発展CSS

CSSの基本を学んで、文字の色やサイズ、背景、余白などを変えられるようになりましたね。今回は、CSSの力をさらに引き出すためのテクニックを見ていきましょう！

よりピンポイントで要素を指定する方法、要素に動きや変化を与える方法、そしてCSSをもっと効率的に書くためのテクニックなどを紹介します。これらをマスターすれば、デザインの自由度が格段に上がり、より洗練された、そしてメンテナンスしやすいWebページを作れるようになります。

1. 高度なセレクタ：狙った要素を逃さない

基本的な要素セレクタ、クラスセレクタ、IDセレクタ以外にも、CSSにはもっと細かく要素を指定するためのセレクタがたくさんあります。これらを使いこなせると、「特定の条件下にある要素だけ」にスタイルを適用できるようになります。

疑似クラス（Pseudo-classes）

要素の「特定の状態」や「位置」に基づいてスタイルを適用するためのセレクタです。セレクタの後ろにコロン（`:`）を付けて記述します。

- **`:hover`**: マウスカーソルが要素の上に乗っている状態。

```
a:hover {
  color: red; /* リンクにマウスを乗せると赤色に */
  text-decoration: underline; /* 下線をつける */
}
button:hover {
  opacity: 0.8; /* ボタンにマウスを乗せると少し透明に */
}
```

- **`:active`**: 要素がアクティブな状態（例: マウスでクリックされている間）。

```
button:active {
  transform: scale(0.95); /* クリック中に少し小さくする */
}
```

- **`:focus`**: 要素がフォーカスを受け取っている状態（例: テキスト入力欄が選択されている時、タブキーで移動した時）。

```
input:focus {
  outline: 2px solid blue; /* フォーカス時に青い枠線を表示 */
```

```
  background-color: lightyellow;
}
```

- **:nth-child(n)**: 親要素の中でn番目の子要素。`n` には数字、`odd`（奇数番目）、`even`（偶数番目）、`2n+1`（数式）などが指定できます。

```
/* リストの奇数番目の背景色を変える（ストライプ）*/
li:nth-child(odd) {
  background-color: #f2f2f2;
}
/* 3番目の段落だけ文字を太くする */
p:nth-child(3) {
  font-weight: bold;
}
/* 最初の2つのアイテム */
li:nth-child(-n+2) {
  border-top: 2px solid red;
}
```

- **:first-child**: 親要素の中で最初の子要素。`:nth-child(1)` と同じ。
- **:last-child**: 親要素の中で最後の子要素。
- **:not(セレクタ)**: 指定したセレクタに一致「しない」要素。

```
/* ”special” クラスが付いていない p 要素すべて */
p:not(.special) {
  color: gray;
}
/* 最後のリスト項目以外のリスト項目に下線を引く */
li:not(:last-child) {
  border-bottom: 1px solid #ccc;
}
```

開発者の視点: 疑似クラス、特に `:hover` や `:focus` は、ユーザーインタラクション（操作に対する反応）を分かりやすくするために非常によく使われます。`:nth-child()` はテーブルの行を交互に色付けしたり、リストのデザインを調整したりするのに便利です。

疑似要素（Pseudo-elements）

要素の「特定の部分」に対してスタイルを適用するためのセレクタです。セレクタの後ろにコロン2つ（`::`）を付けて記述します（古いブラウザとの互換性のために `:` 1つでも動くことが多いですが、`::` が推奨されます）。

- **::before**: 要素の内容の「直前」に、CSSからコンテンツ（テキストや画像など）を挿入します。`content` プロパティが必須です。
- **::after**: 要素の内容の「直後」に、CSSからコンテンツを挿入します。`content` プロパティが必須です。

```
/* 外部リンクの後ろにアイコンを表示する */
a[target="_blank"]::after {
  content: " ↗"; /* contentプロパティで表示内容を指定 */
  font-size: 0.8em;
}
/* 特定のセクションタイトルの前に装飾を付ける */
h2.section-title::before {
  content: "◆ ";
  color: blue;
}
/* clearfix ハック（float解除の古典的なテクニック）*/
.clearfix::after {
  content: "";
  display: block;
  clear: both;
}
```

- **::first-letter**: ブロックレベル要素の最初の文字。
- **::first-line**: ブロックレベル要素の最初の行。

```
p::first-letter {
  font-size: 2em; /* 最初の文字だけ大きく */
  color: red;
  float: left; /* 回り込み */
  margin-right: 0.1em;
}
p::first-line {
  font-weight: bold; /* 最初の行だけ太字 */
}
```

- **::selection**: ユーザーがマウスなどで選択した部分。

```
::selection {
  background-color: yellow;
  color: black;
}
```

開発者の視点: `::before` と `::after` は非常に強力で、HTMLを変更せずに装飾的な要素を追加したり、レイアウトの補助に使ったりできます。`content` プロパティが必須なのを忘れずに。

属性セレクタ（Attribute Selectors）

HTML要素が持つ属性やその値に基づいて要素を選択します。

- `[attribute]`: 指定した属性を持つ要素。

```
/* title属性を持つすべての要素 */
[title] {
  border-bottom: 1px dotted gray;
}
```

- `[attribute="value"]`: 指定した属性が特定の値を持つ要素。

```
/* type="submit" の input 要素 */
input[type="submit"] {
  background-color: green;
  color: white;
}
```

- `[attribute~="value"]`: 指定した属性の値（スペース区切り）に特定の単語が含まれる要素。

```
<img src="a.jpg" alt="dog cute">
<img src="b.jpg" alt="cat photo">
```

```
/* alt属性に "cute" が含まれる img 要素 */
img[alt~="cute"] {
  border: 2px solid pink;
}
```

- `[attribute^="value"]`: 指定した属性の値が特定の文字列で「始まる」要素。

```
/* href属性が "https://" で始まる a 要素（外部リンク）*/
a[href^="https://"] {
  background: url(external-link.png) no-repeat right center;
  padding-right: 15px;
}
```

- `[attribute$="value"]`: 指定した属性の値が特定の文字列で「終わる」要素。

```
/* href属性が ".pdf" で終わる a 要素（PDFリンク）*/
a[href$=".pdf"]::after {
  content: " (PDF)";
}
```

- `[attribute*="value"]`: 指定した属性の値に特定の文字列が「含まれる」要素。

```
/* class属性に "test" という文字列が含まれる要素（例: class="button test primary"）*/
[class*="test"] {
  color: red;
}
```

結合子（Combinators）

複数のセレクタを組み合わせて、要素間の関係性に基づいて選択します。

- **子孫セレクタ（ ）**（スペース）: ある要素の「中にあるすべて」の子孫要素。

```
/* div要素の中にあるすべてのp要素 */
div p {
  color: blue;
}
```

- **子セレクタ（>）**: ある要素の「直下にある」子要素。

```
/* ul要素の直下にあるli要素（孫要素は対象外）*/
ul > li {
  list-style-type: square;
}
```

- **隣接兄弟セレクタ（+）**: ある要素の「すぐ隣にある」弟要素（同じ親を持つ）。

```
/* h2要素のすぐ後ろにあるp要素 */
h2 + p {
  margin-top: 0; /* 見出し直後の段落の上マージンを詰める */
}
```

- **一般兄弟セレクタ（~）**: ある要素の「後ろにあるすべての」弟要素（同じ親を持つ）。

```
/* h2要素の後ろにあるすべてのp要素 */
h2 ~ p {
  text-indent: 1em; /* 字下げ */
}
```

開発者の視点: 結合子をうまく使うと、HTMLに余計なクラスを付けなくても、構造に基づいてスタイルを適用できます。ただし、深くネストした子孫セレクタ（例: `#main .content .box p`）は、CSSの特定度（Specificity）が高くなりすぎて、後で上書きしにくくなったり、HTML構造の変更に弱くなったりするので、使いすぎには注意しましょう。子セレクタ `>` を使う方が影響範囲を限定できて良い場合が多いです。

2. 視覚効果とアニメーション：動きと変化を加える

CSSを使えば、要素の見た目を変化させたり、アニメーションを付けたりすることもできます。Webページがぐっと魅力的になりますよ。

`transition`：滑らかな変化

プロパティの値が変化する際に、時間をかけて滑らかに変化させる効果を追加します。`:hover` などと組み合わせることが多いです。

- `transition-property`: 変化を適用するプロパティ名（例: `background-color`, `opacity`, `all`）。
- `transition-duration`: 変化にかかる時間（例: `0.3s`, `500ms`）。
- `transition-timing-function`: 変化の速度曲線（例: `ease`, `linear`, `ease-in-out`）。
- `transition-delay`: 変化が始まるまでの遅延時間。

まとめて `transition` プロパティで指定するのが一般的です。

```
button {
  background-color: blue;
  color: white;
  padding: 10px 20px;
  border: none;
  border-radius: 5px;
  /* 変化させたいプロパティ、時間、速度曲線を指定 */
  transition: background-color 0.3s ease, transform 0.2s ease-out;
}

button:hover {
  background-color: darkblue;
  transform: scale(1.05); /* 少し拡大 */
}
```

`transform`: 要素の変形

要素を移動、回転、拡大縮小、傾斜させることができます。レイアウトに影響を与えずに見た目を変形できるのが特徴です。

- `translate(x, y)`: 水平方向(x)、垂直方向(y)に移動。`translateX(x)`, `translateY(y)` もあります。
- `rotate(angle)`: 指定した角度だけ回転（例: `45deg`）。
- `scale(x, y)`: 水平方向(x)、垂直方向(y)に拡大縮小。`scale(1.5)` のように一つだけ指定すると縦横比維持。`scaleX(x)`, `scaleY(y)` もあります。
- `skew(x-angle, y-angle)`: 水平方向(x)、垂直方向(y)に傾斜。`skewX(angle)`, `skewY(angle)` もあります。

複数の変形はスペースで区切って指定します。

```
img:hover {
  transform: rotate(5deg) scale(1.1); /* 少し回転して拡大 */
  transition: transform 0.3s ease; /* transformの変化を滑らかに */
}

.icon {
  display: inline-block; /* transformはインライン要素には効きにくい */
  transition: transform 0.2s ease;
}
.icon:active {
```

```
  transform: translateY(2px); /* クリック時に少し下に沈む */
}
```

`animation` と `@keyframes`：複雑なアニメーション

`transition` よりも複雑な、キーフレームに基づいたアニメーションを作成できます。

1. **`@keyframes` ルール**: アニメーションの名前と、そのアニメーション中のどの時点（`0%` または `from`, `50%`, `100%` または `to` など）でどのようなスタイルにするかを定義します。
2. **`animation` プロパティ**: 要素に `@keyframes` で定義したアニメーションを適用します。
    - `animation-name`: `@keyframes` で定義した名前。
    - `animation-duration`: アニメーション1回にかかる時間。
    - `animation-timing-function`: 速度曲線。
    - `animation-delay`: 開始までの遅延時間。
    - `animation-iteration-count`: 繰り返す回数（`infinite` で無限ループ）。
    - `animation-direction`: 再生方向（`normal`, `reverse`, `alternate` など）。
    - `animation-fill-mode`: アニメーション開始前・終了後のスタイル（`none`, `forwards`, `backwards`, `both`）。
    - `animation-play-state`: 再生状態（`running`, `paused`）。

```
/* 点滅するアニメーションを定義 */
@keyframes blink {
  0% { opacity: 1; }
  50% { opacity: 0.2; }
  100% { opacity: 1; }
}

/* ローディングスピナーのアニメーションを定義 */
@keyframes spin {
  from { transform: rotate(0deg); }
  to { transform: rotate(360deg); }
}

.alert {
  /* blinkアニメーションを2秒かけて無限に繰り返す */
  animation: blink 2s infinite;
}

.spinner {
  width: 50px;
  height: 50px;
  border: 5px solid lightgray;
  border-top-color: blue;
  border-radius: 50%; /* 円形にする */
  /* spinアニメーションを1秒で線形に無限ループ */
  animation: spin 1s linear infinite;
}
```

開発者の視点: アニメーションはユーザーの注意を引きつけたり、状態変化を分かりやすく伝えたりするのに有効ですが、やりすぎると鬱陶しくなったり、パフォーマンスに影響したりします。特に `opacity` と `transform` を使ったアニメーションは比較的パフォーマンスが良いとされています。

その他の視覚効果

- **`filter`**: 要素に画像フィルターのような効果（ぼかし、セピア調、グレースケール、明るさ調整など）を適用します。

```
img.blur {
  filter: blur(5px); /* 5pxぼかす */
}
img.grayscale:hover {
  filter: grayscale(0%); /* ホバーで色を戻す */
  transition: filter 0.3s ease;
}
img.grayscale {
  filter: grayscale(100%); /* 最初はグレースケール */
}
```

- **`opacity`**: 要素の透明度を指定します（`0` で完全透明、`1` で完全不透明）。
- **`box-shadow`** / **`text-shadow`**: 要素のボックスやテキストに影を付けます。複数の影を指定することも可能です。

```
.card {
  box-shadow: 5px 5px 10px rgba(0, 0, 0, 0.2); /* 右下にぼけた影 */
}
h1 {
  text-shadow: 1px 1px 2px gray; /* テキストに少し影 */
}
```

- **グラデーション**: `background-image` プロパティで線形（`linear-gradient()`）や放射状（`radial-gradient()`）のグラデーション背景を作成できます。

```
body {
  /* 上から下へ、白から水色へのグラデーション */
  background-image: linear-gradient(to bottom, white, lightblue);
}
.button-shiny {
  /* 左上から右下へ、明るい青から暗い青へのグラデーション */
  background-image: linear-gradient(to bottom right, #5bc0de, #0275d8);
  color: white;
}
```

3. その他：効率化と便利な機能

CSSをより効率的に、そして柔軟に書くための機能も見てみましょう。

CSS変数（カスタムプロパティ）

CSS内で独自の変数（カスタムプロパティ）を定義し、再利用することができます。色の設定や余白のサイズなど、サイト全体で統一したい値を一元管理するのに非常に便利です。

- 変数の定義: `--`（ハイフン2つ）で始まる名前を付け、`:root` 疑似クラス（HTML文書全体を指す）や特定のセレクタ内で定義します。
- 変数の利用: `var()` 関数を使って呼び出します。

```
/* :root でグローバルな変数を定義 */
:root {
  --main-color: #3498db; /* メインカラー */
  --accent-color: #f1c40f; /* アクセントカラー */
  --base-font-size: 16px;
  --padding-small: 8px;
  --padding-medium: 16px;
}

body {
  font-size: var(--base-font-size);
}

h1 {
  color: var(--main-color);
}

button {
  background-color: var(--main-color);
  color: white;
  padding: var(--padding-small) var(--padding-medium);
  border: none;
  cursor: pointer;
  transition: background-color 0.3s ease;
}

button:hover {
  background-color: darken(var(--main-color), 10%); /* Sassなどのプリプロセッサが必要な場合あり
*/
  /* もしくは直接色を指定するか、ホバー用の変数を定義 */
}

.highlight {
  background-color: var(--accent-color);
  padding: var(--padding-small);
}
```

開発者の視点: CSS変数は、特に大規模なプロジェクトや、テーマ変更機能（ダークモードなど）を実装する際に絶大な効果を発揮します。色の変更などが一箇所で済むため、メンテナンス性が劇的に向上します。

`calc()` 関数

CSSプロパティの値の中で計算を行うことができます。異なる単位（例: `%` と `px`）を組み合わせた計算も可能です。

```
.container {
  width: calc(100% - 40px); /* 全体の幅から左右20pxずつの余白を引いた幅 */
  padding: 20px;
  margin: 0 auto; /* 中央揃え */
}

header {
  height: 60px;
}

main {
  /* ヘッダーの高さを引いた残りの高さを指定 */
  min-height: calc(100vh - 60px); /* vhはビューポートの高さに対する割合 */
}

.item {
  /* 親要素の幅の1/3から、アイテム間のマージン分を引く */
  width: calc(100% / 3 - 20px);
  margin: 10px;
}
```

Webフォント

コンピューターにインストールされていないフォントでも、Webサーバーからフォントファイルを読み込んで表示させる技術です。`@font-face` ルールを使うか、Google Fonts などの外部サービスを利用するのが一般的です。

`@font-face` の例:

```
/* フォントファイルを読み込む */
@font-face {
  font-family: 'MyCustomFont'; /* フォント名を定義 */
  src: url('myfont.woff2') format('woff2'), /* 最新フォーマットから順に */
       url('myfont.woff') format('woff');   /* 古いブラウザ向け */
  font-weight: normal;
  font-style: normal;
}

/* 定義したフォントを使う */
body {
  font-family: 'MyCustomFont', sans-serif; /* 読み込めなかった場合の代替フォントも指定 */
}
```

Google Fonts の利用例 (HTML):

```
<head>
  <link rel="preconnect" href="https://fonts.googleapis.com">
  <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
  <link href="https://fonts.googleapis.com/css2?family=Noto+Sans+JP:wght@400;700&display=swap"
rel="stylesheet">
</head>
```

Google Fonts の利用例 (CSS):

```
body {
  font-family: 'Noto Sans JP', sans-serif; /* Google Fontsで指定されたフォント名 */
}
h1, h2 {
  font-weight: 700; /* 読み込んだウェイトを指定 */
}
```

開発者の視点: Webフォントを使うとデザインの幅が広がりますが、フォントファイルの読み込みには時間がかかるため、ページの表示速度に影響を与える可能性があります。必要な文字だけを含むサブセット化されたフォントを使ったり、読み込み方法を工夫したり（`font-display: swap;` など）することが重要です。

背景の高度な設定

`background` プロパティは、色だけでなく画像やサイズ、位置なども細かく指定できます。

- `background-image`: 背景画像を指定（`url('path/to/image.jpg')`）。グラデーションもここで指定。
- `background-size`: 背景画像のサイズ（`cover`, `contain`, `50%`, `100px 50px` など）。
- `background-position`: 背景画像の表示位置（`center`, `top left`, `50% 50%` など）。
- `background-repeat`: 背景画像の繰り返し（`no-repeat`, `repeat-x`, `repeat-y`, `repeat`）。
- `background-attachment`: 背景画像をスクロールに追従させるか固定するか（`scroll`, `fixed`）。

まとめて `background` プロパティで指定することも可能です。また、カンマ区切りで複数の背景画像を指定することもできます。

```
body {
  background-color: #f0f0f0;
  background-image: url('pattern.png'), linear-gradient(to bottom, rgba(255,255,255,0.8),
rgba(255,255,255,0)); /* 画像とグラデーションを重ねる */
  background-repeat: repeat, no-repeat;
  background-position: top left, center;
  background-size: auto, cover;
}

.hero-section {
  background-image: url('hero.jpg');
  background-size: cover; /* 要素全体を覆うように画像を拡大縮小 */
  background-position: center; /* 画像の中央を表示 */
  background-repeat: no-repeat;
```

```
  height: 400px;
  color: white;
  display: flex; /* 中身を中央揃えするため（次の資料で解説）*/
  justify-content: center;
  align-items: center;
  text-shadow: 1px 1px 3px black;
}
```

4. まとめ：表現力と効率を手に入れる

今回は、CSSのさらに奥深い世界を探検しました。

- **高度なセレクタ**（疑似クラス、疑似要素、属性セレクタ、結合子）で、狙った要素にピンポイントでスタイルを適用できるようになりました。
- `transition`, `transform`, `animation` などを使って、Webページに動きや滑らかな変化、アニメーションを加える方法を学びました。
- `filter`, `opacity`, `box-shadow`, **グラデーション** などで、より豊かな視覚表現が可能になりました。
- **CSS変数**, `calc()`, **Webフォント**, **背景の高度な設定** など、効率的で柔軟なCSSコーディングに役立つ機能を知りました。

これらのテクニックを駆使することで、静的なデザインだけでなく、ユーザーの操作に反応したり、動きで情報を伝えたりする、よりインタラクティブで魅力的なWebページを作成できます。

特にCSS変数は、コードの保守性を高める上で非常に強力なツールです。積極的に活用していくことをお勧めします。

次の資料では、いよいよ現代的なWebレイアウトの要である **Flexbox** と **Grid**、そして様々な画面サイズに対応するための **レスポンシブデザイン** について学んでいきます。これで、どんなデバイスでも見やすいWebページを作るスキルが身につきますよ！