

Lecture-02: CSS基礎 (60~90分)

目次と時間配分

1. 導入：CSSとは/適用・読み込み方法（10分）
2. セレクタと詳細度（15分）
3. ボックスモデル/表示/間隔（15分）
4. 値と単位/色/タイポグラフィ（10分）
5. まとめ/Q&A（5分）

CSSとはなにか

- ・見た目（色/余白/枠線/影/タイポ）とレイアウト（配置/整列/段組）を定義する言語
- ・HTMLが表す「構造/意味」に対し、CSSは「表現/一部の振る舞い（状態・アニメーション）」を担う
- ・競合解決の柱：カスケード/継承/詳細度（Specificity）

CSSOMとは？

- CSS Object Model。スタイルシートをパースして得られるツリー構造
- セレクタ照合・継承・詳細度・読み込み順を反映した「計算可能なスタイル情報」
- DOMと結合して描画用の「レンダーツリー」を構築する基盤

カスケードが適用される順序

1. 起源 (User Agent < User < Author)
2. 重要度 (`!important`)
3. 詳細度 (ID > クラス/属性/疑似クラス > 要素/疑似要素)
4. 出現順 (あと勝ち)

値の段階（よく見る用語）

- 指定値 (specified value) : ルールや継承から決まる最初の値
- 計算値 (computed value) : 相対単位や依存関係を解決した値 (DevToolsの Computedが近い)
- 使用値 (used value) / 実行値 (actual value) : レイアウト/フォントなど最終的な計算を経た値

再計算とパフォーマンスの要点

- スタイルが変わると「スタイル再計算 (style recalculation)」→ 要素やツリーの一部を再評価
- レイアウトに影響する場合は「レイアウト (reflow)」→ その後「ペイント/合成」
- 深すぎるセレクタ/複雑な疑似クラス (例: `:has()` の多用) は再計算コスト増の一因
- 実務Tips: セレクタは短く/局所的に、BEMなどで対象要素を特定しやすくする

ブラウザの流れ（おさらい）

1. HTMLをパース → DOMツリーを構築
2. CSSをパース → CSSOMツリーを構築
3. DOM + CSSOM → レンダーツリー生成（見える要素のみ）
4. レイアウト（各要素のサイズ/位置を計算）
5. ペイント/コンポジット（塗り/合成）

HTML → DOM

CSS → CSSOM

DOM + CSSOM → Render Tree → Layout → Paint/Composite

DevToolsヒント: Elements → Styles/Computedタブで「どのルールが最終採用されたか」「計算後の値」を確認できる

はじめに（目的/到達点）

- CSSの役割と基本概念を理解し、Flexboxで一次元レイアウトを作れる
- 簡単なレスポンシブ対応（メディアクエリ）に触れる

目次と時間配分

1. 導入：CSSとは/適用・読み込み方法（10分）
2. セレクタと詳細度（15分）
3. ボックスモデル/表示/間隔（15分）
4. 値と単位/色/タイポグラフィ（10分）
5. まとめ/Q&A（5分）

CSSの適用/読み込み（外部/内部/インライン、読み込み順、 @import、preload、media、CDN/パス）

- 外部CSS（推奨）
- インラインCSS（原則避ける）
- 実務では基本「外部CSS」一択

外部CSS（推奨）

```
<link rel="stylesheet" href=".//styles/app.css">
```

- キャッシュが効く／複数HTMLで再利用しやすい
- `<head>` で早めに読み込む（CSSはレンダーブロッキングだが表示に必要）
- 論理で分割：`base.css` / `components.css` / `page-*.css` （HTTP/2なら過度な結合は不要）
- キャッシュバusting：ファイル名ハッシュ or `?v=YYYYMMDD` などで更新を確実に反映
- 上書きの意図は読み込み順で表現（後ろほど強くなる前提で設計）

インラインCSS（原則避ける）

```
<button style="color: white; background: #2563eb;">送信</button>
```

- 再利用性/保守性が低い。
- 詳細度が最強（`style=""`）のため、後からの上書きが難しくなりやすい
- 代替案：ユーティリティクラスや `data-*` 属性を使い、JSでクラス切替

読み込み順と優先順位（カスケードの入口）

- 後から読み込まれたスタイルが勝ちやすい（詳細度が同等の場合）
- CSSは表示に不可欠なので原則 `<head>` 内に記載する。
- 外部UIライブラリの上書きは「自作の上書きCSSを後ろ」に置くことで衝突を防ぐ

media 属性で条件付き読み込み

```
<link rel="stylesheet" href="/styles/print.css" media="print">  
<link rel="stylesheet" href="/styles/wide.css" media="(min-width: 1280px)">
```

- 不要時に評価されずパフォーマンスに有利。
- `print` 専用CSSを分離すると本番ページのバイト削減に有効
- ファイル分割はHTTP/2以降で許容度が高いが、細切れすぎると逆効果

重要CSSを **preload** (フォールバック付)

```
<link rel="preload" href="/styles/above-the-fold.css" as="style" onload="this.rel='stylesheet'>
<noscript><link rel="stylesheet" href="/styles/above-the-fold.css"></noscript>
```

- ファーストビューに必要なCSSの先読み。過度な分割は逆効果。
- 対象は「折り返し前 (above-the-fold)」の最小限 (数KB～十数KB) に限定
- LCP/FCPの改善を狙って使い、計測 (Lighthouse/Core Web Vitals) とセットで検証

パス解像度（相対/ルート相対）

```
<!-- 現在HTMLからの相対 -->
<link rel="stylesheet" href="./styles/app.css">
<!-- サイトルートからの相対 -->
<link rel="stylesheet" href="/assets/styles/app.css">
```

- 本リポでは `lecture-02/` 配下に置くため、相対パスでの指定が分かりやすい。
- GitHub Pagesなどサブディレクトリ配信では「ルート相対」が想定通りに解決されない場合がある（`<base href>` 等で調整）

ベストプラクティス（まとめ）

- 外部CSS + `link` が基本。
- CSSは意味ごとに分割し、読み込み順で意図を表現
- 条件付は `media`、初期表示に効くものは `preload`

カスケード/継承/詳細度

カスケード (Cascade)

競合時の最終決定は

1. 「起源 (ブラウザ/ユーザー/作者)
2. 重要度 (`!important`)
3. 詳細度
4. 出現順」

→ 同じセレクタなら「後勝ち」

継承 (Inheritance)

- 多くのタイポ系プロパティ（`font-family`、`color`、`line-height`など）は子に継承
- レイアウト系（`margin`、`padding`、`border`など）は継承しない

詳細度 (Specificity)

- 目安 (大きい方が強い)
 - 行内スタイル (`style=""`) : 1000pt
 - IDセレクタ: 100pt
 - クラス/属性/疑似クラス: 10pt
 - 要素/疑似要素: 1pt
- 例 :

```
/* 1点 */      p { color: #111; }                                     CSS
/* 10点 */     .text { color: #222; }
/* 100点 */    #main { color: #333; }
/* 合算 */     #main .text p { color: #444; } /* 100 + 10 + 1 = 111 */
```

`!important` の扱い

- デバッグ目的以外では極力避ける（局所最適で全体が崩れやすい）
- 必要ならスコープを限定（ユーティリティ/テーマ切替など）

セレクタ速習（よく使う実例）

基本セレクタ

```
/* 要素 */
h1 { margin: 0; }

/* クラス */
.btn { padding: .5rem .75rem; }

/* ID */
#main { max-width: 1200px; }

/* 属性 */
input[type="email"] { border-color: #60a5fa; }
```

結合子

```
/* 子孫(スペース) */
.nav a { text-decoration: none; }

/* 子(>) */
.list > li { padding: .5rem; }

/* 隣接(+) */
h2 + p { margin-top: .25rem; }

/* 兄弟(~) */
.notice ~ p { color: #666; }
```

CSS

属性セレクタ (よく使う記法)

- 存在/等価
 - `[attr]` 属性が存在する要素
 - `[attr="value"]` 完全一致
- 単語/言語
 - `[attr~="word"]` スペース区切りの単語に `word` を含む (例: `class` 相当の単語検索)
 - `[lang|="en"]` `en` または `en-XX` の前方一致 (言語コード向け)
- 前方/後方/部分一致
 - `[attr^="pre"]` 前方一致
 - `[attr$="suf"]` 後方一致
 - `[attr*="mid"]` 部分一致

```
/* 例 */
a[target="_blank"] {
    text-decoration: underline;
}

input[type="email"] {
    border-color: #60a5fa;
}

[data-role~="primary"] {
    font-weight: 700; }

[aria-current="page"] {
    color: hsl(220 90% 56%); }
```

頻出疑似クラス: 状態

- `:hover` ポインタが重なったとき
- `:active` クリック（押下）中
- `:focus` キーボード等でフォーカス
- `:focus-visible` 視覚的フォーカス（必要なときだけフォーカスリング）
- `:disabled` / `:enabled` 無効/有効なフォーム要素
- `:checked` チェック状態（checkbox/radio）

頻出疑似クラス: 構造

- `:first-child` / `:last-child` 親内の最初/最後の子
- `:nth-child(n)` / `:nth-of-type(n)` n番目 (`even` / `odd` も可)
- `:not(sel)` 指定に一致しない要素
- `:is(sel-list)` / `:where(sel-list)` いずれかに一致 (`:where` は詳細度0)
- `:has(sel)`
子孫/近傍の存在で親を選択 (強力だがレガシー環境には非対応あり)

頻出疑似クラス: フォーム・検証

- `:required` / `:optional` 必須/任意
- `:read-only` / `:read-write` 読み取り専用/編集可
- `:valid` / `:invalid` / `:in-range` / `:out-of-range` 入力の妥当性
- `:placeholder-shown` プレースホルダ表示中

頻出疑似クラス: その他

- `:link` / `:visited` 未訪問/訪問済みリンク
- `:target` URLのハッシュ（`#id`）が一致した要素
- `:root` 文書ルート（`html`）
- `:empty` 子要素もテキストも無い要素

注意: 構造擬似クラスを過剰にネストすると再計算コストが増えます。セレクタは短く、役割で対象化しましょう。

実戦のコツ

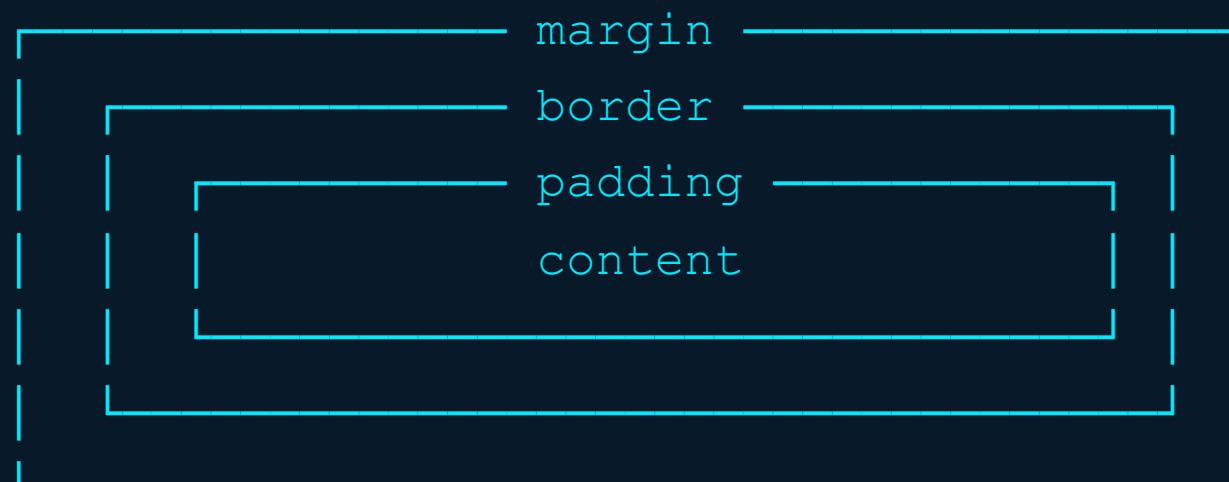
- セレクタは短く・局所的に（詳細度の暴走を避ける）
- BEMなど命名規則で「役割」を表し、階層依存を最小化
- `id` より `class` 中心、`!important` は最後の手段

ボックスモデル/表示/余白

ボックスモデルの基本

- コンテンツ領域 + padding + border + margin

図解（概念）



content-box と border-box の違い

```
/* 幅200pxを指定したときの実効幅 */  
CSS  
.contentBox {  
    box-sizing: content-box; /* 既定 */  
    width: 200px;           /* 実効幅 = 200 + padding + border */  
    padding: 16px;          border: 4px solid #999;  
}  
  
.borderBox {  
    box-sizing: border-box;  
    width: 200px;           /* 実効幅 = 200 に padding/border を内包 */  
    padding: 16px;          border: 4px solid #999;  
}
```

余白の基本：margin と padding

- margin (外側の余白) : 要素の外、隣接要素との間隔を作る。上下方向は状況により「折り畳み (collapse)」が起こる
- padding (内側の余白) : 枠線 (border) と内容 (content) の間隔。折り畳みは起こらない
- box-sizing が border-box の場合、padding / border は指定幅に内包される (レイアウトが読みやすい)

```
/* 方向指定(物理) */
margin-top: 1rem;
padding-left: .75rem;
/* 論理プロパティ(縦書き/言語向け) */
margin-block: 1rem;          /* 上下(書字方向に依存) */
padding-inline: 1rem;         /* 左右(書字方向に依存) */
```

css

表示方式 (display)

- `block` : 縦に積まれる。幅は親いっぱい
 - `<div>` , `<p>` , `<main>` , `<section>` , `<h1-6>` , などなど
- `inline` : 行内に流れる。幅/高さ指定不可
 - ``
- `inline-block` : 行内だが幅/高さが効く
 - `<input>` , `<button>` , `<textarea>` , `<select>`
- `none` : 非表示 (レイアウトから除外)
 - `<script>` , `<style>` , `<template>` , `<meta>` , `<link>`

値と単位/色/タイプグラフィ

単位

- 絶対：`px`（デバイス独立ピクセル）
- 相対：`em`（親のfont-size基準）、`rem`（ルート基準）、`%`
- ビューポート：`vw` / `vh`（幅/高さの1%）

```
html { font-size: 16px; }  
main { padding: 1.5rem; } /* 24px 相当 */
```

css

色表現（基本）

- 16進数: `#rgb` / `#rgba` / `#rrggbb` / `#rrggbbaa` (3/4/6/8桁)
 - 例 : `#FFFFFF`, `#000000`, `#FF0000`など
- `rgb()` / `rgba()`:
 - 従来: `rgb(34, 197, 94)` / `rgba(0,0,0,0.6)`
 - 推奨 (CSS Color 4) : `rgb(34 197 94 / 0.8)`
→ スペース区切り + `/` でアルファ（透過度0.0 ~ 1.0）を指定
- `hsl()` / `hsla()` (色相/彩度/明度):
 - 推奨: `hsl(220 90% 56%)` / `hsl(220 90% 56% / 0.6)`

色表現（発展）

- キーワード（特殊値と名前付き色）

- `currentColor`: 現在の要素の `color` の計算値を参照する特殊値。
- `border-color` / `outline-color` やSVGの `fill` / `stroke` などに使うと、文字色と装飾色を常に同期できる
- `transparent`: 完全透過（アルファ0）。背景が透ける。概念的には `rgba(0 0 0 / 0)` と等価で、上位の背景色が見える
- 名前付き色: 仕様で定義された色名（例: `rebeccapurple` / `tomato` / `gold` など）。
- 手早い指定に便利だが、ブランドカラーやデザイントークン運用では HSL/RGBやカスタムプロパティの使用が推奨

色表現（アルファ値と透過）

- `opacity` は子要素にも影響（要素全体が透過）。部分的な透過は色のアルファを使う
- 背景のオーバーレイや枠線の半透明は `rgba()` / `hsl(... / a)` で表現

タイポグラフィ

- 読みやすさ重視 : `line-height: 1.5~1.8`
- 目安、本文16px以上

タイポグラフィ頻出プロパティ(1/2)

- フォント

- `font-family`: 使用フォントの優先順 (フォールバックを含む)
- `font-size`: 文字サイズ (`rem` / `em` / `px` などの単位)
- `font-weight`: 太さ (`400` / `700` や `normal` / `bold`)
- `font-style`: 斜体など (`normal` / `italic` / `oblique`)
- `font-variant`: スモールキャップ等のバリアント

- 行・文字

- `line-height`: 行間 (単位なし比率や数値/単位で指定)
- `letter-spacing`: 文字間隔
- `word-spacing`: 単語間隔

タイポグラフィ頻出プロパティ(1/2)

- 文書

- `text-align`: 行揃え (`left` / `right` / `center` / `justify`)
- `text-decoration`: 下線/打消し線などの装飾
- `text-transform`: 大文字/小文字/先頭大文字への変換
- `text-indent`: 段落1行目の字下げ

- 折返し/空白

- `white-space`: 空白と改行の扱い (`normal` / `nowrap` / `pre` など)
- `overflow-wrap`: 長い語の任意位置での折返し可否 (`anywhere` 等)
- `word-break`: 単語境界での改行規則 (CJK等の制御)
- `hyphens`: ハイフネーション (自動ハイフン挿入)

ミニ演習（各10分想定）

- 演習1（セレクタ）：[lecture-02/work/starter/01-selectors.html](#)
- 演習2（ボックスモデル）：[lecture-02/work/starter/02-box-model.html](#)

付録：頻出プロパティ

ボックスモデル

- サイズ: `width` / `min-width` / `max-width` / `height` / `min-height` / `max-height`
- 余白系: `margin` / `padding`
- 枠線/角/影: `border` / `border-radius` / `box-shadow`
- はみ出し: `overflow` / `overflow-x` / `overflow-y`
- 枠外装飾: `outline`

余白・間隔

- 方向指定: margin-top/right/bottom/left / padding-*
- 論理プロパティ: margin-block / margin-inline / padding-block / padding-inline
- 要素間ギャップ（コンテナ依存）: gap (Flex/Gridで利用)

表示/可視性/はみ出し

- 表示/可視性: `display` / `visibility` / `opacity`
- はみ出し制御: `overflow` / `text-overflow` / `white-space`
- 行内整列（インライン文脈）: `vertical-align`

色

- 色系: `color` / `background-color` / `border-color` / `outline-color`
- 透過: `opacity` / `background` (グラデーション含む)

タイプグラフィ

- フォント: `font-family` / `font-size` / `font-weight` / `font-style`
- 行/文字: `line-height` / `letter-spacing` / `word-spacing`
- 文章: `text-align` / `text-decoration` / `text-transform`

参考

- MDN: CSS セレクタ
- MDN: ボックスモデル
- MDN: display
- MDN: 値と単位
- MDN: 外部スタイルシート (link)
- MDN: リソースヒント (preload)