## Step19 / DOM 操作（HTML を操作）

これまではコンピュータからの入出力は「ダイアログ」に頼っていました。
今回は、HTML に表示します。
JavaScript から HTML 要素を取得・変更・追加・複製・削除することを「**DOM 操作**」といいます。

**準備：**
Step5 でゲームのスタート画面を作りました。コードは以下の通りです。
step5.html を step19.html に複製してください。
今回は、DOM を説明するために「<h1>」→「<h1 id="title">」に追加編集しています。

**HTML：**

```
<!DOCTYPE html>
<html lang="ja">
<head>
  <meta charset="UTF-8">
  <title>Hello Game</title>
  <style>
    #main {
      text-align: center;
      padding: 30px;
      border: 1px solid #000;
    }
    #menu {
      padding: 30px;
    }
    #footer {
      text-align: center;
      margin: 50px;
    }
    button {
      border: 1px solid #000;
      padding: 10px 30px;
    }
  </style>
</head>
```

```
<body>
  <div id="main">
    <h1 id="title">Hello Game</h1>
    <p>スタートボタンを押してゲームを開始してください</p>
    <div id="menu">
      <button>スタート</button>
      <button>オプション</button>
    </div>
  </div>
  <div id="footer">
    (c) 2025 Hello Studio
  </div>
</body>
</html>
```

## DOM ツリー構造とは

ブラウザはこれを次のような**階層構造（ツリー構造）**として扱います

```
Document
└ html
   └ body
      ├ div#main
      │   ├ h1#title
      │   │   └ "Hello Game"
      │   ├ p
      │   │   └ "スタートボタンを押してゲームを開始してください"
      │   └ div#menu
      │       ├ button
      │       │   └ "スタート"
      │       └ button
      │           └ "オプション"
      └ div#footer
          └ "(c) 2025 Hello Studio"
```

## DOM を操作する

### 取得：

HTML からタグや id 属性の要素を取得してみましょう。

```
<script>
  //要素を取得 id="title" ★方法 1
  const title = document.getElementById("title");
  console.log(title);

  //要素を取得 id="title" ★方法 2
  const title2 = document.querySelector("#title");
  console.log(title2);

  //要素を取得 <h1>タグ
  const h1 = document.querySelector("h1");
  console.log(h1);

  //要素を取得 button タグすべて = 複数
  const buttons = document.querySelectorAll("button");
  console.log(buttons);
</script>
```

上記の通り、要素の取得方法は様々な方法があります。

| 方法 | 指定方法 | 取得対象 | 特徴 |
|---|---|---|---|
| getElementById("id 名") | id 属性 | 単一要素 | 最も基本的・高速 |
| getElementsByClassName("クラス名") | class 属性 | 複数要素 | 同じクラスをまとめて取得 |
| querySelector("CSS セレクタ") | CSS セレクタ | 最初の 1 つ | CSS 的で直感的 |
| querySelectorAll("CSS セレクタ") | CSS セレクタ | 複数要素 | forEach などでループ可能 |

getElementById() や getElementsByClassName() は基本的な取得方法です。
querySelector()／querySelectorAll() は CSS セレクタを使った柔軟な方法です。
柔軟な理由は、HTML タグ（”h1”）、id 属性（"#id”）、class 属性(.title)で指定することができます。
さらに細かな指定が可能ですので、本演習では CSS セレクタを使った要素の取得を基本にします。
※ 上記の buttons は、複数あるため「配列」というものに格納されます。これは別途解説します。

**変更：**

```
<script>
  //要素の内容を書き換え
  title.textContent = "こんばんは！";

  //要素のスタイルを変更
  title.style.color = "red";
</script>
```

**追加：**

```
<script>
  //要素の内容を追加
  const p2 = document.createElement("p");
  p2.id = "newline ";
  p2.textContent = "新しい段落です。";
  document.querySelector("#main").appendChild(p2);
</script>
```

**複製：**

```
<script>
  //要素を複製して追加
  const p3 = document.querySelector("p").cloneNode(true);
  document.querySelector("#main").appendChild(p3);
</script>
```

**削除：**

```
<script>
  //要素を削除する
  const footer = document.getElementById("footer");
  footer.remove();
</script>
```

DevTools で HTML 要素が変わっていることを確認しましょう。

**Step19 (2)：**

ここからは、文章や CSS 装飾（文字の色、大きさ）を DOM 操作で変更してみてください。
自分の好みに合わせて変更してみてください。

## Step20 / イベント処理

ユーザーの操作（クリック・マウス・入力など）に反応して処理を実行する仕組みを学びます。
このような動作を 「イベント処理」 と呼びます。

```
･･･<html><head>割愛･･･
<body>
    <button id="btn">クリックしてね</button>
    <p id="msg">まだクリックされていません</p>

    <script>
    // 要素を取得
    const btn = document.querySelector("#btn");
    const msg = document.querySelector("#msg");

    // クリックされたときにメッセージを表示する関数
    function showMsg(event) {
      msg.textContent = "クリックされました！";
      msg.style.color = "red";
    }
    // イベントリスナーを登録
    btn.addEventListener("click", showMsg);
  </script>
</body>
･･･</html></head>割愛･･･
```

解説：

addEventListener() は「ある操作（イベント）が起きたときに、実行する処理を登録する」ための命令です。 HTML を動かすうえで、ユーザー操作に反応する動的な Web ページを作る基礎となります。

要素.addEventListener("イベント名", 関数);

- 第 1 引数：イベントの種類（例："click"、"mouseover"、"input" など）
- 第 2 引数：実行したい処理（関数）

関数（function）については、後ほど解説します。
今回は click イベントを使いましたがイベントにはキーボード入力、マウス操作など様々な種類があります。「addEventListener イベント」などで調べてみましょう。

## Step21 / 関数

関数（function）とは、「ひとまとまりの処理に名前をつけて、あとで呼び出せるようにしたもの」です。
これまでに関数として、Math.random（ ）、Math.floor( )、alert( )、prompt（）等を活用してきましたが、これらは全て JavaScript に最初から備わっている（組み込み関数）です。
これらとは別に、ユーザが自分で関数を作ることが可能で、そのための書式が function です。

```
function 関数名( 引数 ) {
  実行内容;
  実行内容;
  実行内容;
}

// 関数の呼び出し例 1
関数名( 引数の値 );

// 関数の呼び出し例 2
document.addEventListener("click", 関数名);
```

呼び出し方：
1）関数名(); ・・・すぐに実行する場合は、()を付けて呼び出します
2）関数名　　・・・あとで実行する場合は、 ()を付けずに呼び出す
　　例）document.addEventListener("click", showMsg);　// クリックされたら実行
　　例）setTimeout(showMsg, 2000); // 2 秒後に実行

注意：
関数名は自由に決められます。ただし、変数名と同様に JavaScript の予約語は、ユーザが作る関数名として使うことはできないことに注意してください。どのような変数名や関数名があるかは「JavaScript 予約語」で検索して調べてください。

関数には「データを渡して、結果を変える」こともできます。以下に引数なし・ありの例を挙げます。

```
// コーヒーを作る関数
function makeCoffee() {
  console.log("お湯を沸かす");
  console.log("コーヒーを入れる");
}
// 呼び出し(実行)
makeCoffee();
```

```
// 挨拶をする関数
function greet(name) {
  console.log("こんにちは、" + name + "さん！");
}
// 呼び出し（実行）
greet("太郎"); // → こんにちは、太郎さん！
greet("花子"); // → こんにちは、花子さん！
```

引数（name）は、呼び出すときに外から値を渡すための変数です。