコロンは基本的に付ける、というルールで進めていきましょうね。

波括弧 `{}` ：まとまりを示す記号

波括弧 `{}`（curly braces）は、JavaScriptでいくつかの重要な役割を果たします。

- **ブロックを示す**: 複数の文を一つのまとまりとして扱う「ブロック」を作るのに使います。例えば、`if` 文や `for` ループ、関数の定義などで使われます。

```
// filepath: (どこかのフォルダ)/braces_block.js
let age = 18;
if (age >= 20) { // if文のブロックの始まり
  console.log("成人です。");
  console.log("お酒が飲めますね！");
} // if文のブロックの終わり
else { // else文のブロックの始まり
  console.log("未成年です。");
} // else文のブロックの終わり

function greet(name) { // 関数のブロックの始まり
  console.log("こんにちは、" + name + "さん！");
} // 関数のブロックの終わり
```

- **オブジェクトリテラルで使う**: 後で詳しく学びますが、「オブジェクト」というデータの塊を作る際にも波括弧を使います。

```
// filepath: (どこかのフォルダ)/braces_object.js
let user = { // オブジェクトリテラルの始まり
  name: "高専花子",
  age: 17,
  hobby: "プログラミング"
}; // オブジェクトリテラルの終わり
console.log(user.name); // 高専花子 と表示
```

波括弧は、コードの構造を明確にするために非常に重要です。どこからどこまでが一つのまとまりなのかを意識しながらコードを書いていきましょう。

---

4. 「オブジェクトという便利な箱」の具体性

JavaScriptでは、「オブジェクト」という概念が非常に重要です。先ほど少し出てきましたが、ここで改めて、オブジェクトがどんなものかイメージを掴んでおきましょう。

オブジェクトとは、簡単に言うと、**関連するデータ（情報）や機能（操作）をひとまとめにした「便利な箱」**のようなものです。 この箱の中には、「名前（**プロパティ**と呼びます）」と、それに対応する「値（データや機能）」が入っています。

例えば、「ユーザー」というオブジェクトを考えてみましょう。

```
let user = {
  // "name" という名前（プロパティ）に "山田太郎" という値（文字列データ）
  name: "山田太郎",
  // "age" という名前（プロパティ）に 30 という値（数値データ）
  age: 30,
  // "isStudent" という名前（プロパティ）に false という値（真偽値データ）
  isStudent: false,
  // "greet" という名前（プロパティ）に、挨拶する機能（関数）
```

```
  greet: function() {
    console.log("こんにちは、" + this.name + "です！");
  }
};

// オブジェクトのデータにアクセス
console.log(user.name);     // 山田太郎 と表示
console.log(user.age);      // 30 と表示

// オブジェクトの機能（メソッドと呼びます）を実行
user.greet(); // こんにちは、山田太郎です！ と表示
```

この`user`オブジェクトは、「名前」「年齢」「学生かどうか」といったデータと、「挨拶する」という機能をまとめて持っています。まるで、整理整頓された引き出しのように、関連するものを一箇所に集めて管理できるので、プログラムがとても分かりやすくなります。

この「オブジェクト」という考え方は、JavaScriptプログラミングのあらゆる場面で登場します。今はまだ「ふーん、そんなものがあるんだな」くらいで大丈夫です。これから少しずつ、この便利な箱の使い方に慣れていきましょうね！

✏️ 本日の演習

さあ、今日学んだことを早速試してみましょう！

1. **HTMLとJavaScriptの連携ファイル作成！**
    - デスクトップなど分かりやすい場所に、`js_practice`という名前で新しいフォルダを作成してください。
    - VSCodeで`js_practice`フォルダを開いてください。
    - `js_practice`フォルダの中に、`index.html`というファイルと`script.js`というファイルを作成してください。
    - index.html には、基本的なHTML構造（`<!DOCTYPE html>`, `<html>`, `<head>`, `<body>`タグなど）を書き、`<body>`タグの最後に`<script src="script.js"></script>`と記述して`script.js`を読み込むようにしてください。
    - `script.js`に、`console.log("JavaScriptの練習開始！");`と`alert("HTMLとJSの連携成功！");`という2行を書いて保存してください。
    - index.html をブラウザで開いて、アラートが表示され、開発者ツールのコンソールにメッセージが表示されることを確認しましょう。
2. **コメントを使ってみよう！**
    - 先ほど作成した`script.js`を開いてください。
    - 1行目の`console.log`の上に、一行コメントで「これはコンソールにメッセージを表示する命令です」と書いてみましょう。
    - 2行目の`alert`の上に、複数行コメントで「これはアラートダイアログを表示する命令です。¥nユーザーに簡単な通知をするのに使えます。」（`¥n`は改行を意味します）と書いてみましょう。
    - ファイルを保存して、もう一度 index.html をブラウザで開き、動作が変わらないこと（コメントは実行に影響しないこと）を確認しましょう。
3. **`console.log`で色々表示！**
    - `script.js`に、以下の内容を追記してみましょう。

    - 自分の名前を `console.log` で表示する。
    - 好きな食べ物を `console.log` で表示する。
    - `1 + 1` の計算結果を `console.log` で表示する。（ヒント: `console.log(1 + 1);` のように書けます）
- 保存してブラウザで確認し、コンソールに正しく表示されるか見てみましょう。

演習の解答例

演習1、2、3の指示に従って、`js_practice` フォルダ内に以下の2つのファイルを作成してください。

1. index.html
2. `script.js`

それぞれのファイルの内容は以下の通りです。

```
<!DOCTYPE html>
<html lang="ja">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>JavaScript 演習</title>
</head>
<body>
    <h1>JavaScriptの練習ページ</h1>
    <p>ブラウザの開発者ツールを開いてコンソールを確認してください。</p>

    <script src="script.js"></script>
</body>
</html>
```

```
// これはコンソールにメッセージを表示する命令です
console.log("JavaScriptの練習開始！");

/*
  これはアラートダイアログを表示する命令です。
  ユーザーに簡単な通知をするのに使えます。
*/
alert("HTMLとJSの連携成功！");

// 自分の名前を表示
console.log("高専太郎"); // ここはご自身の名前に置き換えてください

// 好きな食べ物を表示
console.log("カレーライス"); // ここはご自身の好きな食べ物に置き換えてください

// 1 + 1 の計算結果を表示
console.log(1 + 1);
```

実行方法：

1. `js_practice` フォルダを作成します。
2. 上記のコードをそれぞれ index.html と `script.js` という名前で `js_practice` フォルダ内に保存します。
3. index.html ファイルをウェブブラウザで開きます。
    - アラートが表示されることを確認してください。
    - ブラウザの開発者ツール（通常はF12キーで開きます）のコンソールタブを開き、メッセージや計算結果が表示されていることを確認してください。

まとめと次回予告

お疲れ様でした！今回は、JavaScriptがどんな言語で、ウェブページの中でどんな役割を果たしているのか、そしてプログラムを書き始めるための基本的なお作法について学びましたね。

- JavaScriptはHTML（骨組み）、CSS（見た目）と協力して、ウェブページに**動きや対話性**を加えること。
- JavaScriptコードは、外部ファイルとして読み込んだり、HTML内に直接書いたりできること。
- `console.log()` や `alert()` で簡単なメッセージを表示できること。
- コメントはコードを分かりやすくするための重要なメモであること。
- 文の終わりにはセミコロン（`;`）を付けるのが推奨されること。

実際に手を動かして、HTMLファイルとJavaScriptファイルを連携させたり、`console.log` を使ってみたりすることで、少しずつJavaScriptに慣れてきたのではないでしょうか。

次回は、プログラミングの基本中の基本である「**変数とデータ型**」について学びます。情報を一時的に保存しておくための「変数」とは何か、そしてJavaScriptが扱えるデータの種類にはどんなものがあるのか、といったことを詳しく見ていきます。ここも非常に大切な基礎となる部分ですので、お楽しみに！