# 📚 「やること動的サイト」演習のための補足学習資料

この資料は、「やること動的サイト」の演習を進める上で役立つHTML、CSS、JavaScriptの重要な概念について解説します。演習中の穴埋め問題で迷ったときや、さらに理解を深めたいときに参照してください。

## HTML編：文書の骨格を作る

### 1. 基本的なHTML構造

- `<!DOCTYPE html>`: 文書型宣言。これがHTML5文書であることをブラウザに伝えます。
- `<html lang="ja">`: HTML文書全体のルート（根っこ）となる要素です。`lang="ja"` は、このページの主要言語が日本語であることを示します。
- `<head>`: このHTML文書に関するメタ情報（ブラウザや検索エンジン向けの裏設定）を記述する部分です。ページには直接表示されません。
  - `<meta charset="UTF-8">`: 文字エンコーディングをUTF-8に指定します。これにより、日本語などの多言語が正しく表示されます。
  - `<meta name="viewport" content="width=device-width, initial-scale=1.0">`: スマートフォンやタブレットなど、さまざまなデバイスでページが適切に表示されるようにするための設定です（レスポンシブデザインの基本）。
  - `<title>`: ブラウザのタブやブックマーク、検索結果のタイトルとして表示されるテキストを指定します。
  - `<link rel="stylesheet" href="style.css">`: このHTMLファイルに適用するCSSファイルを指定します。`href` 属性でCSSファイルの場所を指定します。
- `<body>`: 実際にブラウザのウィンドウに表示されるすべてのコンテンツ（テキスト、画像、リンクなど）を記述する部分です。
- `<script src="script.js"></script>`: JavaScriptファイルを読み込みます。通常、`</body>` タグの直前に記述することで、HTML要素の読み込みが完了した後にスクリプトが実行されるようにし、エラーを防ぎます。

### 2. セマンティックHTML

- **意味を持つタグを使うことの重要性**: HTMLタグは、コンテンツの「意味」や「役割」を示すために使います。例えば、`<header>` はページのヘッダー部分、`<nav>` はナビゲーションメニュー、`<main>` は主要なコンテンツ、`<footer>` はフッター部分といった具体的な意味を持ちます。
- **なぜ使うの？**:
  - **検索エンジン最適化（SEO）**: Googleなどの検索エンジンがページの内容を正しく理解しやすくなり、検索結果で上位に表示されやすくなる可能性があります。

- **アクセシビリティの向上**: スクリーンリーダー（視覚障碍者がウェブページを読むためのソフト）などがコンテンツの構造を理解しやすくなり、より多くの人が情報にアクセスできるようになります。
- **コードの可読性・保守性の向上**: `<div>`タグばかりで構成されたページよりも、意味のあるタグで構成されたページの方が、開発者にとって構造が理解しやすく、後からの修正や機能追加が容易になります。

- **代表的なセマンティックタグ**:
  - `<header>`: ページやセクションのヘッダー（導入部）。
  - `<nav>`: ナビゲーションリンクのグループ。
  - `<main>`: 文書の主要なコンテンツ。1ページに1つだけ使用します。
  - `<article>`: 独立した自己完結型のコンテンツ（ブログ記事、ニュース記事など）。
  - `<section>`: 文書内のテーマごとの区切り。通常、見出し（`<h1>`～`<h6>`）を含みます。
  - `<aside>`: メインコンテンツとは間接的に関連する補足情報（サイドバー、広告など）。
  - `<footer>`: ページやセクションのフッター（著者情報、コピーライトなど）。

## 3. `class` と `id` の違い

- `class`（クラス）:
  - 複数の要素に同じスタイルを適用したり、同じグループとして扱いたい場合に使用します。
  - 1つのHTML文書内で同じクラス名を複数の要素に使用できます。
  - 1つの要素に複数のクラス名をスペースで区切って指定することも可能です（例: `<div class="card featured">`）。
  - CSSやJavaScriptでスタイルや動作をまとめて指定するのに便利です。
- `id`（アイディー）:
  - HTML文書内で特定の要素を一意に識別するために使用します。
  - **1つのHTML文書内で同じID名を複数の要素に使用してはいけません。必ずユニーク（唯一）である必要があります。**
  - 主にJavaScriptから特定の要素を直接操作したり、ページ内リンクの飛び先として使用されます。

## 4. フォーム要素: `<input type="checkbox">` と `<label>`

- `<input type="checkbox">`: チェックボックスを作成します。ユーザーがオン/オフを選択できます。
- `<label for="ID名">`: `input` 要素と関連付けるラベル（説明テキスト）を作成します。
  - `for`属性の値は、関連付けたい `input` 要素の `id`属性の値と一致させます。
  - これにより、ラベルテキストをクリックしても、対応するチェックボックスがオン/オフされるようになり、操作性が向上します。 ```html

☐ HTMLの基本構造を理解する ¥`¥`¥`

# CSS編：見た目を美しく整える

## 1. CSSの基本構文

- CSSは「セレクタ」「プロパティ」「値」の3つの要素で構成されます。 ```css セレクタ { プロパティ: 値; プロパティ2: 値2; } ```
- **セレクタ**: スタイルを適用したいHTML要素を指定します。（例: `body`, `.main-header`, `#task1`）
- **プロパティ**: 変更したいスタイルの種類を指定します。（例: `color`, `font-size`, `background-color`）
- **値**: プロパティに設定する具体的な内容を指定します。（例: `red`, `16px`, `#ffffff`）

## 2. `box-sizing: border-box;`

- 要素の `width`（幅）や `height`（高さ）の計算方法を指定する重要なプロパティです。
- デフォルト（`content-box`）では、`width` や `height` はコンテンツ領域のみのサイズを示し、`padding`（内側の余白）や `border`（枠線）の幅はそれに加算されます。
- `border-box` を指定すると、`width` や `height` に `padding` と `border` の幅が含まれるようになります。これにより、要素の最終的な表示サイズが指定した `width` や `height` と一致するため、レイアウト計算が直感的になり、扱いやすくなります。 ```css
  - { /* 全ての要素に適用 */ box-sizing: border-box; } ```

## 3. CSSの単位

- `px`（ピクセル）: 絶対単位。画面上の1ピクセルに対応します。
- `rem`（root em）: ルート要素（`<html>`）のフォントサイズを基準とする相対単位。ユーザーがブラウザの基本フォントサイズを変更した場合にも柔軟に対応できるため、アクセシビリティに配慮したサイト制作でよく使われます。
- `%`（パーセント）: 親要素の同じプロパティの値を基準とする相対単位。
- `vh`（viewport height）: ビューポート（ブラウザの表示領域）の高さに対する割合。`100vh` はビューポートの高さ全体を意味します。
- `vw`（viewport width）: ビューポートの幅に対する割合。

## 4. `display: flex;`（Flexbox）

- 要素を柔軟に配置するための強力なレイアウトモードです。アイテムを横並びにしたり、間隔を調整したり、中央揃えにしたりするのが容易になります。
- 親要素に `display: flex;` を指定すると、その子要素（フレックスアイテム）がFlexboxのレイアウトに従います。
- `align-items: center;`: フレックスアイテムを交差軸（デフォルトでは垂直方向）の中央に配置します。
- `justify-content: space-between;`: フレックスアイテムを主軸（デフォルトでは水平方向）に沿って均等に配置し、最初と最後のアイテムは両端に寄せます。

- `flex: 1;`: フレックスアイテムがコンテナ内の余剰スペースをどれだけ占めるかを指定します。`1` を指定すると、他のアイテムがスペースを占めた残りの部分をすべて使おうとします。

## 5. `position` プロパティと `z-index`

- `position`: 要素の配置方法を指定します。
  - `static`（デフォルト）: 通常のHTMLのフローに従って配置されます。`top`, `left` などは効きません。
  - `relative`: 通常のフローに従いつつ、`top`, `right`, `bottom`, `left` プロパティで相対的な位置調整が可能です。
  - `absolute`: 最も近い `position` が `static` 以外の上位要素（親要素など）を基準に配置されます。そのような上位要素がない場合は、`body` 要素が基準になります。
  - `fixed`: ビューポート（ブラウザの表示領域）を基準に固定配置されます。ページをスクロールしても位置が変わりません。（例: 画面上部に固定されるヘッダー、モーダルウィンドウなど）
- `z-index`: 要素の重なり順を指定します。`position` が `static` 以外の場合に有効です。数値が大きいほど手前（上）に表示されます。

## 6. `:hover` 擬似クラスと `transition` プロパティ

- `:hover`: マウスカーソルが要素の上に乗ったときのスタイルを指定するための擬似クラスです。 ```css .button:hover { background-color: lightblue; } ```
- `transition`: CSSプロパティの値が変化する際に、アニメーション効果（滑らかな変化）を付けるためのプロパティです。
  - `transition-property`: アニメーションを適用するCSSプロパティ名（例: `background-color`, `transform`, `all` ですべてのプロパティ）。
  - `transition-duration`: アニメーションにかかる時間（例: `0.3s`, `500ms`）。
  - `transition-timing-function`: アニメーションの速度変化のパターン（例: `ease`（ゆっくり始まってゆっくり終わる）, `linear`（一定速度）, `ease-in-out`）。
  - `transition-delay`: アニメーションが始まるまでの遅延時間。 ```css .task-item { background-color: white; transition: background-color 0.3s ease; /* background-colorが0.3秒かけて滑らかに変化 */ } .task-item:hover { background-color: #f0f0f0; } ```

## 7. レスポンシブデザインと `@media` クエリ

- `@media` クエリは、画面の幅や高さ、デバイスの種類など、特定の条件に応じて異なるCSSスタイルを適用するための仕組みです。これにより、スマートフォン、タブレット、デスクトップなど、さまざまな画面サイズでウェブページが見やすくなるように調整できます（レスポンシブデザイン）。 ```css /* 基本のスタイル（主にデスクトップ向け） */ .container { width: 960px; margin: 0 auto; }

/* 画面幅が768px以下の場合に適用されるスタイル（主にタブレット向け）
/ @media (max-width: 768px) { .container { width: 90%; / 幅を可変にす

る */ } }

/* 画面幅が480px以下の場合に適用されるスタイル（主にスマートフォン向け） */ @media (max-width: 480px) { body { font-size: 14px; } 文字サイズを少し小さくする */ } .main-header h1 { font-size: 1.8rem; } } ```

## JavaScript編：ウェブページに動きと対話性を加える

### 1. DOM (Document Object Model)

- ブラウザがHTML文書を読み込むと、その内容をツリー構造のオブジェクトとしてメモリ上に展開します。このオブジェクト構造のことをDOMと呼びます。
- JavaScriptは、このDOMを介してHTMLの要素を取得したり、内容を変更したり、新しい要素を追加したり、イベントを処理したりすることができます。つまり、ウェブページを動的に操作するためのインターフェースです。

### 2. `DOMContentLoaded` イベント

- HTML文書が完全に読み込まれ、DOMツリーの構築が完了した時点で発生するイベントです。
- 画像やCSSファイルなどの外部リソースの読み込み完了を待たずに実行されるため、スクリプトがDOM要素を操作しようとして、まだその要素が存在しないためにエラーになるのを防ぐのに役立ちます。
- JavaScriptのコードは、このイベントが発生した後に実行されるようにするのが一般的です。 ```javascript document.addEventListener('DOMContentLoaded', function() { // この中にDOM操作を行うコードを記述する console.log('DOMの準備ができました！'); }); ```

### 3. HTML要素の取得

- JavaScriptでHTML要素を操作するには、まず対象の要素を取得する必要があります。
- `document.getElementById('id名')`: 指定された `id` 属性を持つ要素を1つ取得します。IDはページ内で一意なので、確実に特定の要素を取得できます。 ```javascript const mainTitle = document.getElementById('main-title'); ```
- `document.querySelector('CSSセレクタ')`: 指定されたCSSセレクタに一致する最初の要素を1つ取得します。複雑な条件で要素を指定できます。 ```javascript const firstTaskItem = document.querySelector('.task-item'); ```
- `document.querySelectorAll('CSSセレクタ')`: 指定されたCSSセレクタに一致するすべての要素を `NodeList` （配列のようなもの）として取得します。複数の要素をまとめて操作したい場合に使用します。 ```javascript const allCheckboxes = document.querySelectorAll('.task-checkbox'); ```
- `element.closest('CSSセレクタ')`: `element` 自身を含め、その祖先要素をたどっていき、指定されたCSSセレクタに最初に一致した要素を返します。イベントが発生した要素から特定の親要素を見つけるのに便利です。 ```javascript // checkboxがクリックされたとき、その親の.task-item要素を取得する checkbox.addEventListener('change', function() { const taskItem = this.closest('.task-item'); }); ```

### 4. イベントリスナー（`addEventListener`）

- ユーザーのアクション（クリック、マウスオーバー、キー入力など）やブラウザの出来事（ページの読み込み完了など）を「イベント」と呼びます。
- `element.addEventListener('イベント名', コールバック関数);` は、特定の要素で指定したイベントが発生したときに、指定した関数（コールバック関数）を実行するように登録するメソッドです。
  - `'click'`: 要素がクリックされたとき。
  - `'change'`: フォーム要素（`<input>`, `<select>`, `<textarea>`など）の値が変更されたとき。チェックボックスのオン/オフもこれに該当します。
  - `'input'`: テキスト入力フィールドの値が変更されるたび（一文字ごとなど）。
  - `'mouseover'`: マウスカーソルが要素の上に乗ったとき。
  - `'mouseout'`: マウスカーソルが要素の上から離れたとき。 ```javascript const myButton = document.getElementById('myButton'); myButton.addEventListener('click', function() { alert('ボタンがクリックされました！'); }); ```

## 5. クラスの操作（`element.classList`）

- JavaScriptを使ってHTML要素のクラスを動的に追加、削除、確認することができます。これにより、CSSと連携して要素の見た目を変更できます。
- `element.classList.add('クラス名')`: 指定したクラスを要素に追加します。
- `element.classList.remove('クラス名')`: 指定したクラスを要素から削除します。
- `element.classList.toggle('クラス名')`: 指定したクラスが要素になければ追加し、あれば削除します。
- `element.classList.contains('クラス名')`: 要素が指定したクラスを持っているかどうかを真偽値（`true` / `false`）で返します。 ```javascript const messageDiv = document.getElementById('message'); messageDiv.classList.add('highlight'); // 'highlight'クラスを追加 if (messageDiv.classList.contains('error')) { // 'error'クラスを持っているか確認 console.log('エラーメッセージです'); } ```

## 6. スタイルの直接変更（`element.style`）

- `element.style.プロパティ名 = '値';` の形で、JavaScriptから直接要素のインラインスタイルを変更できます。
- CSSのプロパティ名がハイフンを含む場合（例: `background-color`）、JavaScriptではキャメルケース（例: `backgroundColor`）で記述します。 ```javascript const titleElement = document.getElementById('page-title'); titleElement.style.color = 'blue'; titleElement.style.fontSize = '24px'; titleElement.style.backgroundColor = '#f0f0f0'; // background-color -> backgroundColor ```
- ただし、多くのスタイル変更はCSSクラスの付け外しで行う方が、コードの分離（関心事の分離）や保守性の観点から推奨されます。

## 7. テキストコンテンツの変更

- `element.textContent = '新しいテキスト';`: 要素内のテキストコンテンツを取得したり、設定したりします。HTMLタグは解釈されず、そのまま文字列として扱われます。セキュリティ的に安全です。

- `element.innerHTML = '新しいHTMLコンテンツ';`: 要素内のHTMLコンテンツを取得したり、設定したりします。HTMLタグは解釈されて表示に反映されます。ユーザーからの入力をそのまま `innerHTML` に設定すると、クロスサイトスクリプティング（XSS）の脆弱性を生む可能性があるため、取り扱いに注意が必要です。 ```javascript const userNameElement = document.getElementById('user-name'); userNameElement.textContent = '山田太郎'; // テキストとして設定

const greetingElement = document.getElementById('greeting'); // greetingElement.innerHTML = '**こんにちは！**'; // HTMLとして設定（注意して使用）```

## 8. `forEach` ループ（NodeListに対して）

- `document.querySelectorAll()` で取得した `NodeList` は配列に似ていますが、一部の古いブラウザでは配列のすべてのメソッドが使えないことがあります。しかし、`forEach` メソッドは多くのモダンブラウザで `NodeList` に対しても使用できます。
- `NodeList` の各要素に対して、指定した関数を一度ずつ実行します。 ```javascript const allTaskItems = document.querySelectorAll('.task-item'); allTaskItems.forEach(function(item, index) { // item は現在の要素, index はそのインデックス番号 item.style.borderLeft = `5px solid hsl(${index * 60}, 70%, 50%)`; }); ```

## 9. 条件分岐（`if...else`）

- 特定の条件が真（true）か偽（false）かによって、実行する処理を分岐させます。 ```javascript const score = 75; if (score >= 80) { console.log('素晴らしい！合格です。'); } else if (score >= 60) { console.log('合格です。'); } else { console.log('残念ながら不合格です。'); }

// チェックボックスの状態を確認 const myCheckbox = document.getElementById('my-checkbox'); if (myCheckbox.checked) { // .checked プロパティでチェック状態を取得（true/false） console.log('チェックされています'); } else { console.log('チェックされていません'); } ```