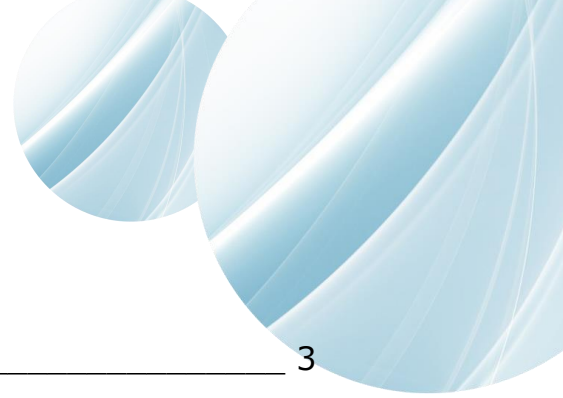




HTML/CSS



## 目次

1. はじめに	3
2. HTML	4
HTML 概要	4
HTML タグ	5
文字コード	9
HTML Living Standard	10
文書型宣言	10
XHTML の記述方法	11
ページタイトル	12
コメント	14
Web ページに接続する仕組み	15
3. HTML タグの種類	17
段落	17
改行	18
見出し	19
箇条書き	22
画像	23
リンク	25
表組み	30
4. CSS とは	34
CSS の基本書式	34
CSS を記述する場所	36
セレクタの指定方法	38
5. プロパティの種類	43
文字装飾	43
背景装飾	47

6. ブロック構造	50
ブロック要素を分割してグルーピング	50
インライン要素を部分的に装飾	50
コンテンツの大きさと余白、枠線を制御する	52
7. ページレイアウト	64
ページレイアウト	64
float プロパティによる段組み	69
float の解除	71
8. フォーム	79
フォーム領域	79
コントロール	80
フォームの作成	82

## 1. はじめに

皆さんは日常的にインターネットを利用し、様々な Web ページを閲覧していることでしょう。本書では Web ページを作成するために必要となる HTML、CSS について学んでいきます。細かい部分を学び始める前に、まず意識しておく方が良いのが、HTML と CSS が持つそれぞれの役割についてです。

例え話をしましょう。建物を想像してみてください。建物を建てるには設計図と骨組みが必要です。設計図をもとに、必要最低限の骨組みがあれば住むことは可能ですよね。しかし、ただの骨組みだけでは、どこがリビングでどこが寝室なのかわかりません。リビングや寝室であることを示す名札や、住み心地を豊かに彩るデザインが必要になるはずです。



これらを Web という媒体に置き換えると以下のように考えることができます。家を建てるための設計図や、「ここからここまで寝室」「ここからここまでリビング」といったラベル付けをする役割を担うのが HTML になり、それらにデザインを加えていくのが CSS の役割になるのです。



**HTML**

設計図、骨組み



**CSS**

デザイン

今後 HTML と CSS を利用して Web ページを構築する際に、ここは骨組みにあたるだろうか？それともデザインにあたるだろうか？と天秤にかけることで整理整頓された構造の Web ページを組み立てることができるようになります。

整頓されたソースは運用面において、更新や修正を正確かつ手早く行うことができるので、ぜひ、意識しながら学習を進めて行きましょう。

## 2. HTML

### HTML 概要

HTML(Hyper Text Markup Language)は、文字や画像、リンクといった様々な情報の表示や配置を決める（コンテンツを記述する）ための設計図や骨格（骨組み）です。



HTML ファイル自体は単純なテキストファイル形式となっており、拡張子は.htmlあるいは.htm となります。これらの拡張子を付けないと、HTML ファイルであることをコンピュータが認識してくれないので注意しましょう。Web ブラウザはこの設計図に基づいて画像や動画などの別のファイルを読み込み、ページを表示しているのです。

HTML はソースコードであるため、普段 Web ブラウザから直接目にすることはありませんが、ブラウザ上で右クリックを押し、「ページのソースを表示」という項目を選択すれば表示することができます。では実際に当社のサイトにアクセスし、ソースを見てみましょう。

ソースコードをチェックしてみよう！  
<https://www.3sss.co.jp/>

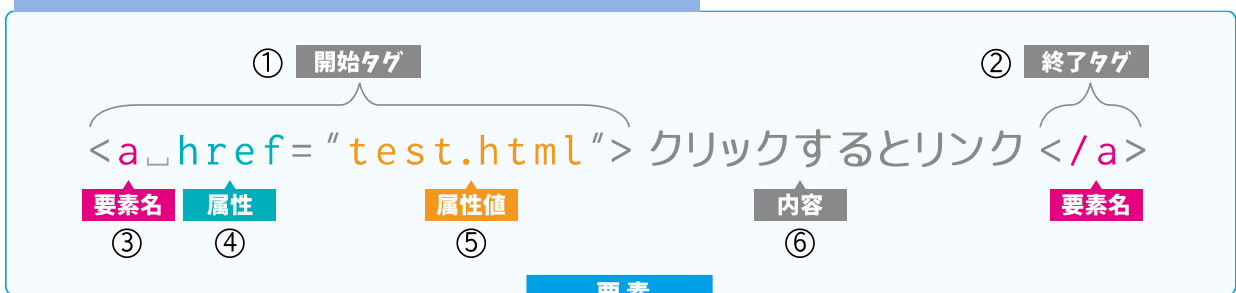


## HTML タグ

HTML には、タグと呼ばれる記号を使ってテキストに特別な意味を持たせる仕組みがあります。タグは開始タグと終了タグの対で構成され、その間に挟んだ内容に対し効果を与えます。また、タグ構成で見出しや段落、表などの役割を持たせた部品のことを「要素」と呼びます。以下の例では、「クリックするとリンク」という文字にリンクを設定する要素ですが、開始タグと終了タグに同じ **<a>** というアルファベットが含まれています。これを要素名と呼びます。

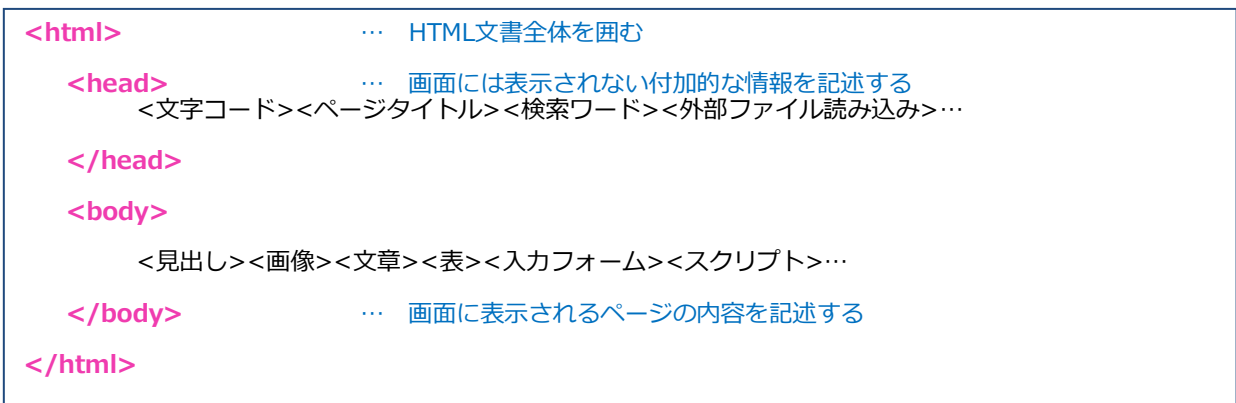
また、開始タグの中に属性を追加し、大きさや配置など様々な調整を行うこともできます。これらをまとめると、以下のようになります。

HTML ソースコード (00\_HTML タグ/index.html)

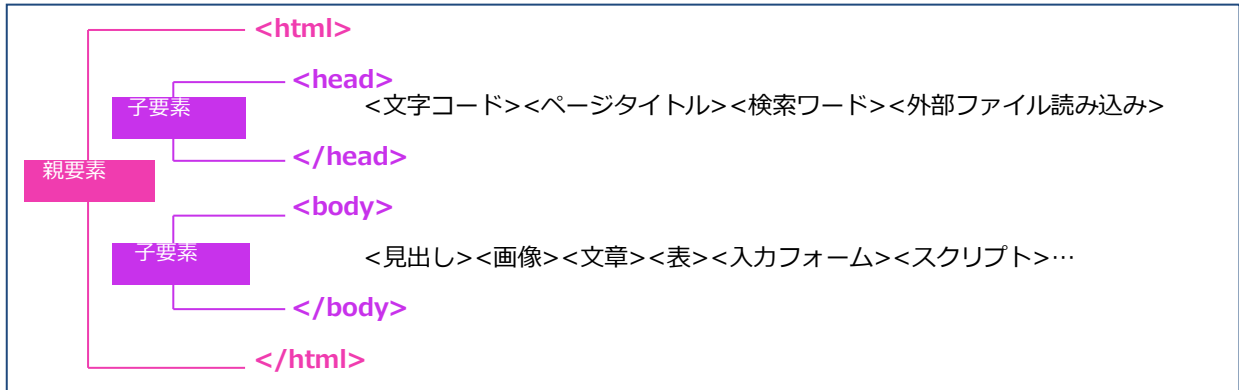


- |              |  |
|--------------|--|
| ① 開始タグ       | 要素の開始位置を表す。                              |
| ② 終了タグ       | / と、開始タグで宣言した要素名を指定し、要素の範囲を閉じる。          |
| ③ 要素名 (タグ)   | 名前に応じ、段落や見出しといったさまざまな役割を果たす。             |
| ④ 属性         | 要素名とセットで扱うことで機能を追加するオプション。記述が必須となるものもある。 |
| ⑤ 属性値        | 属性で扱いたい詳細な数値やアドレス、文字列などの値。               |
| ⑥ 内容 (コンテンツ) | 要素名によって効果を付与される文字列や画像など。                 |

HTML の記述で最も基本となるタグは **<html>** タグ、**<head>** タグ、**<body>** タグです。各タグの持つ意味は、以下の図で示す通りです。



HTML はタグの中に他のタグをさらに記述していくことで 1 画面を構成していきます。このことを入れ子構造（ネスト）と呼び、前述の例をなぞるのならば、**<html>** タグが親要素となっており、**<head>** タグと**<body>** タグが子要素となっています。



## 起こりやすいミス

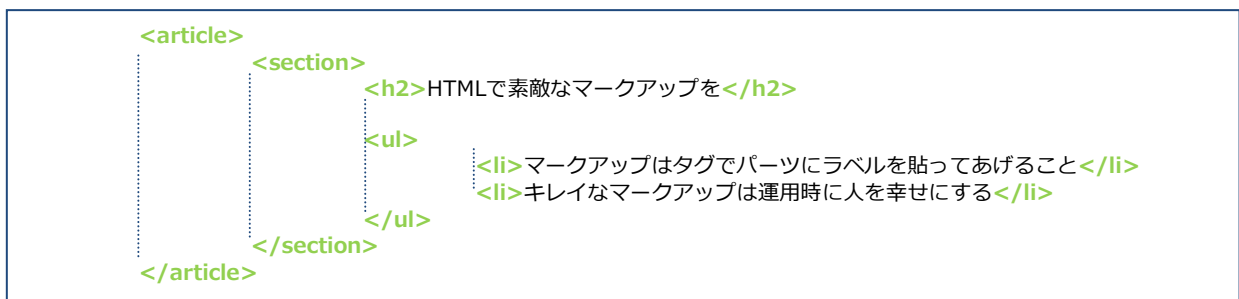
**<p>** 内容 **</p>...** 開始タグが無い。

**<p>** 内容 **</p>...** タグのカッコが抜けてる（<p と画面に表示されてしまい景観が崩れる）

**<p>** 内容 **</p>...** 終了タグが無い（実は終了タグは省略が許可されており、次のタグが開始してしまう。結果的にレイアウト崩れが起きる）

HTML の記述では、この入れ子構造が何層にも重なることが多くなります。可読性の低いソースコードで書かれた Web ページは運用時に、ヒューマンエラーを引き起こしやすくなります。可読性を担保するために、親と子では文頭に半角スペースや Tab でインデントを付けることが一般的です。開始タグと終了タグで頭を揃えて記述するよう心がけましょう。

特に、HTML を記述する際に入れ子の数が増えてくると、タグのカッコが欠ける、終了タグの記述漏れといった問題が起こりやすくなります。タグを記述するときは内容の前にまず開始タグと終了タグをセットで書いてしまいましょう。



ちなみに、<article>タグと<section>タグはセクショニング要素と呼ばれ、範囲を明確にする要素となります。<article>タグは文書の内容が独立しているセクションであることを示す際に使うタグ、<section>タグはページ内の意味や機能などの範囲を示す際に使われるタグとなります。

実際の使用方法などは「7. ページレイアウト」にて紹介されています。

## STUDY!!

**要素名 :**      <html></html>

**要素型 :**      その他

ページ全体を囲んで HTML 文書であることを指定する最上位の要素。

[lang="ja"]属性を追記することで、そのページが日本語文書であることを明記することが可能。これは、Google などの自動翻訳機能に影響を及ぼすので、日本語サイトを作成するのであれば、明記しておくことが推奨される。

**要素名 :**      <head></head>

**要素型 :**      その他

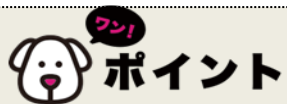
ページタイトルや文字コードの指定、検索ワードの指定や外部ファイルの読み込みなど、ページ上には表示されない重要な情報（メタデータ）を記述する要素。

**要素名 :**      <body></body>

**要素型 :**      その他

Web ブラウザに表示される具体的な内容を記述する、HTML の本体と要素。





## ブラウザの開発者ツールを活用しよう！(Chrome)

『ページのソースを表示』を選択することで、HTML のソースコードを Web 上で閲覧することができることは覚えましたが、今回は、右クリック→検証(もしくはF12 キー)を選択してみましょう。

単純に HTML のソースコードを表示するだけでなく、各入れ子の構造がツリー状で表示され、より視覚的にわかりやすく表示されるようになります。それだけではありません！HTML のソースコードを試しに変更することも、CSS を活用したデザインを変更することも可能ですし、昨今欠かせないスマートフォンで表示する場合どうなるかといったこともチェックすることができます！

細かくはお伝えしませんが、是非開発者ツールを活用して、ミスのない素敵な画面を作っていきますよう！

## 文字コード

文字コードとは、コンピュータ上で文字を利用するために各文字に割り当てられたコードのことです。この文字コードが文字を表示するコンピュータに適していないと、文字を正しく表示することができません。これを「文字化け」と呼びます。HTML で文字コードを指定するためには<meta>タグを使用します。日本語では主にシフト JIS、EUC、UTF8 という 3 種類の文字コードが使われており、それぞれ次のように指定します。

シフト JIS	<code>&lt;meta charset="Shift_JIS" /&gt;</code>
EUC	<code>&lt;meta charset="EUC_JP" /&gt;</code>
UTF8	<code>&lt;meta charset="UTF-8" /&gt;</code>

以下は、文字コードに UTF8 を指定した場合の例です。

```
<head>

<meta charset="UTF-8" />

</head>
```

## 使用する文字コード

本書では、現在一般的に使われている UTF-8 で学習を進めます。しかし、利用するテキストエディタによっては、他の文字コードが使われていることもあるため、事前に調べておきましょう。



お電話でのお問い合わせ  
03-5812-7529

[メニュー](#)
[お問い合わせ](#)

[お問い合わせ](#)
[お問い合わせ](#)

1

2

x

y

plus

3

10

20

x

y

plus

30

文字化けしたWebページ。  
文字が解読不能な状態で表示され、  
ユーザが情報を読み取れない。

## HTML Living Standard

現在の HTML の標準仕様は HTML Living Standard です。以前は、W3C(World Wide Web Consortium)という団体が策定した HTML5 と、WHATWG(Web Hypertext Application Technology Working Group)の HTML Living Standard の 2 つの標準が存在しました。しかし、2018 年に Google の Chrome や Mozilla の Firefox、Apple の Safari などが HTML Living Standard を標準仕様として採用し始め、2019 年に唯一 HTML5 を標準にしていた Microsoft の Edge も、これを標準として採用しました。これをきっかけとして、2021 年に WHATWG の HTML Living Standard が統一された標準仕様として採用されることになりました。

HTML Living Standard ではいくつかの新しい機能の追加や変更がありましたが、HTML5 と比べても基本的な要素や属性の扱い方に関して大きく変わりはありません。実務では過去のバージョン仕様に沿って作成されたウェブサイトに触れる場合も多くあります。本書で紹介する内容は、HTML を使って画面表示をするために覚えておくべき基本的な技術や知識を中心としており、HTML Living Standard の仕様には準拠しない形となっています。

## 文書型宣言

HTML には様々なバージョンが存在します。そのため、どのバージョンの HTML として Web ブラウザに読み込んで欲しいのかを、「文書型宣言 (DOCTYPE 宣言)」で指定します。

前述した HTML Living Standard の仕様に沿ってウェブサイトを構成することにより、検索エンジンがその構造をより詳細に把握することができ、検索結果の順位向上やコンテンツの品質担保などに効果があります。検索運用が古くから続いている Web サイトは XHTML 1.0 や、ごく稀に HTML4.01 が使用されていることもあります。以下は各バージョンの文書型宣言を並べたものですが、HTML Living Standard ではコードのシンプル化が図られていることが見て取れます。なお、HTML5 と HTML Living Standard の宣言の記述は同一です。

### 【HTML4.0】

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
```

### 【XHTML 1.0】

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN""http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

### 【HTML5】

```
<!DOCTYPE html>
```

### 【HTML Living Standard】

```
<!DOCTYPE html>
```

▲HTML4.0 と XHTML1.0 では上記ソースコードのマーカーを付けた部分に注目すれば一目で判別が付きま  
す。すべてを覚えておく必要はありませんが注目する部分だけは知っておきましょう。

文書型宣言は<HTML>タグよりも前に記述します。

```
<!DOCTYPE html>
<html lang="ja">
  ~中略~
</html>
```

※ちなみに<html>の宣言よりも前に書くことから想像できるように、この文書型宣言はタグではなく、SGMLのマーク宣言と呼ばれるものに分類されます。

## XHTMLの記述方法

バージョンの違いの中でも、XHTMLは煩雑になりがちな従来のHTMLの記述方法を整備する目的で以下のように厳密なルールを設けています。XHTMLで運用されているサイトに手を加える機会がある場合には、扱いに注意しましょう。

### XHTMLの代表的なルール

1. 要素は小文字でなければならない  
タグの省略ができない  
`<p>content <p>content2` は禁止。  
`<p>content</p> <p>content2</p>`と記述する。
2. 属性値は必ず引用符で囲う  
`<a href=index.html>`という記述はNG。  
`<a href="index.html">`と記述する。
3. 空要素は終了タグを記載するか、`/>`で終了する。  
`<br>`はNG。  
`<br />`と記述する。
4. 属性の省略記法はサポートされない。  
`<input type="radio" checked />` はNG。  
`<input type="radio" checked="checked" />`と書く。

などなど…

## ページタイトル

本や映画に必ずタイトルがあるように、Web ページにもタイトルを必ずつける必要があります。HTML でタイトルを付けるには<title>タグを使用します。

<title>は<head>内に記述し、</title>で閉じるまでの間に記述された文字列がタイトルとして認識されます。

```
<!DOCTYPE html>
<html lang="ja">
  <head>
    <meta charset="UTF-8" />
    <title>日本語も使えるタイトル</title>
  </head>
  <body>
    ...
```

タイトルに記述された文字列が影響を及ぼすのは以下になります。

- Web ブラウザの上部タイトルバー
- 検索結果のタイトル
- ブックマークのタイトル

検索エンジンのアルゴリズム上、タイトルに記述された文字列は重要度が高いこともあり、忘れずに記述したいタグのひとつと言えます。

なお、タイトルには英語も日本語も利用可能ですが、前述の文字コードを指定する前に日本語を使用すると、タイトル部分のみ文字化けが起きることもあるので、文字コードをセットした後に記述するよう心がけましょう。



ここまでの知識をまとめた Web サイトを作成するための必要最低限な記述を以下にまとめておきます。以下をテンプレートにしながら Web ページ作成の学習を進めると良いでしょう。

## HTML ソースコード (01\_テンプレート/index.html)

```
<!DOCTYPE html>
<html lang="ja">
  <head>
    <meta charset="UTF-8" />
    <title>HTMLテンプレート</title>
  </head>
  <body>
  </body>
</html>
```

## STUDY!!

**要素名 :**      **<meta />**

**要素型 :**      **その他**

**属性 :**        **charset="UTF-8"**

メタ情報を定義するタグ。すでに学んだ文字コードの指定以外にも、**[keyword=""]**や**[description=""]**といった検索ワード対策（通称 SEO）に大きく影響を及ぼす属性も多い。いわばそのページのプロフィール設定欄に使用すると考えておこう。

**要素名 :**      **<title></title>**

**要素型 :**      **その他**

そのページのタイトルを設定する際に使用。ブラウザのタブ、検索時のタイトル、ブックマークに表示される。

## コメント

HTML のソースコード内にメモを残したいときや、一時的に要素を非表示にしたいときなどに便利なのがコメントです。ページ規模が大きくなるにつれて入れ子構造が複雑化しやすい HTML では、コメントをどう使いこなすかが鍵になります。コメントを使った以下のテクニックを参考にコメントの使い方を学習しましょう。

```
<!-- ここにかかれた内容がコメントになる。画面上に表示されません。 -->
```

```
<!--
```

```
コメント内で改行することもできる
```

```
-->
```

### コメントを使ったテクニック

[1] 終了タグの目印に使う。

```
<section id="paper_drip">
```

```
  <h2>ペーパードリップの淹れ方</h2>
```

```
  <p>道具が手に入りやすく、気軽にコーヒーを淹れられるのがペーパードリップ。フィルターが使い捨てなので手入れが簡単なのも魅力のひとつです。</p>
```

```
</section><!-- / paper_drip -->
```

[2] 複雑化しそうな場合はブロックのまとまりとして区切る。

```
<!-- ▼▼▼ここからfrench_press▼▼▼ -->
```

```
<seciton id="french_press" class="col-xs-12">
```

```
  <div class="row">
```

```
    <h2 class="ttl">フレンチプレスの淹れ方</h2>
```

```
    <p class="mt10">フレンチプレスはコーヒー粉を湯に浸す浸漬法の一つで、常に均一な味になりやすいと人気の方法。ポットに豆と湯を入れてフィルターで濾すというシンプルなスタイルです。</p>
```

```
  </div><!-- /row -->
```

```
</section><!-- / #french_press -->
```

```
<!-- ▲▲▲ここまでfrench_press▲▲▲ -->
```

[3] 一時的に非表示にしたいときにコメントで非表示にする。

```
<!--<a href="shop/french_press.html">フレンチプレスの購入はこちら</a>-->
```



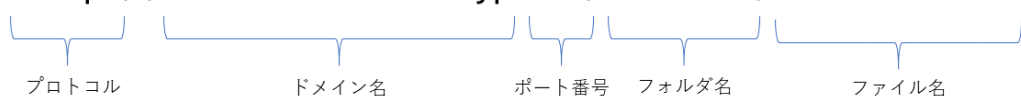
## Web ページに接続する仕組み

ここでは、HTML の書き方についてではなく、HTML で作成した Web ページに接続（アクセス）して閲覧する際の仕組みについて簡単に紹介します。作成した Web ページをインターネット上で公開する際に知っておいた方が良い知識になりますが、まずは用語の名前と役割についてのみ覚えていただければ大丈夫です。

Web ページは、Web ブラウザのアドレスバーに URL を入力することで閲覧できます。URL とは「Uniform Resource Locator」の略であり、インターネット上のデータを特定してアクセスするための表記のことです。

URL は、下記のように決められた形式に沿って書きます。

http://www.3sss.co.jp:80/school/index.html



プロトコル	ドメイン名	ポート番号	フォルダ名	ファイル名
-------	-------	-------	-------	-------

**プロトコル**とは、「通信規約」とも呼ばれ、ネットワーク上でデータを送受信する際のルールのことを指します。Web ページの閲覧に関する通信では、一般的に「http」というプロトコルでやりとりします。

**http** とは、「Hypertext Transfer Protocol」の略であり、HTML を代表とするハイパーテキスト、写真、後述の CSS などのデータを送受信するためのルールのことを指しています。このプロトコルを URL に記述することで、インターネットを通して Web ページが閲覧できるようになります。

**ドメイン名**とは、インターネット上の住所の役割を担う情報です。Web ブラウザから特定の URL にアクセスすると、ドメイン名は DNS（Domain Name System）により IP アドレスと呼ばれる数値形式の住所に変換されます。そして、その住所に該当するサーバー（ネットワークを通してデータを提供するコンピュータ）にアクセスします。

**ポート番号**は、サーバーが提供するサービスの種別を特定するためのあらかじめ決められた番号のことです。

なお、一般的な Web ブラウザからのアクセスは 80 番（http 通信）に該当しますので、多くの場合この番号は省略されます。



ポート番号の代表例を以下に記載します。

サービス	番号	説明
FTP(File Transfer Protocol)	20、21	ファイル転送
SSH(Secure Shell)	22	セキュアなファイル転送
SMTP(Simple Mail Transfer Protocol)	25	電子メールの送信
WWW	80	HTTP プロトコルによる通信
POP2/POP3(Post Office Protocol)	109/110	電子メールの受信
SSL(Secure Sockets Layer)	443	セキュアな HTTP プロトコルによる通信

最後に、**フォルダ名**はアクセスしたい Web ページのファイル（HTML ファイルなど）が保存されているフォルダのことを、**ファイル名**はアクセスしたい Web ページのファイルのことを指します。

なお、自分のパソコン内で作成した HTML の内容を Web ブラウザで直接閲覧する場合は、前述の**プロトコル**、**ドメイン名**、**ポート番号**はアドレスバーに記入不要です。

### 3. HTML タグの種類

HTML には様々なタグが用意されています。その中でも特に重要なのが段落を表す **<p>** タグ、改行を表す **<br />** タグ、見出しを表す **<h1>** ~ **<h6>** タグ、画像を表す **<img />** タグ、リンクを表す **<a>** タグです。これらはすべて body タグの子要素として記述されます。

#### 段落

画面の中で文章のまとまり = 段落 (paragraph) を表したい時は **<p>** タグを使用します。表示上、文字を羅列するだけならば **<p>** タグを利用しなくても良いと考えるかもしれませんが、HTML が持つマークアップ (意味付け) という役割を鑑みると、タグ付けされていない情報があってはいけないと考えてください。

**<p>** タグによって段落が指定された場合、その次に続く記述は改行された状態で始まります。これは **<p>** タグによって作られた段落が画面の横幅いっぱいを専有するからです。このような性質を持つタグはいくつかあり、**ブロック要素**と呼ばれます。

次の例では、2 つの段落がそれぞれ横幅を専有したため、文章が改行されたように見えています。

#### HTML ソースコード (02\_段落/index.html)

```
<body>

  <p>コーヒー豆を栽培する生産地は、世界の様々な場所にある。</p>

  <p>多くは赤道を中心としたコーヒーベルトと呼ばれる熱帯・亜熱帯地域に集中している。</p>

</body>
```

#### 実行画面

コーヒー豆を栽培する生産地は、世界の様々な場所にある。

多くは赤道を中心としたコーヒーベルトと呼ばれる熱帯・亜熱帯地域に集中している。

## 改行

文章中に改行（line break）を入れたい場合は**<br />**タグを使用します。このタグには開始タグと終了タグが存在せず単独で完結するという特徴があります。このようなタグは他にもいくつか存在し、空要素と呼ばれています。**空要素は終了タグの代わりに、「要素名+半角スペース+/(スラッシュ)」という記述ルールが設けられているので覚えておきましょう。**なお、改行タグは文字として扱われ、**インライン要素**と呼ばれます。

**<p>**タグとの使い分けは、あくまで**<p>**は文章のまとまりとして扱い、**<br />**は段落内の文章中で改行を挟みたいときに利用します。

### HTML ソースコード (03\_改行/index.html)

```
<body>
  <p>コーヒー豆の正体は「コーヒーチェリー」と呼ばれる、<br />サクランボに似た赤い実の種子だ。</p>
</body>
```

### 実行画面

コーヒー豆の正体は「コーヒーチェリー」と呼ばれる、  
サクランボに似た赤い実の種子だ。



### <br />タグは余白調整用のタグではない

HTML 入門者に多く見られるのが、段落と段落の間を大きく開けたいという理由で、Word のように改行を連続で記述して調整するといった行為です。

改行タグはあくまで改行という役割を持っており、余白調整用のタグでは無いので、改行タグが2つ以上連続で記述される行為は文法上で禁止されています。

余白を調整する場合はレイアウト調整用のスタイルシートが用意されているので、そちらを利用しましょう。

## 見出し

見出し(heading)となる文章を作りたい場合は、「h」の後ろに見出しレベルを表す 1~6 の数字を添えた<h1>~<h6>タグを使用します。<h1>であれば見出しレベル 1 に相当する大見出し、<h2>であれば中見出し、<h3>であれば小見出しと続きます。

見出しレベルの数字が低いほど重要度はあがります。主に<h1>はページ全体のタイトルに、<h2>は次に大きい見出しに、さらに<h2>に紐づく小さな分類を<h3>で…といった具合に利用していきます。<h4>以降は、禁止されているわけではありませんが、情報が複雑化している可能性が高いので、もし使うことになりそうならば、一度情報整理を見直してみるといいでしょう。

### HTML ソースコード (04\_見出し/index.html)

```
<body>
  <h1>生産地で見えるコーヒー豆ガイド</h1>
  <h2>ブラジル</h2>
  <h3>ブラジル サンタナ</h3>
  <p>甘味とコクが強めで、酸味は控えめ。</p>
  <h2>グアテマラ</h2>
  <h3>グアテマラ ブエナビスタ RA</h3>
  <p>フルーティで芳醇な香りと、しっかりとしたコクが味わえる。</p>
</body>
```

### 実行画面

## 生産地で見えるコーヒー豆ガイド

### ブラジル

#### ブラジル サンタナ

甘味とコクが強めで、酸味は控えめ。

### グアテマラ

#### グアテマラ ブエナビスタ RA

フルーティで芳醇な香りと、しっかりとしたコクが味わえる。

## 見出しレベルの上下関係に注意

見出しのレベルは階層になっており、例えば<h3>の見出しから始まる内容のグループ内に見出しレベルが上位である<h2>を入れるべきではありません。HTML が持つ文書構造をしっかりと意識しながら画面を組み立てましょう。

<!--正しい構造-->

<h1>見出し h1</h1>

<h2>見出し h2</h2>

<h3>見出し h3</h3>

<h3>見出し h3</h3>

<!--良くない構造-->

<h1>見出し h1</h1>

<h3>見出し h3</h3>

<h2>見出し h2</h2>

<h2>見出し h2</h2>

## 文字サイズ調整は CSS で。

なお、見出しタグは初期状態で文字サイズに差がありますが、文字サイズ調整のためのタグではないので、サイズ調整は後に学ぶスタイルシートを利用しましょう。



**STUDY!!**

**要素名：** `<p></p>`

**要素型：** **ブロック要素**

段落を示す際に使用する登場頻度の高いタグ。スタイルシートで余白制御を行わなければ、上下に数ピクセル（Chrome、FireFox では 16px）の余白が生まれる。

**要素名：** `<br />`

**要素型：** **インライン要素**

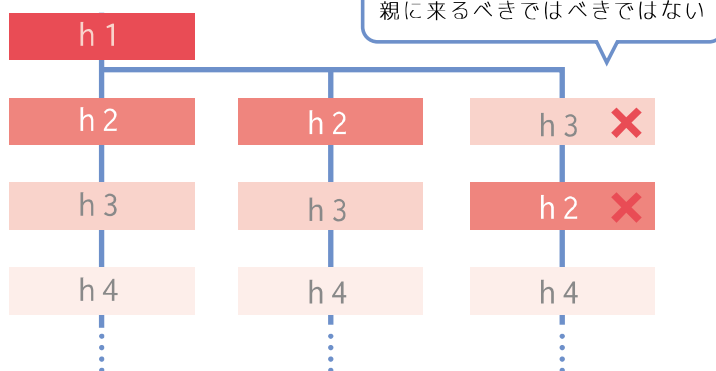
改行を表すタグで、唯一インライン要素で改行させることができるタグ。2 回以上連続で使わない。

**要素名：** `<h1></h1>`

**要素型：** **ブロック要素**

見出しを表す際に使用。`<h1>` から `<h6>` まで存在し、見出しレベルの数字が低いほど重要度が増す。Google 検索エンジンに大きく影響する。また、見出しのレベルは階層になっており、下位レベルの見出しが上位の見出しレベルに設定されるべきではない。

見出しレベルのルール



## 段落、改行、見出しは真っ先にマスターしよう！

body 要素の中に記述する数多くあるタグの中でも、特に`<p><br /><h1>~<h3>`タグは重要です。乱暴な言い方をすれば、この3つがあればある程度の画面は作れてしまいます。（もっとも、これだけだと淡泊なページにしかありませんが…。）真っ先に覚えておくべきタグであることは間違いありません。

## 箇条書き

箇条書きを行いたい場合は<ul>または<ol>タグを使用します。<ul>は「・」による箇条書きを行い、<ol>は番号付きの箇条書きを行います。

<li>タグは箇条書きの各項目を表記するためのタグです。<li>タグを<ul>または <ol>の子要素として記述することで、<li>タグに囲まれたテキストが箇条書きとして表示されます。

### HTML ソースコード (05\_箇条書き/index.html)

```
<body>
  <h1>生産地で見えるコーヒー豆ガイド（・）</h1>
  <ul>
    <li>ブラジル</li>
    <li>グアテマラ</li>
    <li>コロンビア</li>
  </ul>

  <h1>生産地で見えるコーヒー豆ガイド（番号付き）</h1>
  <ol>
    <li>ブラジル</li>
    <li>グアテマラ</li>
    <li>コロンビア</li>
  </ol>
</body>
```

### 実行画面

## 生産地で見えるコーヒー豆ガイド（・）

- ・ブラジル
- ・グアテマラ
- ・コロンビア

## 生産地で見えるコーヒー豆ガイド（番号付き）

- 1.ブラジル
- 2.グアテマラ
- 3.コロンビア

## 画像

画像は画面を華やかに飾るだけでなく、ユーザに対して情報を直感的に伝えやすい手段となりますのでしっかりと利用方法を学んだうえで積極的に利用しましょう。

画面の中で特定の画像 (**image**) を表示させたい場合は、**<img />** タグを使用します。**<img />** タグは単独では機能せず、表示したい画像を示す **src 属性** と、画像の説明を記述するための **alt 属性** が必要になります。ただし、alt 属性は記述しなくても画像表示することは可能です。

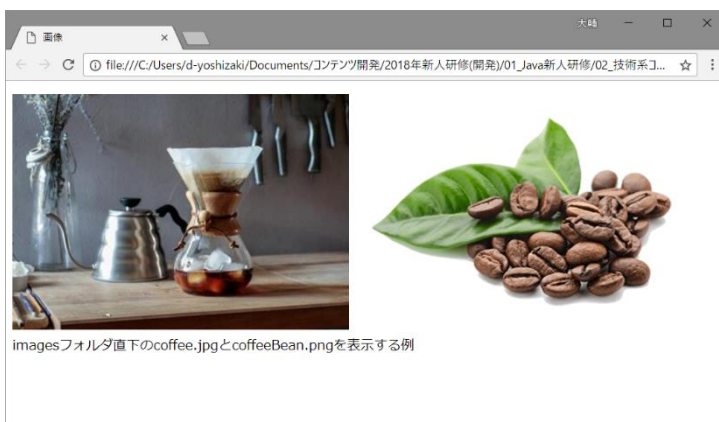
**src 属性** に表示したい画像ファイルの名前をファイルの場所 (パス) と共に記述します。このとき、ファイルパスが誤っていると画像の表示に失敗するのでしっかり確認しましょう。

また、**alt 属性** は画像が読み込めない場合の代替テキストとして表示させたい場合、目が不自由な方に向けた音声読み上げブラウザで画像の説明を読み上げさせる場合、Google の検索エンジンに情報を読み込ませるなどといった様々な役割を持つ大事な属性になります。装飾用のアイコンや矢印、罫線などを画像で表示したい場合は **alt 属性** の内容は空のままにしても構いません。

### HTML ソースコード (06\_画像/index.html)

```
<body>
  <p>
    
    <br />
    imagesフォルダ直下のcoffee.jpgとcoffeeBean.pngを表示する例
  </p>
</body>
```

### 実行画面





## よく使われる画像のファイル形式

形式	特徴
<b>.jpg</b>	jpg（ジエイペグ）は約 1670 万色まで扱える。画像ファイルの圧縮率に優れ、写真のような色数の多い画像を利用するのに向く。なお画像の圧縮率を上げすぎるとモスキートノイズと呼ばれるノイズが画像に入り込み画質が荒れる。不可逆圧縮という圧縮形式を採用しており、一度圧縮された画像はもとの画質を復元できないのでバックアップ推奨。
<b>.gif</b>	gif（ジフ）は最大 256 色までの色を扱え、色数が少ないイラストやアイコンなどと相性が良い。色数を捨てることで画像サイズを圧縮する jpg に対し、gif は色数の並んでいる数を数えて整理する可逆圧縮という圧縮形式を採用しており、水平方向に同色を並べるとファイルサイズが軽くなるという特徴を持つ。また gif の特徴として画像内の 1 色を透明色として扱うことができる点と、パラパラマンガの要領でアニメーション表示をすることもできる。
<b>.png</b>	png（ピング）は jpg と gif の良いところを併せ持つファイル形式で、Web 技術の標準化を推進する団体である W3C が推奨する比較的新しい画像形式でもある。複数の色の透過を行うことができ、jpg と同等のフルカラーにも対応できるが、jpg よりもファイルサイズが大きくなるという欠点を持つ。現在普及しているブラウザでは問題ないが、古いブラウザ（IE6 など）には対応しておらず、透過部分が灰色になる、そもそも表示されないといった問題が起こりうる。
<b>.svg</b>	jpg、gif、png と違い、画像データを点と線の座標で演算し表示する二次元ベクター形式で扱う。数値データをもとに演算された画像なので、拡大をしても画像が劣化しないため、スマートフォンやタブレット、高解像度ディスプレイでの表示に強いという利点を持ち、企業ロゴやアイコンなどに利用する Web サイトが増えている。

拡張子を打ちかえるだけではファイル形式を変えられません。ファイル形式を変換したい場合は GIMP などの拡張子に対応したツールを利用して画像変換を行いましょう。

### 有名な画像加工ツール

（Photoshop は有料です）

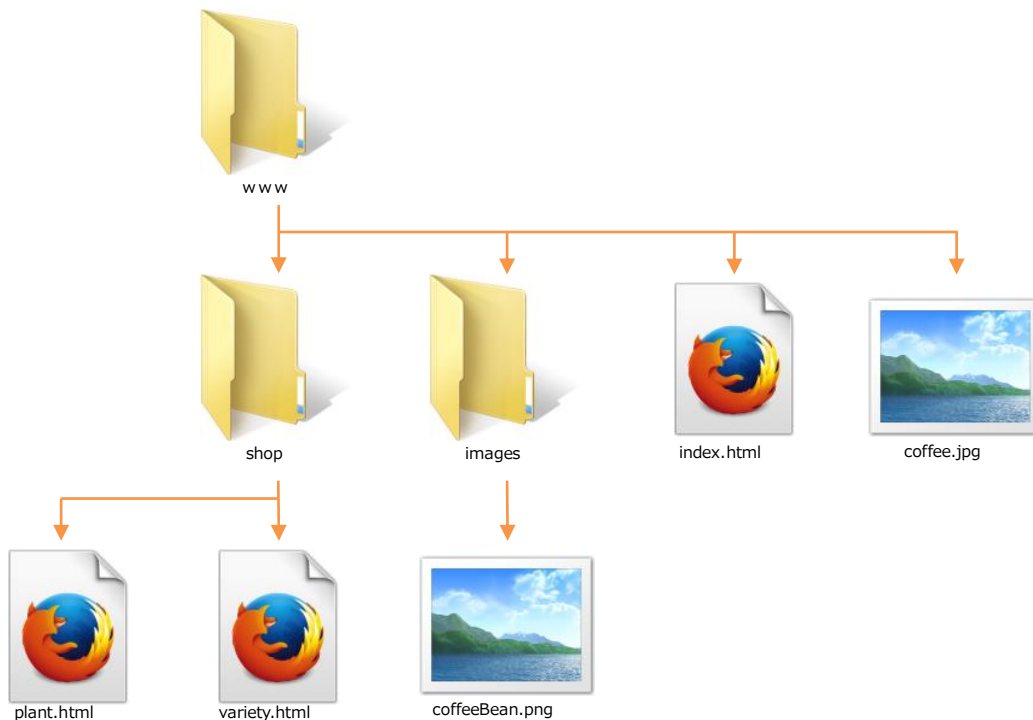


## リンク

リンクは Web という情報媒体が持つ強みのひとつです。百科事典を開いてページをめくるような行為も、Web ページ内に関連リンクさえ設置してしまえば、ワンクリックで欲しい情報を得ることができます。ここでは他のページに対するリンクの設置方法について学びます。リンク設置には船の錨を意味する **a** anchor の頭文字を抜粋した **<a>** タグを使用します。開始タグと終了タグの間に挟まれた情報すべてがリンクの対象になります。文字に対するリンク設置はもちろんのこと、画像を設置すれば画像にリンクを設置することも可能です。

**<a>** タグ単体ではリンクは機能せず、**href 属性**の値に遷移先を指定することで、はじめてリンク機能を持つことができます。また、遷移先の指定には**絶対パス**と**相対パス**という 2 通りの方法があることも覚えておきましょう。

次の例では、index.html 内の「農園から選ぶ」「品種から選ぶ」という文字にリンクを設置しています。クリックすると shop フォルダ内の plant.html、variety.html に遷移することが確認できます。



## HTML ソースコード (07\_リンク/index.html)

```
<h2>こだわりコーヒー</h2>
<p>世界中から取り寄せた選りすぐりのコーヒーをご堪能ください。</p>
<p><a href="shop/plant.html">農園から選ぶ</a></p>
<p><a href="shop/variety.html">品種から選ぶ</a></p>

<h2>もっと詳しく知りたいコーヒー豆</h2>
<p><a href="https://en.wikipedia.org/wiki/Coffee_bean" target="_blank">コーヒー豆を
wikiで調べる</a></p>
```

## 実行画面

## こだわりコーヒー

世界中から取り寄せた選りすぐりのコーヒーをご堪能ください。

[農園から選ぶ。](#)

[品種から選ぶ。](#)

## もっと詳しく知りたいコーヒー豆

[コーヒー豆をwikiで調べる](#)

「農園から選ぶ」「品種から選ぶ」の **href 属性**を見ると index.html から見た相対的なファイルパスを指定しています。この記述方法が相対パスになります。

一方で[コーヒー豆を wiki で調べる](#)というリンクの **href 属性**を見ると「https://」から始まる URL が貼られていることが確認できます。このように URL をそのまま貼り付ける方法が絶対パスです。

また、**href 属性**に加えて **target="\_blank"**という記述にも注目しましょう。**target 属性**はリンクをクリックした際の挙動を制御する属性で、リンクを別のタブで開きたいときに使用します。**target 属性**の値には他にもいくつか存在しますが、現在ではあまり使われていないものがほとんどなので **\_blank** を知っておく程度で問題はありません。

場合にもよりますが、基本的な考え方として、自身のサイト内で遷移する場合は相対パスを、外部サイトへの遷移を促す場合は絶対パスを利用します。

## 相対パスと絶対パス

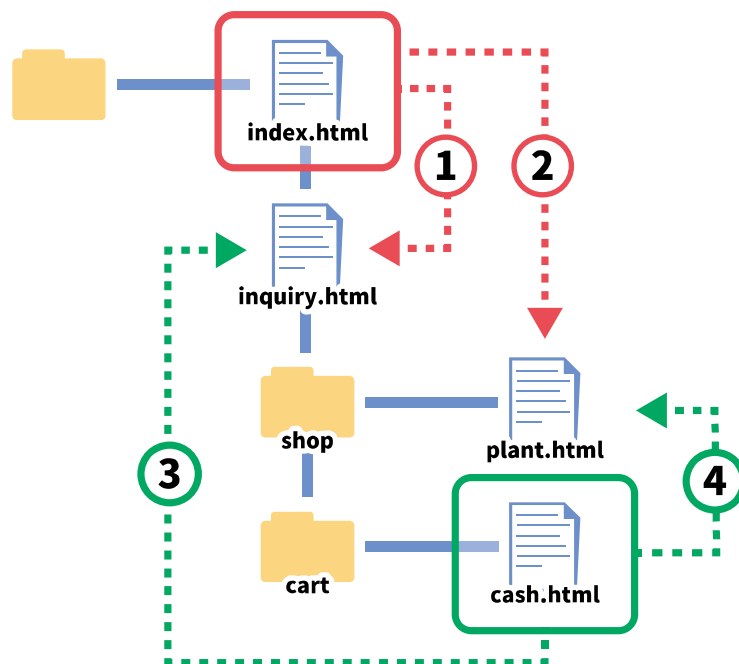
ファイルが存在する場所を指定する際には、パスを指定する必要があります。指定方法には相対パスと絶対パスと呼ばれる2通りの手段があります。

例えるならば、「この郵便物を1つ目の角を右に曲がったビルの6Fに届けてね」と、自分から見た相対的な位置を指定する方法が**相対パス**、郵便物の送付先に住所をすべて書いて指定する方法を**絶対パス**と呼びます。



### 【相対パス】

パスが記述されたファイルから見て、パスで指定した先のファイルがどこにあるのかを記す方法です。指定した先のファイルが上の階層にいる場合は../で一つ上の階層を表し、下の階層にいる場合はそのファイルが含まれるフォルダを記述します。下図で説明します。



- ・①リンク対象が同じ階層だった場合はファイル名

```
<a href="inquiry.html">
```

- ・②リンク対象が一個下の階層だった場合はフォルダとファイル名

```
<a href="shop/plant.html">
```

- ・③リンク対象が一個上の階層だった場合は../とファイル名

```
<a href="../inquiry.html">
```

- ・④リンク対象が一個上のフォルダの下だった場合は../を指定してフォルダとファイル名

```
<a href="../shop/plant.html">
```

## 【絶対パス】

「http://」や「https://」から始まる URL をそのまま貼り付けてファイル位置を記す方法です。相対パスに比べると、パスが記述されたファイルの位置が自由に決められるのが特徴です。外部サイトへのリンク外部のサイトからファイルを読み込む際に使用します。

```
<a href="http://www.yahoo.co.jp/index.html">
```

**STUDY!!**

要素名 : `<img />`

要素型 : インライン要素

属性 : `src="ファイルパス"`  
`alt="画像の説明"`  
`width="横幅"`  
`height="高さ"`

画像を表示するタグで **src 属性** とセットで扱う。**alt 属性** は Google の検索エンジンに画像の情報を与えるほかユーザビリティ向上の観点でも重要な意味を持つ。

**width** と **height** により画像のサイズを指定できるが、比率を変えたり、拡大すると画質が荒れるため、基本的には画像のサイズと同じサイズを指定する。なお、サイズを指定することで、ブラウザが画像をダウンロードしてから画像サイズなどのレイアウトを計算する手間が省けるため若干画像の表示速度が向上するうえ、画像読み込みミスによるレイアウト崩れも防止できる（使用するブラウザによって、表示結果が変わる可能性があります）。

要素名 : `<a></a>`

要素型 : インライン要素

属性 : `href="ファイルパス"`  
`target = "_blank "`

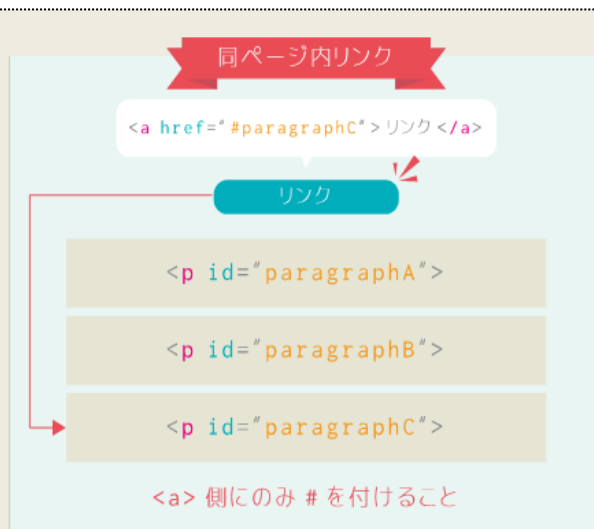
囲まれた要素にリンク機能を付与。**href 属性** の値に遷移させたいファイルパスを指定。パスの指定には絶対パスと相対パスがある。**target 属性** は属性値に **"\_blank"** を指定することでリンク時に別タブでリンク先を表示することができるため、外部サイトへのリンクなど、遷移後も自分自身を表示させておきたいときに有効である。



## 同ページ内リンク

リンクは別ページへのリンクだけではなく、同一ページ内の特定の目印に向かってジャンプさせることもできます。やり方は href の属性値に **"#id 名"** を記述することで同じ名前の id を持つ要素へとページ内リンクさせることもできる。

ちなみに、id 名を省略して # のみを記述するとそのページの最上部へリンクさせることもできる。





## 表組み

表組みを使うと、多くの情報を少ないスペースで効率的に表示することができます。社員の一覧データや商品の一覧をまとめたり、試合の星取表をまとめたりと、用途は多岐に渡ります。

### (1) 表組みのタグ構成

表組み(テーブル)とは、情報を升目状に並べる表示形式です。マス目の横方向を行、縦方向を列、表内のそれぞれの項目をセルと呼びます。HTMLの表組みは、この構造を表現するために4種類のタグを使います。

まず表全体を表すのが<table>タグ、その子が各行を表す<tr>タグで、さらにその子として各セルを表す<td>タグが並びます。また、セルの中でも特に見出しにあたるセルには<th>タグを用います。各セルの内容は<td>タグ、または<th>タグの中に記述します。

<table>	
<tr>	<th> <th>
<tr>	<td> <td>
<tr>	<td> <td>
<tr>	<td> <td>
<tr>	<td colspan="2">

### (2) 表組みで利用するタグ

#### ① <table>

表組み(table)を作成するタグです。<table>タグは、必ず<tr>と<td>(もしくは<th>)とセットで使用されなければなりません。親子関係は<table> → <tr> → <td>( <th>)と決まっており、順番を入れ替えることはできません。

#### ② <tr>

表組みにおいて、横一行(table row)を表すタグです。必ず<table>タグの子要素として記述する必要があります。

#### ③ <td>

表組みにおいて、各セル(table data)を表すタグです。必ず<tr>タグの子要素として記述される必要があります。表組みに行が1つしかない場合でも、<tr>タグを省略して<td>タグのみを記述することはできません。colspan属性やrowspan属性を使うと、複数のセルを結合できます。

#### ④ <th>

見出しセル(table header)であることを意味するタグです。標準では太字かつ中央揃えで表示されます。

## HTML ソースコード(08\_表組み/index.html)

```
<body>

  <table>

    <tr>

      <th colspan="2">日本のコーヒー史</th>

    </tr>

    <tr>

      <th>起きたこと</th>

      <th>年代</th>

    </tr>

    <tr>

      <td>日本初のカフェ開店</td>

      <td>1888 年～</td>

    </tr>

    <tr>

      <td>缶コーヒーの誕生</td>

      <td>1969 年～</td>

    </tr>

    <tr>

      <td>喫茶店ブーム</td>

      <td>1970～1980 年頃</td>

    </tr>

    <tr>

      <td>スターバックス上陸</td>

      <td>1996 年</td>

    </tr>

  </table>
```

## 実行画面



The screenshot shows a web browser window with the title 'table sample'. The address bar shows the file path 'file:///C:/Users/User/Documents/kaneyama/0...'. The rendered table is titled '日本のコーヒー史' and contains the following data:

日本のコーヒー史	
起きたこと	年代
日本初のカフェ開店	1888年～
缶コーヒーの誕生	1969年～
喫茶店ブーム	1970～1980年頃
スターバックス上陸	1996年



**STUDY!!**

要素名 : `<table></table>`

要素型 : **ブロック要素**

表組みを使いたいときに使用。必ず `<table>` → `<tr>` → `<td>`(`<th>`)とセットで使用する。  
初期状態では枠線が表示されないが、枠線表示にはスタイルシートを利用する。

要素名 : `<tr></tr>`

要素型 : **その他**

表組みを表す中で行を示す要素。 `<table>` 直下に必ず必要。

要素名 : `<td></td>`

要素型 : **その他**

属性 : `colspan="左右で結合したいセルの数を指定"`

`rowspan="上下で結合したいセルの数を指定"`

表組みを表す中でセルを示す要素。 `<tr>` 内に記述した数だけセルを描画できる。同一テーブル内のすべての `<tr>` は、内包される `<td>` と `<th>` の合計数が同じでなければならないというルールがある。また、**colspan 属性**と **rowspan 属性**では結合したいセルの数を属性値に記述することで隣り合うセルを結合することができる。

要素名 : `<th></th>`

要素型 : その他

属性 : `colspan="左右で結合したいセルの数を指定"`

`rowspan="上下で結合したいセルの数を指定"`

見出しセルを示す要素で、基本的な仕様は<td>とほとんど同じ。見出しを付けたいときに利用。

`<!--テーブル作成時の悪い例-->`

```
<table>
```

```
  <tr>
```

```
    <th>見出しセル</th>
```

```
    <td>セル 1</td>
```

```
    <td>セル 2</td>
```

```
    <td>セル 3</td>
```

```
  </tr>
```

```
  <tr>
```

```
    <th>見出しセル</th>
```

```
    <td>セル 1</td>
```

```
  </tr><!--最初と 2 番目の <tr> 内のセルの数が合わないので表示がおかしくなる-->
```

```
</table>
```

## 4. CSS とは

CSS (Cascading Style Sheets) は HTML で構築された画面の要素を操作して、文書の体裁や見た目を整えるために用いられる言語です。整理された Web サイトは、閲覧者に対し、情報を的確に伝えます。いかにすぐれた機能を実装していても、ユーザは見た目や使い心地が拙いツールは選びたがりません。すなわち、見た目も機能と言えるでしょう。この章では CSS を用いてさまざまな装飾やレイアウトの技法を学んでいきます。

### CSS の基本書式

CSS はそれ単体では機能せず、HTML とセットで使用します。CSS の基本的な考え方は HTML 要素に対して、「どこを」「何を」「どうするか」です。指定した HTML 要素に対して CSS が持つプロパティを指定して装飾を加えていきます。まずは土台となる書式を覚えましょう。

どこを	何を	どうする
p	{ font-size: 14px; }	
セレクト	プロパティ	値

p 要素の文字の大きさを 14p にする

### ① セレクタ

どこに対してデザインを施すかを指定します。CSS を使いこなすうえでのキモになります。

### ② プロパティ

セレクタで指定した対象に対して何を操作するかを指定します。

### ③ 値

フォントサイズ、文字の色などの各プロパティに対してどうするかを指定します。

```
/*改行を挟んで可読性を上げることもできる*/
```

```
p {  
    color:#cc0000;  
}
```

```
/*複数のプロパティを記述することもできる*/
```

```
p {  
    font-size:16px;  
    color:#cc0000;  
}
```

## CSS を記述する場所

CSS を記述することのできる場所をまとめると全部で以下の 3 通りです。

1. HTML の要素に style 属性を使う
2. <head>内に<style>要素を記述
3. CSS ファイルを作成し HTML から読み込む。

すべて必要な知識ではありますが、HTML と CSS の長所を最大限に活かせるのは 3 番目の CSS ファイルを作成し、HTML で読み込ませる方法になります。ひとつずつ追いかけていきましょう。

### 1. HTML の要素に style 属性を使う（インラインスタイル）

HTML タグに **style 属性**を付与し、直接 CSS を付与します。影響範囲は記述したそのタグのみです。複数のページに同じデザインを施したい場合、コピー＆ペーストを多用する羽目になり、ヒューマンエラーも起きやすくなるため、特別な意図が無い限りは、極力使用を避けましょう。

HTML ソースコード (09\_インラインスタイル/index.html)

```
<p style="color:#cc0000;">コーヒーチェリーのように赤い文字</p>
```

### 2. HTML の style 要素を使う（内部参照）

**<style>** タグを記述しその中に記述します。**<style>** タグは**<head>**タグ内に記述します。記述されたページ全体に影響します。特定のページだけデザインを変更したいときや、単独のページで完結し、作成する CSS の量も多くない場合に利用します。

HTML ソースコード (10\_内部参照/index.html)

```
<head>
  <meta charset="UTF-8" />
  <style>
    p {
      color:#cc0000;
    }
  </style>
</head>
<body>
  ...
```

### 3. CSS ファイルを作成し HTML から読み込む。(外部参照)

HTML を骨組み、CSS をデザインとする考えに基づき、CSS ファイル (.css) を外部ファイルとして扱い、役割を切り分け運用効率を向上させることができます。

**<link>要素**を利用して埋め込むことで、同一の CSS が適用されます。なお、外部参照は複数の属性も指定できますので併せて覚えておきましょう。

#### HTML ソースコード(11\_外部参照/index.html)

```
<head>

  <meta charset="UTF-8" />
  <title>外部 CSS の読み込み</title>
  <link rel="stylesheet" href="style.css" />

</head>
<body>

  <h1>h1 はセレクトに含まれないので深煎りコーヒーのように黒い文字のまま。</h1>
  <p>p は指定されているので文字色がコーヒーチェリーのように赤くなります。</p>

</body>
```

#### CSS ソースコード(11\_外部参照/style.css)

```
@charset "utf-8";

p {
  color:#cc0000;
}
```

▲@charset "utf-8";は CSS での文字コード指定になります。特に意図が無ければ1行目に書いておきましょう。

#### 実行画面



## セレクトアの指定方法

どの HTML 要素に CSS を適用させるかを決めるセレクトアは、CSS を学ぶ上でもっとも注目すべき知識になります。

指定方法の基本は、前節の<p>タグのように HTML 内の要素を指定する方法が基本になりますが、例えば同一画面内に複数存在する<p>タグに対して異なる文字サイズを指定したい場合に、対応が難しくなります。そのような状況に臨機応変に対応できるよう、セレクトアにはいくつかの指定方法が用意されています。

### [1] タイプセレクトア

最も基本となるセレクトアの指定方法です。HTML のタグを直接指定し、同じ種類のタグはその画面の中で同じデザインになります。しかし、適用範囲が広く、使い勝手がいいとは言えません。使いどころは、各ブラウザで設定が異なる HTML の初期設定を均一化させる場合（CSS リセット）、文字のリンク色のように全ページ必ず共通でなければいけない部分がある場合に利用します。

CSS ソースコード (12\_タイプセレクトア/style.css)

```
a {  
  color:#3388ee;  
}
```



### 有名な CSS リセット

CSS リセットは、ブラウザごとに異なる HTML の余白設定や文字サイズなどといった設定（デフォルトプロパティ）をいったんリセットすることで、意図しないレイアウト崩れを防止するのに効果的です。以下のものはよく使われている CSS リセットになります。

#### bootstrap.css

本来はリセット目的ではなく、ノンデザイナー向けのデザインキットという設計思想から生まれた CSS ライブラリではあるが、normalize.css を使用しているのでこれ一本で何重にもお得。レスポンシブ Web デザインに強く、エンジニアに人気。迷ったらこれで OK。

#### normalize.css

上記の bootstrap を利用せず単独で利用したい場合に。余計なリセットは行わず最低限の有用な要素のみリセットしている。

#### YUIreset.css

Yahoo が提供するリセット CSS。シンプルで癖が無い。CDN での利用も可能。

## [2] class セレクタ

同じデザインを複数の要素で使いまわしたいときや、特定の要素を指定してデザインを当てたいときに便利な指定方法で、もっとも利用頻度が高い記述方法です。HTML のタグの中に class 属性を記述し、その値を目印として CSS 側からデザインを指定します。CSS 内では、[. (ピリオド) **class 名**] をセレクタとして使います。HTML 側に記述する際にピリオドを付けてしまうと CSS の指定に失敗するので、ピリオドを記述するのは CSS ファイルのみと覚えておきましょう。

class 属性と id 属性は、各要素に任意の名前をつけるための属性です。名前をつけることで、CSS 側で各要素を区別できるようになります。

記述例 : <○○ class="f\_small、id="wrapper">

HTML ソースコード (13\_class セレクタ/index.html)

```
<p class="f_small">小さい文字</p>
<p class="f_large">大きい文字</p>
```

CSS ソースコード (13\_class セレクタ/style.css)

```
.f_small {
  font-size:10px;
}
.f_large {
  font-size:18px;
}
```

### class セレクタの注意点

[**要素.class 名**] で記述したときと [**.class 名**] で記述した時に違いが生まれます。

```
p.f_small { font-size:10px;} /*p要素限定*/
.f_large { font-size:18px;} /*すべての要素が対象*/
```

▲/\*\*/は CSS でのコメントになります。よく、HTML のコメントと混同しやすいので気を付けましょう。

上記のように記述した場合、[**p.f\_small**] は、厳密にいうとタイプセレクタ + class セレクタの合わせ技になり、<p>タグに f\_small クラスが付与されている場合のみが条件に当てはまります。したがって <h2> など <p> 以外の要素に [**f\_small**] のデザインが適用されなくなります。

一方で、タイプセレクタを指定しない [**f\_large**] は、どんな要素であろうが [**class="f\_large"**] の記述さえあればデザインが適用されます。class の使い勝手が悪くなる原因にもなるので、意図が無い限りはタイプセレクタは省略し、class セレクタのみで記述するようにしましょう。



## class の値は複数記述できる

class セレクタのもうひとつの特徴として、ひとつの class 属性の属性値に複数の class を組み合わせることができるという点があります。よく使いそうな class をあらかじめ用意しておき、必要に応じて class を組み合わせることで余計な記述を省くことも可能です。複数の class を繋げる場合は、半角スペースを入れて繋げます。

### HTML ソースコード (13\_class セレクタ/index.html)

```
<p class="b">太字</p>
<p class="b fs24">太字で24pxのサイズ</p>
<p class="b fs24 fc_red">太字で24pxのサイズで赤い</p>
```

### CSS ソースコード (13\_class セレクタ/style.css)

```
/*文字色を赤*/
.fc_red{ color: #cc0000;}

/*文字サイズを 24px*/
.fs24{ font-size: 24px;}

/*文字太字*/
.b{ font-weight: bold;}
```

### 実行画面



## [3] ID セレクタ

基本的にはクラスセレクタと同じ働きをしますが、クラスセレクタと違い、**同じ ID セレクタは 1 つの HTML ファイル内で 1 つまでしか指定できない**という機能制限があります。主な使いどころはページ内で 2 つ以上登場しない大きなブロック(ヘッダー、フッター、メインコンテンツなど)の内容をまとめたいときに利用します(詳細は後述を参照)。JavaScript と組み合わせて使うと便利です。CSS 内では、**[#ID 名]**をセレクタとして使います。

### HTML ソースコード (14\_id セレクタ/index.html)

```
<div id="wrapper">
  <div>
    <h2>カフェオレ</h2>
    <p>通常のブラックコーヒーに同量のミルクを加えたフレンチスタイルの飲み方。</p>
  </div>

  <div>
    <h2>カフェラテ</h2>
    <p>イタリア発祥のスタイルで、エスプレッソにフォームドミルクを加える。</p>
  </div>
</div>
```

### CSS ソースコード (14\_id セレクタ/style.css)

```
#wrapper {
  width: 600px;          /*ブロック要素の横幅を600pxに*/
  background-color:#cef; /*背景色を"#cceeFF"色に*/
  padding: 10px;         /*内側に10pxの余白を設ける*/
}
```

### 実行画面



## [4] その他のセレクトタ

まずは class セレクトタを中心に基本となる上記 3 つの記述方法をマスターしましょう。その他の応用知識として、以下にソースコードがスマートになるテクニックをいくつか紹介します。CSS に慣れてきたら調べてみるのも良いでしょう。

名前	書式	例
ユニバーサルセレクトタ	*{ }	*{margin:0; padding:0;}
複数のセレクトタ	セレクトタ,セレクトタ{ }	h1,h2{color:#888;}
子孫セレクトタ	セレクトタ セレクトタ	div#container h2{margin-top:10px}
子セレクトタ	セレクトタ>セレクトタ	p>span{color:#c00;}
隣接セレクトタ	セレクトタ+セレクトタ	p+p{ margin-top:10px }
属性セレクトタ	要素名 もしくは 要素名[属性名="属性値"]	input[type="text"]{padding:1em;}



### CSS の競合と詳細度

もし同じセレクトタに対して別々の CSS プロパティが重なった場合、CSS の競合が起きます。例えば太字指定+文字色指定といった具合にプロパティが被らなかった場合はお互いのプロパティが適用され、太字+文字色といったデザインが反映されます。

しかし、プロパティが競合した場合はあとで書いた方の CSS が先に読まれた CSS を上書きします。例えば、p 要素に赤文字を指定した後に青文字を指定した場合は実際に表示される時は青文字で表示されます。

基本的には後で書いた方が反映されやすいということを覚えておきましょう。実際には各セレクトタが持つ詳細度というポイントがあり、競合時にポイント計算され高い方が優先されるというルールがあります。慣れないうちはいったん忘れてもいい知識ですが、知っておくと上手くデザインが反映されないときのヒントになるでしょう。以下は詳細数値が低い順に並べたものになります。

#### 詳細度（低い順）

ユニバーサルセレクトタ(0) < タイプセレクトタ(1) < クラスセレクトタ(10) < 属性セレクトタ(10)  
< 疑似クラス(10) < ID セレクトタ(100) < インラインスタイル(1000) < !important (10000:例外)

## 5. プロパティの種類

スタイルシートには様々なプロパティがあります。ここでは、代表的なプロパティとして知られる文字装飾と背景装飾のプロパティについて記します。

### 文字装飾

#### 文字の色の指定(color プロパティ)

HTML+CSS では、色の指定は次のような 6 ケタの 16 進数で表現されます。この 6 ケタをカラーコード、または“Hex (hexadecimal) 値”と呼びます。

**#RRGGBB**

R は赤、G は緑、B は青を表し、光の 3 原色となっています。この光の 3 原色を様々な組み合わせで混ぜることで、1677 万色に及ぶカラーバリエーションを生み出します。

数値には 16 進数が用いられ 0~9~a~f の順番で数値が上がります。数値は各光源の出力度合いを示しており、数値が低いほど光の出力も低くなり色合いは暗くなり、数値が高くなるほど色合いが明るくなります。各色が最大まで出力されると白（#ffffff）になります。例えば次の色指定はピンクを表しています。

**# f000c8**

ただし、実際には 1677 万色のパターンを記憶する事は不可能です。カラーコードを紹介しているサイトがありますので、必要な色は Web で検索して調べましょう。なお、CSS ではカラーコードを 3 桁に省略でき、省略された場合 RGB それぞれに振られた数値と同じ数値が省略されたと解釈されます。例えば #641 という記述をしたのであれば、暗黙的に #664411 と同じ色として解釈されます。スタイルシートには文字の色を指定するプロパティ **color** が用意されています。

HTML ソースコード (15\_文字色/index.html)

```
<h3>深煎りのイタリアンローストはエスプレッソに最適</h3>
```

CSS ソースコード (15\_文字色/style.css)

```
h3 {  
  color:#641;  
}
```

実行画面

深煎りのイタリアンローストはエスプレッソに最適

## 文字の太さの指定（font-weight プロパティ）

文字を太文字にするには **font-weight** プロパティに **bold** を指定します。特定の文字を強調したいときに利用しますが、乱用しすぎると、画面にまとまりが無くなりますので利用時には注意が必要です。

また、**<h1>**や**<th>**など一部の要素はデフォルトプロパティで太字が設定されていますが、これらの要素に対して太字を適用したくない場合には、**font-weight** プロパティに **normal** を指定することで、太字設定を打ち消し、通常の太さを反映させることができます。

### HTML ソースコード（16\_文字幅/index.html）

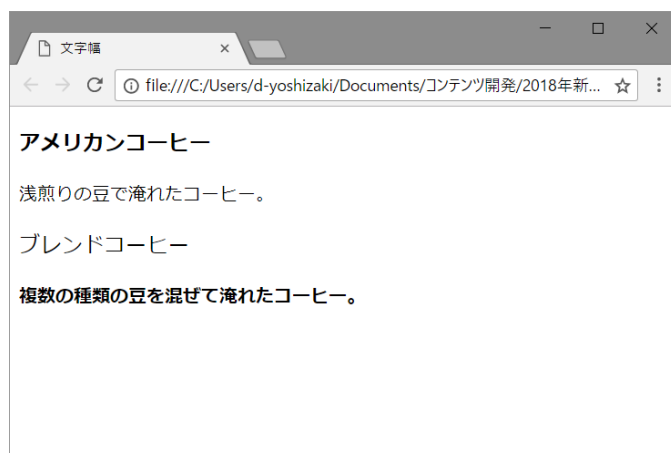
```
<h3>アメリカンコーヒー</h3>
<p>浅煎りの豆で淹れたコーヒー。</p>

<h3 class="fwn">ブレンドコーヒー</h3>
<p class="fwb">複数の種類の豆を混ぜて淹れたコーヒー。</p>
```

### CSS ソースコード（16\_文字幅/style.css）

```
.fwn {
    font-weight:normal;
}
.fwb {
    font-weight:bold;
}
```

### 実行画面



## 文字列に下線を引く(text-decoration プロパティ)

文章に線を引く場合は **text-decoration** プロパティを使い、値には **underline** を使用します。リンクを示す **<a>** 要素には初期値で underline が適用されていますが、デザインの都合上、下線を表示したくない場合には **<a>** 要素に対して **text-decoration:none** を付与します。ただし、やみくもに下線を非表示にするとテキストリンクだと判断しづらくなり、ユーザビリティの低下を招くので、リンクはしっかりリンクであるように工夫する必要があります。

### HTML ソースコード (17\_文字装飾/index.html)

```
<p class="tdu">下線はリンクと勘違いされやすい乱用禁止。</p>
<a href="#" class="tdn">逆にリンクの下線を非表示にするとただのテキストに見える。</a>
```

### CSS ソースコード (17\_文字装飾/style.css)

```
.tdu { text-decoration:underline;}
.tdn { text-decoration:none;}
```

### 実行画面

下線はリンクと勘違いされやすい乱用禁止。

逆にリンクの下線を非表示にするとただのテキストに見える。



## 疑似セレクトタ :hover

「マウスカーソルが重なったとき」という条件のときのみ CSS を適用させる :hover は、リンクを示す a 要素と相性が良い。ユーザは画面上で動くものや変化するものに注目するからだ。text-decoration と :hover をうまく組み合わせれば、よりリンクであることをユーザに対して直感的に伝えることもできる。

```
a{ text-decoration:none; color:#08c;} /*通常時*/
a:hover{text-decoration:underline; font-weight:bold;} /*a 要素にカーソルが重なる時*/
```

## 文字の表示位置を指定する。(text-align)

文章を記述するうえで、左詰めや中央揃えなど、レイアウトの調整をしたくなることもあるでしょう。そんなとき便利なのが **text-align** プロパティです。**text-align** プロパティでは値にそれぞれ **left** (左揃え)、**center** (中央揃え)、**right** (右揃え) を指定できます。**text-align** が制御するのはインラインコンテンツ (文字列や画像) のみでブロック要素自体の位置は調整されません。

また、Web 上では基本的には左揃えで文頭を揃えた方が可読性を確保できるため、やみくもに **text-align** プロパティを触る必要はありませんが、画像や見出しのデザインといった部分的なアクセントで **text-align: center;** を利用すると効果的にメリハリを付けることもできます。

### HTML ソースコード(18\_行揃え/index.html)

```
<h2 class="tal">コーヒーの格言</h2>
<p class="tac">「一杯のコーヒーはインスピレーションを与え、一杯のブランデーは苦悩を取り除く」
</p>
<p class="tar">ルートヴィヒ・ヴァン・ベートーヴェン</p>
```

### CSS ソースコード(18\_行揃え/style.css)

```
.tal{text-align: left;} /*左揃え*/
.tac {text-align: center;} /*中央揃え*/
.tar {text-align: right;} /*右揃え*/

/*ブロック要素の影響範囲を知るために背景色を付ける*/
p,h2{ background: #eca;}
```

### 実行画面





## 背景装飾

CSS では、要素の背景に装飾をほどこすための、**background** プロパティが用意されており、背景色や位置などの一括指定ができます。色の指定方法のみであれば、**background-color** プロパティを利用し、**color** プロパティと同様に 16 進数の値を指定することで設定できます。

### HTML ソースコード(19\_背景色/index.html)

```
<h2 class="light_roast">浅煎り</h2>
<p>酸味が強くなりがち。豆のテイスティングで使われる。</p>

<h2 class="medium_roast">中煎り</h2>
<p>酸味と苦みのバランスが取れた焙煎度。喫茶店のコーヒーで使われる。</p>

<h2 class="dark_roast">深煎り</h2>
<p>苦みが強くなる。エスプレッソやアイスコーヒーに最適。</p>
```

### CSS ソースコード(19\_背景色/style.css)

```
@charset "utf-8";

.light_roast, .medium_roast, .dark_roast {
    color: #fff;
}

.light_roast {
    background: #b96;
}

.medium_roast {
    background: no-repeat center url("coffee.png") #963;
}

.dark_roast {
    background-color: #630;
}
```

## 実行画面



## background プロパティのバリエーション

**background** プロパティには、画像を背景に表示する **background-image** や、背景画像の繰り返しルールを定める **background-repeat** など、たくさんのプロパティが用意されております。

プロパティ	機能	記入例
<b>background-color</b>	背景色を決める	<code>background-color: #fc0;</code>
<b>background-image</b>	背景に画像を配置する	<code>background-image: url("coffee.png");</code>
<b>background-repeat</b>	背景画像の繰り返しルールを決める	<code>background-repeat: repeat-x;</code>
<b>background-attachment</b>	背景画像の画面スクロールを決める	<code>background-attachment: fixed;</code>
<b>background-position</b>	背景画像の位置を決める	<code>background-position: top center;</code>
<b>background-size</b>	背景画像の表示サイズを決める	<code>background-size: cover;</code>

## ショートハンドプロパティ

**background** プロパティは **background** 以降の記述を省略することが可能です。このように特定のプロパティを省略した記法をショートハンドと呼びます。ショートハンド記入時は記述する順番にルールが存在する場合もあるため注意が必要ですが、慣れてしまえば記述量を減らすことができ、ソースコードの可読性担保/軽量化が期待できます。

```
/*ショートハンド未使用（*例は極端な例です。すべて指定する必要はありません。）*/
```

```
.shorthand_example{  
    background-color: #ccc;  
    background-image: url("coffee.png");  
    background-repeat: no-repeat;  
    background-attachment: fixed;  
    background-position: right 100px bottom 50px;  
    background-size: 100%;  
}
```

```
/*ショートハンド利用時（background-sizeを指定する場合には、ポジションを指定した後に / を前につけて記述しなければいけません。）*/
```

```
.shorthand_example{  
    background:#ccc url("coffee.png") no-repeat fixed right bottom / 30%;  
}
```

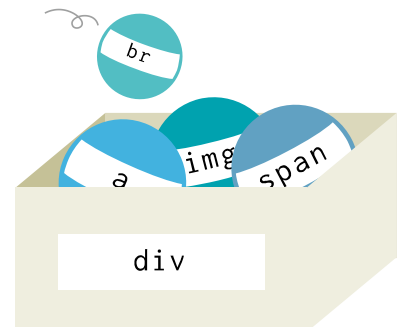
## 6. ブロック構造

HTML のタグは大きく分類すると**ブロック要素**と**インライン要素**の2種類に分けられます。ブロック要素には、文書の構造を決めるタグが分類され、インライン要素には、テキスト、画像関連、リンクのタグが分類されます。ブロック要素は画面全体のレイアウトを決定する骨組みとして使われ、インライン要素は、その各ブロック要素の中に納まるコンテンツとして機能しています。ここでは、代表的なブロック要素とインライン要素のタグを学んでいきます。

ブロック要素	インライン要素
<code>&lt;p&gt; &lt;h1&gt; ~ &lt;h6&gt; &lt;table&gt; &lt;form&gt; &lt;div&gt; &lt;hr&gt; &lt;ul&gt;...</code>	<code>&lt;br&gt; &lt;img&gt; &lt;a&gt; &lt;span&gt; &lt;input&gt; &lt;label&gt; &lt;select&gt;...</code>

### ブロック要素を分割してグルーピング

ページ作成を行う場合、いくつかのグループを分割（**division**）してレイアウト調整を行います。その際に必ず登場するのが**`<div>`**タグです。**`<p>`**タグには段落、**`<h1>`**～**`<h6>`**タグには見出しというマークアップ上の役割がありましたが、**`<div>`**タグにはそれ単体に特別な意味はありません。主に `id` や `class` を付与し、レイアウト調整を目的に使用します。なお、ブロック要素内にブロック要素を包んではいけないというルールがありますが、**`<div>`**要素は例外的に許容されていることから非常に登場数が多いタグとなります。



### インライン要素を部分的に装飾

さて、登場機会が多いと紹介した**`<div>`**でしたが、インライン要素にもまた、特定の意味を持たず、特定の範囲（**span**）を囲むだけの**`<span>`**要素というものが用意されています。この**`<span>`**要素は**`<div>`**同様にマークアップ上の意味は持ちませんが、文字の見た目を装飾したいときにとっても便利で、**`<div>`**要素と並んで CSS と組み合わせて本領を発揮するオールラウンダーな要素と言えます。

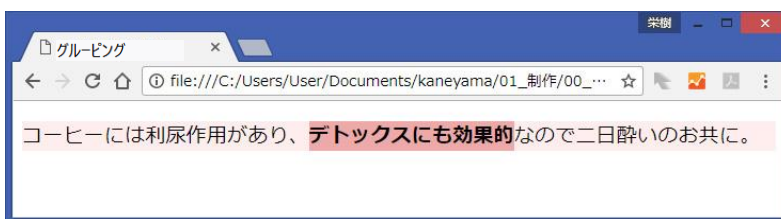
HTML ソースコード(20\_グルーピング/index.html)

```
<div class="wrapper">  
  <p>コーヒーには利尿作用があり、<span class="marker">デトックスにも効果的</span>なので二日酔いのお共に。</p>  
</div><!-- wrapper -->
```

## CSS ソースコード(20\_グルーピング/style.css)

```
.wrapper{  
    font-size: 120%;  
    background: #fee;  
}  
  
.marker{  
    background: #eaa;  
    font-weight: bold;  
}
```

## 実行画面



## STUDY!!

要素名 : **<div></div>**

要素型 : **ブロック要素**

特にマークアップ上での意味は持たない万能要素。レイアウトを切るときの必須要素とも言えるくらい使用頻度が高い。主に id 属性と class 属性を組み合わせながら CSS と連携する目的で利用される。必然的に</div>が連続しやすいので、複雑になる場合はコメントを上手く活用することで可読性を担保できる。

要素名 : **<span></span>**

要素型 : **インライン要素**

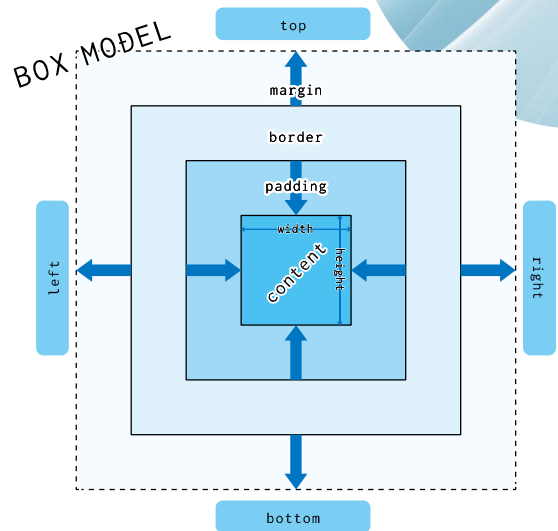
div 要素のインライン版とも言える万能要素。インライン要素の装飾に利用される。日付、入力フォーム、画像のアクセント、疑似セレクトなどの連携などアイデア次第で活躍の場が増える。

## コンテンツの大きさと余白、枠線を制御する

HTML のあらゆる要素は、情報を表示するためのコンテンツ（内容）と呼ばれるエリアを中心に、内側の余白・枠線・外側の余白を指定する CSS のプロパティを持ちます。

この構造を**ボックスモデル**と呼び、CSS でレイアウトを行う際に非常に重要な概念となっています。

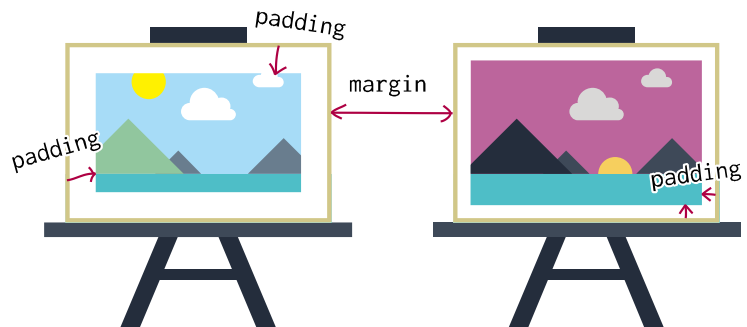
ここでは、**ボックスモデル**を構成するプロパティである、**padding**・**border**・**margin** を学びます。



## ブロック要素の余白(padding プロパティ / margin プロパティ)

ブロック要素のふちを境に、ブロック要素の内側に余白を取りたい場合は、**padding** プロパティを使用します。対して、外側に余白を取りたい場合は **margin** プロパティを利用します。

主な用途は、**padding** は子要素に対して余白調整をしたい場合、**margin** は各要素同士の距離感を調整したい場合と考えて使い分けましょう。



### HTML ソースコード(21\_余白/index.html)

```
<div class="box">
  <h2>ウォッシュド</h2>
  <p>果肉とミューシレージを除去。水洗いして薄皮の状態で乾燥させ生豆にする。</p>
</div>
<div class="box ">
  <h2>ナチュラル</h2>
  <p>収穫したコーヒーチェリーをそのまま乾燥した後、種子以外をすべて除去して生豆にする。</p>
</div>
```

## CSS ソースコード(21\_余白/style.css)

```
.box {  
    margin: 10px;  
}  
  
h2 {  
    background: #963;  
    color: #fff;  
    margin: 0px;  
    padding: 10px;  
}  
  
p {  
    background: #cca;  
    margin: 0px;  
    padding: 10px;  
}
```

## 実行画面



## padding と margin の適用方向を指定する

**padding** と **margin** (あと、後に登場する **border**) プロパティはそれぞれ上右下左方向に適用範囲を指定することができます。適用方法はそれ以下のように、それぞれのプロパティの後ろに **-top** (上)、**-right** (右)、**-bottom** (下)、**-left** (左) と修飾子を付与してあげればよいというルールになります。修飾子を省略した場合は全方向に適用されます。



全方向	margin	padding	border
上方向	margin-top	padding-top	border-top
右方向	margin-right	padding-right	border-right
下方向	margin-bottom	padding-bottom	border-bottom
左方向	margin-left	padding-left	border-left

## padding と margin の便利なショートハンド

**background** プロパティで登場したショートハンドというテクニックですが、**padding** や **margin**、**border** でも利用することができます。やり方はカンタンで修飾子を使わずに指定したい数値をルールに則った順番で記述するだけです。覚えておけばソースコードが大幅に短くなるので、使いこなせると心強いテクニックです。

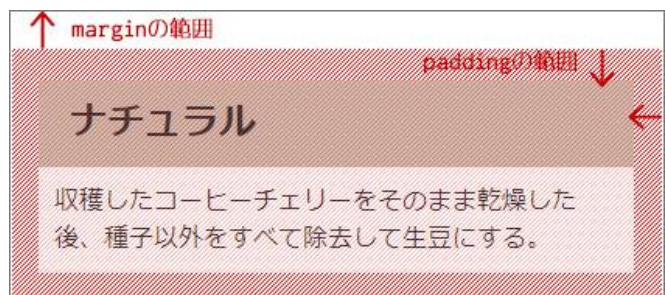
1 ケタ指定	全方向	margin: 10px;
2 ケタ指定	上下 左右	margin: 10px 20px;
3 ケタ指定	上 左右 下	margin: 10px 20px 0px;
4 ケタ指定	上 右 下 左	margin: 10px 0px 20px 10px ;

例は **margin** ですが、**padding** も **border** も同じルールです。4 ケタ指定は一見複雑に見えますが、**上から時計回り**と順番を覚えれば理解しやすいでしょう。

## padding と margin の違いを知る

余白を内側取るか外側取るかの違い以外にも、両者には挙動の違いがあります。大きな違いとして挙げられるのは、**padding** は **background** プロパティの影響を受ける点が挙げられます。

**background** で背景色や背景画像を指定した場合、要素の内側を制御する **padding** の範囲にも背景色が適用されますが、要素の外側を制御する **margin** には背景色は適用されません。このことから、**margin** は要素と要素の距離を制御する役割に適していると言えます。なお、背景色の範囲に関しては、厳密にいうと枠線を制御する **border** プロパティまで影響します。



## padding の使い方

**padding** は背景色を使ったり、枠線を利用した際に、併せて指定することで、文章の可読性を上げることができます。ほんの少し余白を加えるだけで見違えるので、意識的に **padding** を利用していきましょう。

### paddingなし

#### ウォッシュド

果肉とミューシレージを除去。水洗いして薄皮の状態乾燥させ生豆にする。

### paddingあり

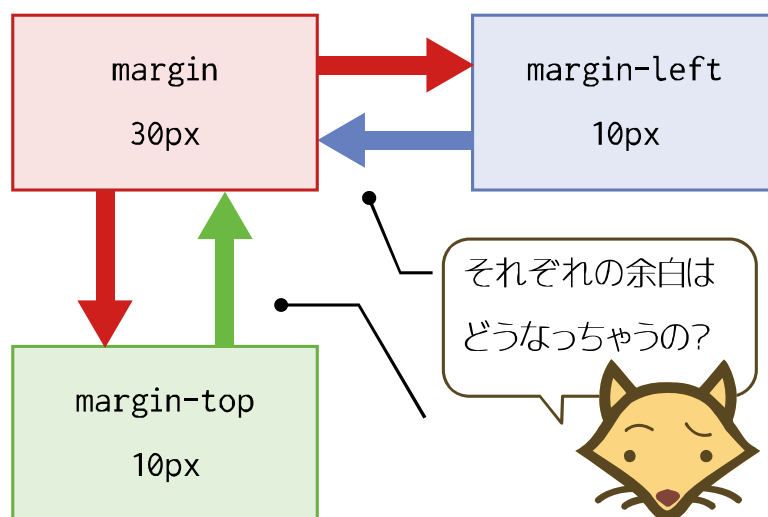
#### ナチュラル

収穫したコーヒーチェリーをそのまま乾燥した後、種子以外をすべて除去して生豆にする。

## margin の特別ルール

対する **margin** には背景色を受けないことからブロック要素間の余白調整で利用されることが多くなります。その際に気を付けなければならないのが、**margin** だけの特別ルールです。

少し手前で **margin** は余白方向を指定できるとを学びましたが、ここでクイズです。以下のように別々のブロック同士で指定した余白がぶつかったとき、**margin** の範囲はそれぞれ合計される？それとも相殺される？さて、実際にどのように表示されるでしょうか？



答えは想像しているよりも複雑です。実は **margin** は左右と上下で結果が異なるのです。以下重要なので必ず覚えておきましょう。

1. margin は**左右でぶつかったときは合計**される。
2. margin は**上下でぶつかったときは数値が大きい方**が優先される。(margin の相殺)

したがって、クイズの答えは上下の **margin** は 30px、左右は 40px の余白で表示されます。

この現象によって意図しないレイアウト崩れが起きやすいので、**margin** を利用する場合は、上下左右の指定方向が衝突しないように、あらかじめ一方向に定めておく と便利 です。上方向を使うのであれば、下指定は利用しない。左方向を利用するなら右方向は利用しない…といった具合です。どちらに寄せるかは設計者と相談しましょう。



## ネガティブマージン

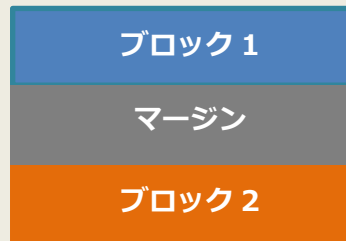
padding と margin には、もう一点違いがあり、margin だけは指定する数値にマイナスが利用できます。**マイナスの値が増えると、指定した方向と逆側に引っ張られる挙動**をします。これはネガティブマージンと呼ばれ、凝ったデザインを施したいときに知っておくと便利です。



## margin の相殺の発生条件

margin の相殺について、上下でぶつかったときは数値が大きい方が優先されると伝えましたが、発生する為の詳細な条件があり、以下の条件といずれかと一致した場合に発生します。

- ・ 隣り合う兄弟要素（並列の関係）の上下である



上図の場合、ブロック 1 の下マージン、ブロック 2 の上マージンが相殺されています。

- ・ 親要素と子要素との間が接している（子要素は先頭または末尾のどちらでもよい）

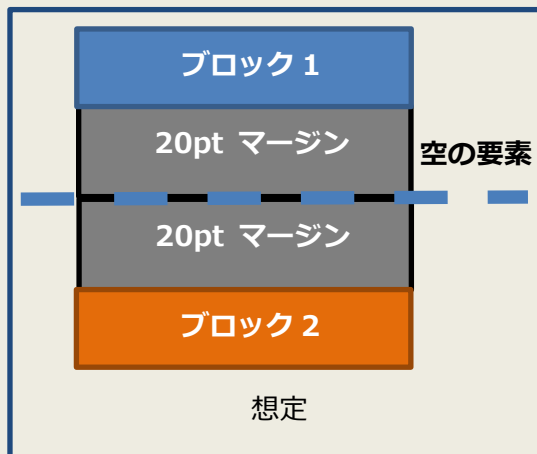


上図の場合親要素の持つ上マージンと子要素の持つ上マージンが相殺されています。

- ・ 空の要素（何も含んでいないブロック要素）が持つ上下のマージン

上下に 20pt マージンを持つ空の要素があった場合、

上マージンと下マージンが相殺され、20pt 分のマージンだけ残ります。



## ブロック要素の枠線(border プロパティ)

ブロック要素のふちに明確な枠線を引きたい場合は、**border** プロパティを使用します。border プロパティでは、枠線の色(**border-color**)、種類(**border-style**)、太さ(**border-width**)をそれぞれ指定できます。

また、4 辺の一括指定が可能である他、**padding** や **margin** 同様に **-top**、**-right**、**-bottom**、**-left** を付与することで個別に指定をすることもできます。

### HTML ソースコード(22\_枠線/index.html)

```
<div id="history">
  <h2>日本での初コーヒー</h2>
  <p>日本には 18 世紀に長崎の出島にオランダ人が持ち込んだところから広がったとされる。</p>
</div><!-- #history -->
```

### CSS ソースコード(22\_枠線/style.css)

```
#history h2{
  background:#fed;
  color:#321;
  padding-left:12px;
  border-left-color:#963;      /* 左の枠線の色 */
  border-left-style:solid;     /* 左の枠線の種類 */
  border-left-width:8px;      /* 左の枠線の太さ */
}
```

### 実行画面



## border-style の種類

**border-style** には以下のような種類があります。他にもいくつかありますが、あまり利用頻度は高くないので最低限下図くらいを押さえておけば問題ありません。また、むやみに線を太くすると、線が目立ってしまい、肝心の本文に注目を集められなくなるので、デザイン的な意図が無い限りは線の太さは 1px もあれば十分です。

solid 実線

dotted 点線

dashed 破線

double 二重線

none なし

## border のショートハンド

**border** には **color** や **style**、**width** といった細かいルールが設定できますが、これらをすべてまとめて書くこともできます。**border** のショートハンドには特に順序の指定が無いので、色のプロパティ、種類、太さを半角スペース区切りでまとめて書くだけです。上記のソースコードをショートハンドで書いた場合は以下のようになります。

### CSS ソースコード(22\_枠線/style.css)

```
border-left-color:#963; /* 左の枠線の色 */
border-left-style:solid; /* 左の枠線の種類 */
border-left-width:8px; /* 左の枠線の太さ */

border-left: #963 solid 8px; /* borderのショートハンド */
```



## ブロック要素の幅と高さの指定方法

ボックスモデルで、余白と枠について学んできましたが、ボックスモデルにはもっとも内側に存在する**コンテンツ**と呼ばれるエリアがあります。コンテンツは指定しなければ、横幅はブロック要素ならばヨコ軸いっぱい、インライン要素ならば含まれる情報の長さに左右され、高さは内包する情報の中でもっともタテに長い数値を持つ要素に左右されます。

意図的に幅と高さを指定したい場合は **width**（幅）プロパティと **height**（高さ）プロパティを利用します。それぞれの大きさを指定できる単位はいくつかありますが、中でもよく使われるのはピクセル（px）、比率（%）になります。ピクセルは固定幅、%は可変幅となり、最近ではスマホ向けに%指定を行うサイトも増えてきたので、ピクセル指定と併せて覚えておきましょう。

なお、インライン要素に **width** と **height** は指定できない点に注意してください。

### HTML ソースコード(23\_コンテンツ/index.html)

```
<div class="wrapper">
  <div class="boxA">boxA</div><!-- boxA -->
  <div class="boxB">boxB</div><!-- boxB -->
</div><!-- wrapper -->
```

### CSS ソースコード(23\_コンテンツ/style.css)

```
.wrapper {
  width:500px;
  background:#cfc;    /*背景グリーン*/
}
.boxA {
  width:100%;
  height:50px;
  background:#aaf;    /*背景ブルー*/
}
.boxB {
  width:200px;
  height:100px;
  background:#faa;    /*背景ピンク*/
}
```



## 実行画面



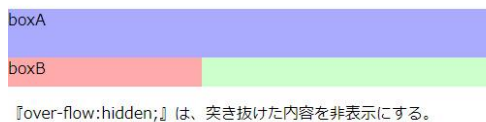
### height は無闇に決めない。

**height** は数値で固定してしまった場合、子要素が固定された高さを超える長さを持った場合、親要素の範囲を突き抜けてしまうという現象が起きます。**overflow** プロパティにより、ある程度は制御できるものの、あまり運用上好ましいとは言えません。高さが少しでも変動する可能性があるブロックには極力 **height** プロパティを指定しないようにしましょう。

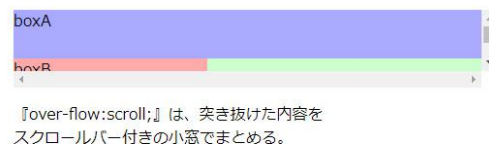
#### heightを超える子要素



#### over-flowプロパティで制御 (hidden)



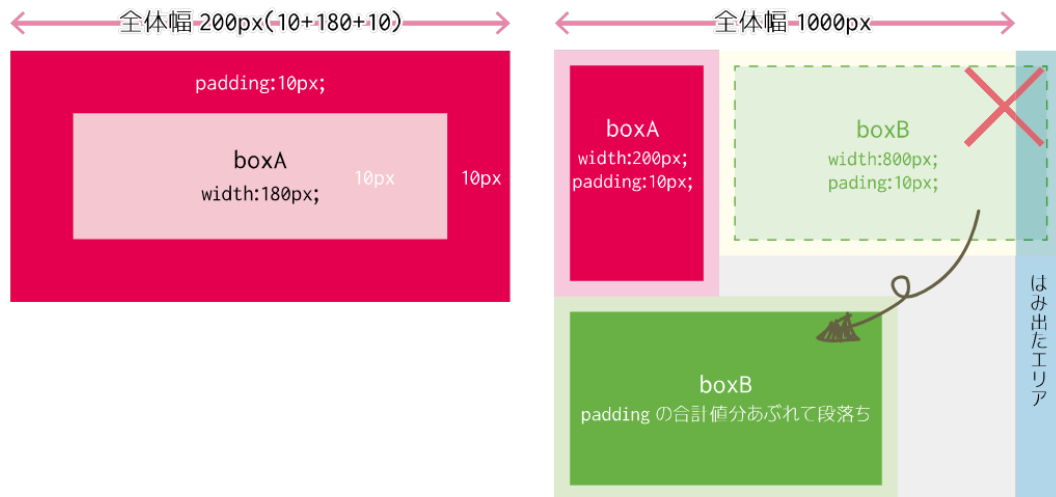
#### over-flowプロパティで制御 (scroll)



**padding、border、margin の数値はコンテンツに含まれない。**

よくある間違いで **width** と **height** で指定した数値の範囲に **padding** や **border** の範囲を含んで考えてしまうケースがあります。例えば **padding** を左右に 10px 持つブロックの横幅を 200px で表示したい場合、ブロックの **width** は何 px で指定するべきかという問題です。

答えは **width : 200px ;** …ではなく、**padding** の左右の合計値 20px を差し引いた **width : 180px ;** になります。この考え方はページレイアウトをする際にとても大切な考え方になります。



右図のように、親要素の横幅ピッタリにブロックを並べようとした場合に、**padding** や **border** 分の数値を差し引いておかないと収まりきらなくなり、結果的に最初の要素の下に落ちてしまいます。これは段（カラム）落ちといい、よく見かけるレイアウト崩れになります。段落ちした場合はまず、親要素と子要素の横幅と余白の合計値を疑ってみましょう。

## ブロックの中央寄せ

ブロックの水平位置を中央に寄せたい場合は、**margin** と **width** を組み合わせて使用します。



まずは、**width** でブロックの横幅を固定しましょう。そして、**margin** を使用して、そのブロックの左右の余白に「**auto**」と指定します。

## HTML ソースコード(24\_中央寄せ/index.html)

```
<div class="center">
  box
</div>
```

## CSS ソースコード(24\_中央寄せ/style.css)

```
.center {
  width:500px;      /* ブロックの横幅を固定 */
  margin:0px auto;  /* ブロックを中央寄せ */
}
```

**テキストとブロックにおける中央寄せの違い**

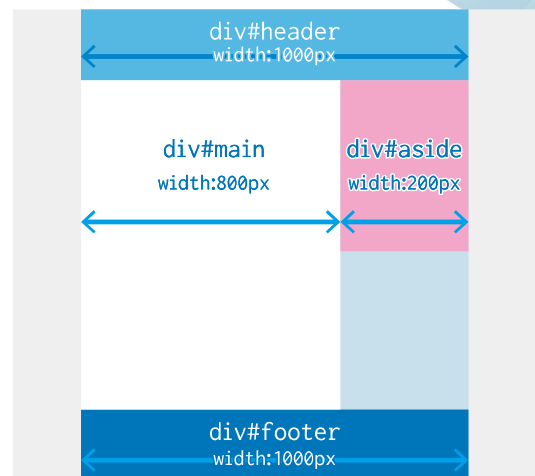
前述の通り、テキストとブロックでは水平方向に中央寄せする方法が異なります。テキストの場合は「**text-align:center**」、ブロックの場合は「**margin:00px auto**」を使用します。混同しないように、注意しましょう。

## 7. ページレイアウト

### ページレイアウト

ここまでの知識で概ねレイアウトを組むための知識が出そろってきました。あとは、ブロック要素を横に並べるための仕組みさえ覚えてしまえばたいのレイアウトは組めるようになります。では、横並びの知識を学ぶついでに次のようなページテンプレートを作成してみましょう。

右図では **main** と **aside** と呼ばれるエリアが横並びになっていますが、いったん横並びは引き出しの中にしまっておいて、まずは今までの知識にならって各ブロックを縦に並べてみましょう。



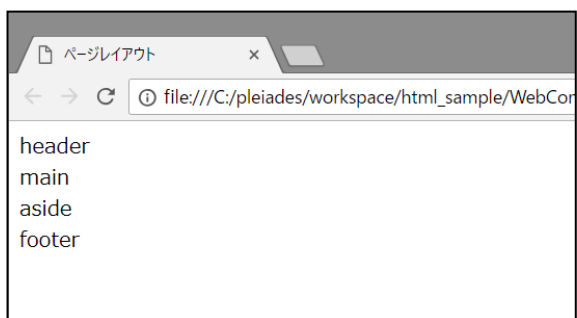
### 各パーツの役割

- header** : 会社/サービス名のロゴや各ページへのナビゲーションリンクを設置
- main** : そのページの本文記事を配置するブロック
- aside** : 同じカテゴリ内のページへ誘導するナビゲーションリンクやバナーを設置
- footer** : コピーライトやサイトマップ、関連企業へのリンクなどを設置するブロック

## HTML ソースコード (25\_ページレイアウト/index01.html)

```
<!DOCTYPE html>
<html lang="ja">
  <head>
    <meta charset="UTF-8" />
    <title>ページレイアウト</title>
  </head>
  <body>
    <div class="wrapper">
      <div id="header">header</div><!-- #header -->
      <div class="container">
        <div id="main">main</div><!-- #main -->
        <div id="aside">aside</div><!-- #aside -->
      </div><!-- #container -->
      <div id="footer">footer</div><!-- #footer -->
    </div><!-- wrapper -->
  </body>
</html>
```

## 実行画面



まだ CSS によるデザインを当てていませんので、シンプルな見た目になっています。ここまでに、各ブロックを縦に並べることができました。次に、HTML ファイル（25\_ページレイアウト/index02.html）を Web ブラウザで開いて、CSS でデザインを当てたものを確認してみましょう。

## HTML ソースコード (25\_ページレイアウト/index02.html)

```
<!DOCTYPE html>
<html lang="ja">
  <head>
    <meta charset="UTF-8" />
    <title>ページレイアウト</title>
    <link rel="stylesheet" href="style02.css" />
  </head>
  <body>
    <div class="wrapper">
      <div id="header">header</div><!-- #header -->
      <div class="container">
        <div id="main">main</div><!-- #main -->
        <div id="aside">aside</div><!-- #aside -->
      </div><!-- #container -->
      <div id="footer">footer</div><!-- #footer -->
    </div><!-- wrapper -->
  </body>
</html>
```

## CSS ソースコード (25\_ページレイアウト/style02.css)

```
#header {  
    width:700px;  
    height:100px;  
    background-color:#4dc;  
}  
#aside {  
    width:200px;  
    height:300px;  
    background-color:#7ffd4;  
}  
#main {  
    width:500px;  
    height:300px;  
    background-color:#fec;  
}  
#footer {  
    width:700px;  
    height:100px;  
    background-color:#dcdcdc;  
}
```



## 実行画面



CSSにより、縦並びにした各ブロックに色がつきました。色によってわかりやすくなりましたが、各ブロック要素は画面の横幅いっぱいに領域をとります。そのため、サイドバーの右側の空白の部分も実際にはサイドバーの領域ということになり、次のブロック要素であるメインはさらにその下から始まることとなります。ブロック要素の性質としてはこのままで正しい表示なのですが、ここからサイドバーとメインブロックを横並びにしたい場合、どのようにすればいいのでしょうか。

## float プロパティによる段組み

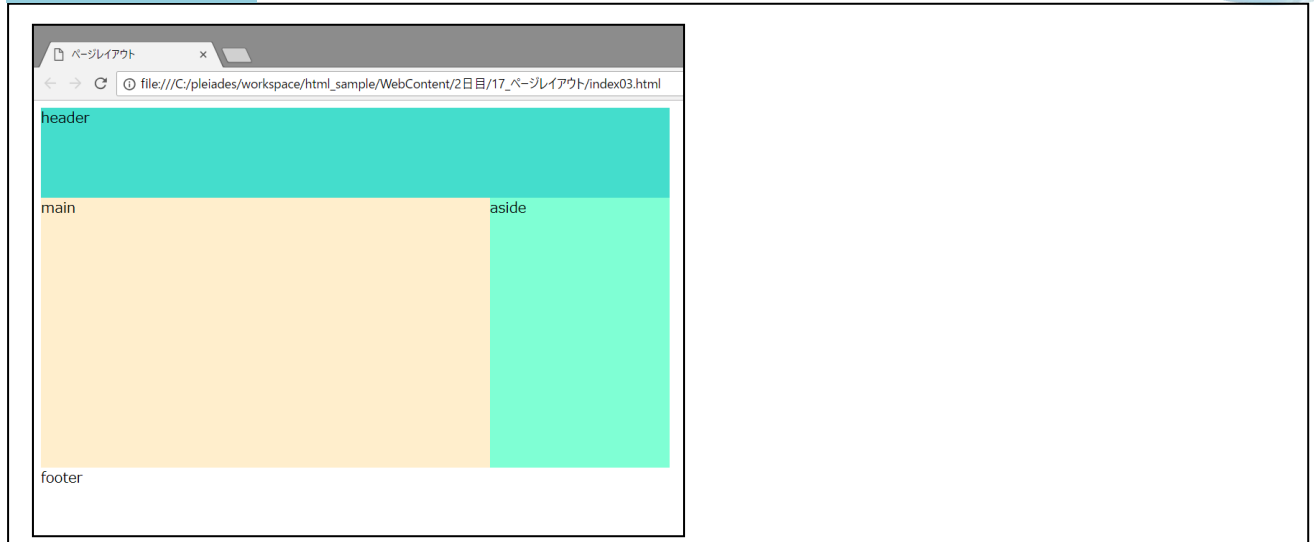
このような場合、**float** プロパティを使用します。**float** プロパティにはブロック要素は左詰めもしくは右詰めに並べる効果がありますので、サイドバーとメインを横に並べる時などは重宝します。次の例はサイドバーとメインを左詰めで並べた CSS 記述例です。

CSS ソースコード (25\_ページレイアウト/style03.css)

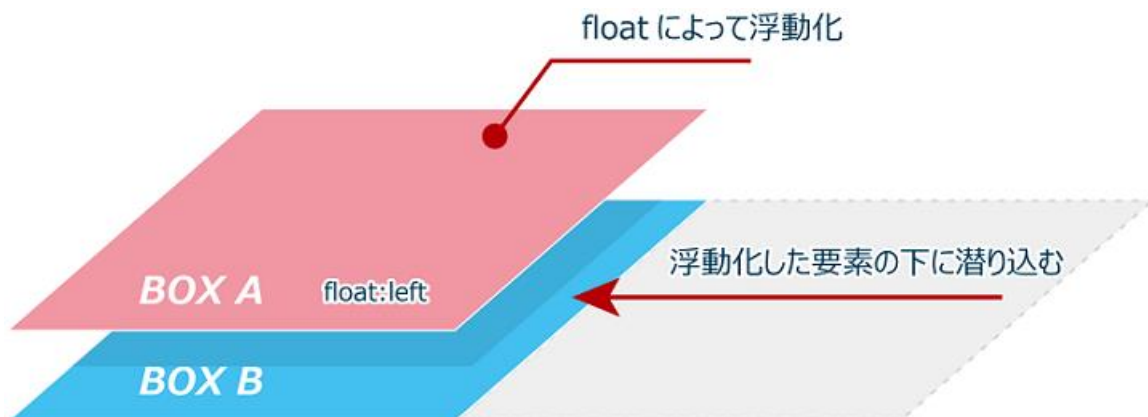
```
#header {  
    width: 700px;  
    height: 100px;  
    background-color: #4dc;  
}  
  
#aside {  
    width: 200px;  
    height: 300px;  
    background-color: #7ffd4;  
    float: left;  
}  
  
#main {  
    width: 500px;  
    height: 300px;  
    background-color: #fec;  
    float: left;  
}  
  
#footer {  
    width: 700px;  
    height: 100px;  
    background-color: #dcdcdc;  
}
```

**float** を使用することで、サイドバーとメインのブロックを横並びにします。HTML ファイル(25\_ページレイアウト/index03.html) を Web ブラウザで開いて、実行結果を確認しましょう。

## 実行画面

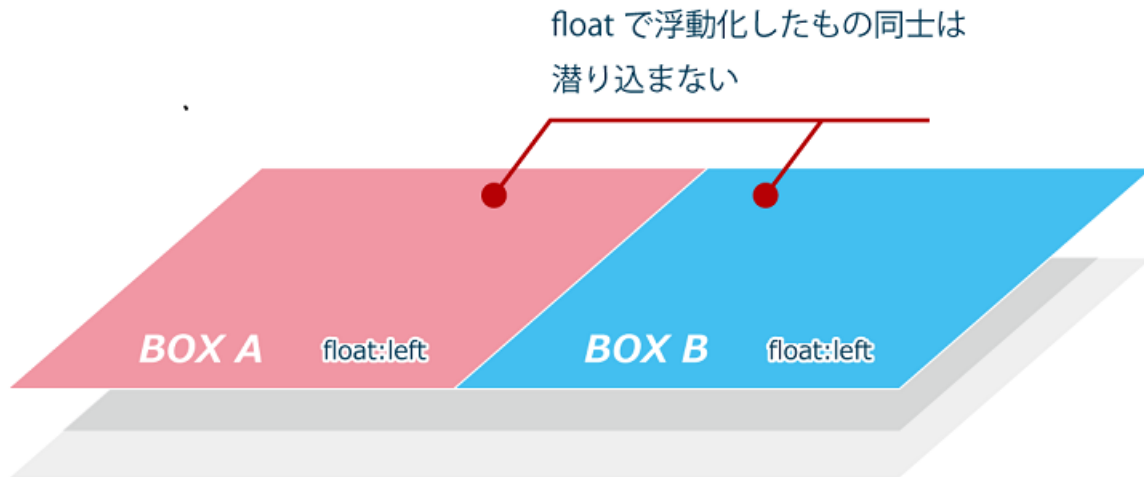


ここで、サイドバーとメインを横並びにすることはできましたが、今度はフッターのブロックが消えてしまいました。この現象は、厳密にはブロックがきえたのではなく、**float** の効果が掛かったブロックの下に隠れた状態です。**float** を使用すると、対象のブロックは宙に浮いた状態となります。その状態になることを「**浮動化**」と呼びます。浮動化したブロックの下には隙間が生まれます。すると、そのブロックに続く他のブロックは、その隙間に潜り込んでしまいます。そのため、浮いたブロックに隠れてしまい、結果として画面上から消えてしまったように見えます。



ただし、ブロック内のテキスト情報などは、ブロックの下に潜らず、画面上に表示されます。

また、浮動化したブロック同士では潜り込みは発生しません。



## float の解除

**float** の影響でブロックが隠れてしまう問題を解消するには、**float** の影響から解除する必要があります。このことを「**float 解除**」と呼びます。

**float** を解除する方法は主に以下の 2 種類があります。

- ① ブロックに「**overflow:hidden;**」を追加する。
- ② ブロックに「**clear**」プロパティを追加する。

### 「overflow:hidden;」の追加

**float** 解除したいブロックに対して、「**overflow:hidden**」を追加してください。すると、そのブロックは他のブロックの下に隠れず、正常に表示されます。

**overflow** は本来、ブロック中のテキスト量がブロックのサイズを超過した場合、ブロックにスクロールバーを表示させるためのプロパティです。しかし、値として「**hidden**」を指定することで、付加的な効果として **float** を解除できます。

具体的には、横並びにしたいブロックの親要素に対して、「**overflow:hidden**」を適用させます。上記のサンプルコードでいえば、「<div class="container">」が対象の親要素となります。下記のようにスタイルシートを追加することで、「<div class="container">」が子要素の高さを的確に把握し、footer 要素を潜り込ませないようにします。

## CSS ソースコード (25\_ページレイアウト/style04.css)

```
.container {  
    width:700px;  
    height:300px;  
    overflow:hidden;  
}
```

## 「clear」プロパティの追加

**clear** とは、**float** の解除を行うためのプロパティです。float 解除したいブロックに対して、**clear** を追加してください。そして、値として以下のいずれかを指定してください。

値	機能
left	「float:left」からの影響を解除する。
right	「float:right」からの影響を解除する。
both	「float:left」、および「float:right」からの影響を解除する。
none	解除しない（初期値）。

値に関わらず、float 解除したい場合は「**clear:both;**」と指定しておくことが一般的です。

## CSS ソースコード (25\_ページレイアウト/style05.css)

```
footer {  
    width:700px;  
    height:100px;  
    background-color:#dcdcdc;  
    clear:both;  
}
```

上記 2 種類の手法を取った HTML ファイル 2 つ（25\_ページレイアウト/index04.html と 25\_ページレイアウト/index05.html）をそれぞれ Web ブラウザで開いて、実行結果を確認しましょう。

## 実行画面





## clearfix による float 解除

「overflow:hidden」や「clear」を使用した方法以外に、「clearfix」を使用しても float 解除が可能です。clearfix は、clear と同様に float 解除のための CSS の機能であり、2004 に公表されました。clearfix は、疑似要素というものを利用して、ある要素内の子要素に対してまとめて float 解除するという特徴があります。

たとえば、下記のような入れ子形式でブロックが存在し、子要素（class 属性の値が「child1」、「child2」のブロック）が、float により浮動化しているとします。

```
<div class="parent clearfix">

  <div class="child1">child1</div>

  <div class="child2">child2</div>

</div>

<div class="footer">footer</div>
```

```
div.clearfix:after {

  clear:both;

}
```

CSS 中の「:after」は疑似要素であり、手前側のセレクトタで特定された要素よりも後ろにある要素を CSS の適用対象にします。したがって、上記のサンプルコードの場合、class 属性の値が「footer」であるブロックは、class 属性の値が「clearfix」である要素内の子要素に対して float 解除されます。

clearfix は、float 解除の方法としては一般的ではありませんが、このような方法もあるということは知っておきましょう。





## flexbox によるページレイアウト

float プロパティ以外にもレイアウトを調整する CSS の機能がありますが、flexbox はその中でもよく使われるものです。ここでは詳細な説明は行いませんが、親要素をコンテナとし、子要素を縦横好きな方向に並べることができます。このような、ページレイアウトの仕方もあると名前だけ知っておくといいでしょう。

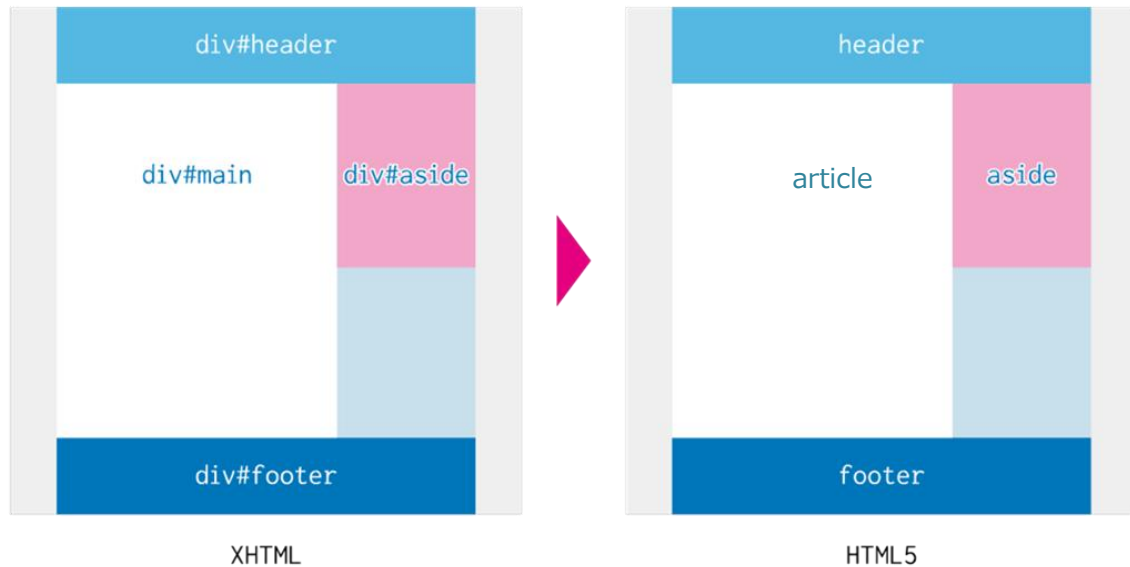
以下の例は flexbox の基本であり、これらに加えて様々な操作可能プロパティが存在します。

```
<div class="flex-container">
  <div class="child">child1</div>
  <div class="child">child2</div>
</div>
<div class="footer">footer</div>
```

```
div.flex-container {
  display: flex;
}
```

上に記述した `display: flex` を使用することで flexbox としての各種機能を使用することができるようになります。また `display: flex` が適応されているブロックの子要素はフレックスアイテムとして機能し、同じように flexbox の子要素専用の機能を使用することができるようになります。

ここまでの例で記述した書き方はマークアップ上間違っていないのですが、特に **body** の子要素たちが少し前の XHTML で使われていた記述方法になります。HTML5 以降では、ただ **div** 要素として記述していたブロックパーツに対して、明確な意味付けをするタグが用意されています。せっかくですので、ここで新しい記述に改造してみましょう。



HTML ソースコード (25\_ページレイアウト/index06.html)

```
<!DOCTYPE html>
<html lang="ja">
  <head>
    <meta charset="UTF-8">
    <title>ページレイアウト</title>
    <link rel="stylesheet" href="style06.css">
  </head>
  <body>
    <div class="wrapper">
      <header>header</header>
      <div class="container">
        <article>main</article>
        <aside>aside</aside>
      </div><!-- #container -->
      <footer>footer</footer>
    </div><!-- #wrapper -->
  </body>
</html>
```

## CSS ソースコード (25\_ページレイアウト/style06.css)

```
header {  
    width: 700px;  
    height: 100px;  
    background-color: #4dc;  
}  
  
aside {  
    width: 200px;  
    height: 300px;  
    background-color: #7ffd4;  
    float: left;  
}  
  
article {  
    width: 500px;  
    height: 300px;  
    background-color: #fec;  
    float: left;  
}  
  
footer {  
    width: 700px;  
    height: 100px;  
    background-color: #dcdcdc;  
    clear: both;  
}
```

## 変更点

<code>div#header</code>	→	<b>header</b>
<code>div#main</code>	→	<b>article</b>
<code>div#aside</code>	→	<b>aside</b>
<code>div#footer</code>	→	<b>footer</b>

`div#main` は `article` ではなく、`main` というタグに置き換えて記述することも可能です。

## <header>タグ

<header>タグとは、今まで使用していた<div>と大きく異なるものではありません。ページ内のコンテンツにおける冒頭、導入部分であると宣言しているだけであり、このタグを使ったからといって、実際の web ページで変化するものではありません。

利点としては、ヘッダーであることを明示でき、他者が見た際にわかりやすくなります。よって HTML のコードがわかりやすくなり、CSS でもタグ指定でヘッダーを設計できるようになります。注意としては、<div> と異なり、入れ子構造に作成することはできません。

## <article>タグ

<article>タグは<header>タグとは宣言している部分が異なり、本文、主題部分であると宣言しているだけで、性質は同じです。また、<article>同じ役割として<main>があります。どちらを使ってもかまいません。

## <aside>タグ

<aside>タグも<header>タグと同じですが、主題と関係する、本筋ではない部分に使用します。<aside>タグがよく使われるのはサイドバーになります。

## <footer>タグ

<footer>タグも<header>タグと同じですが、フッターとなる部分を宣言しています。フッター内には作者や企業名などを記述することが多いです。

## <section>タグ

<section>タグも<header>タグと同じですが、以上の4つのタグに当てはまらず、構成部分としてひとくくりにまとめたい際に使用します。主に、<article>タグの中にで、要素を分けたい際に使用することが多いです。また、使用する際には、<h1>~<h6>を子要素にすることが多いです。

## 8. フォーム

ここまでは、HTML と CSS を組み合わせた画面レイアウトの方法を紹介しました。ここからは、フォームという HTML の機能について紹介します。

様々な Web 上のサービスでは、キーボードで文字を入力する、ボタンを選択するなど、ユーザーが何らかの入力を行える画面が利用されます。このような入力項目のことを「**入力フォーム**（以下、**フォーム**）」と呼びます。フォームは、クライアントから Web サーバへ情報を送信するための機能を持っており、Web アプリケーション開発において必ず知っておくべき知識です。

本書では、フォームの基本的な機能を紹介します。フォームはサーバ上で実行されるプログラムと連携して初めて機能するため、実践的な実装方法については別単元で紹介します。

### フォーム領域

フォームを作成するには、まずは**<form>**タグを使用して、Web サーバにデータを送信するためのフォーム領域を作成する必要があります。**<form>**タグには入力値の送信先を設定する機能があるため、このタグがないと、入力項目が機能しません。

また、画面上の入力項目のことを「入力コントロール（以下、コントロール）」と呼びます。対応する主な要素は**<input>**、**<textarea>**、**<select>**の3つです。各コントロールは**<form>**タグ内の子要素として記述します。

**<form>**タグは、**action**属性で指定された URL に各コントロールの入力値を送信します。また、入力値の送信方法は **method** 属性で指定します。

属性名	値	説明
action	入力値の送信先（URL）	データの送信先 URL を指定します。
method	get（URL に付けて入力値を送信） post（入力値をそのまま送信）	データの送信方法を設定します。設定を省略した場合は get になります。

## コントロール

コントロールを作成する各要素には、共通して **name** 属性と **value** 属性という 2 種類の重要な属性があります。これらの属性により、プログラムはサイト閲覧者の入力情報を取得できます。

属性名	値	説明
name	任意の文字列	各コントロールを識別するための名前
value	入力値	入力値

### <input>タグ

**<input>**タグは、フォームのコントロールを作成するための要素です。**<input>**タグは、**name** 属性と **value** 属性の他に、**type** 属性を指定できることが特徴で、コントロールの種類を指定できます。コントロールには、「テキストボックス」、「チェックボックス」、「ラジオボタン」、「送信ボタン」など、機能が異なる様々な種類が存在します。

属性名	値	説明
type	text(テキスト入力) number(数値入力) password(パスワード入力) email(メールアドレス入力) checkbox(チェックボックス) radio(ラジオボタン) file(ファイル参照) hidden(非表示項目) image(画像ボタン) reset(リセットボタン) submit(送信ボタン) button(ボタン)	コントロールの種類を指定

**value** 属性の値を指定しておくと、その値が対象のコントロールの初期値として入力された状態となります。なお、**type** 属性の値が **reset**、**submit**、**button** の場合、**value** 属性の値はボタン上に表記される文言となります。

また、**type** 属性の値が **radio**、または **checkbox** のコントロールで、選択肢を複数用意したい場合、各コントロールの **name** 属性には同一の値を指定しましょう。値が異なると、各ボタンが別グループの選択肢として扱われてしまいます。例えば、「本来複数の選択肢から 1 つを選択するた

めのラジオボタン（**radio**）で、複数のボタンを選択できてしまう」などです。

また、**input** 要素で作成されるコントロールは種類によって固有の属性もあります。

属性名	値	説明
checked	checked	type 属性が radio または checkbox の場合、初期状態でチェックされている項目であることを指定する。
maxlength	整数(最大文字数)	type 属性が password または text の場合、入力できる最大文字数を指定する。
size	整数(半角での文字数)	type 属性が password または text の場合、コントロールの大きさを文字数で指定する。  あくまで表示の大きさであり、入力できる文字数とは無関係である。

## <textarea>タグ

**<textarea>** タグは、複数行入力できるテキストボックス（テキストエリア）を作成するための要素です。コントロールの大きさは **cols** 属性(横幅)と **rows** 属性(高さ)によって決まり、それぞれ文字数と行数で指定します。このコントロールには value 属性がありませんが、初期値は要素の内容で決めることができます。

属性名	値	説明
cols	整数(半角での文字数)	テキストボックスの横幅を文字数で指定する。
rows	整数(行数)	テキストボックスの高さを行数で指定する。

なお、**input** 要素の **size** 属性や、**textarea** 要素の **cols** 属性により幅を指定すると、ブラウザによって表示幅が異なってしまいます。そのため、幅の指定には CSS の **width** プロパティを使用することが一般的です。



## <select>タグ

**<select>**タグは、ドロップダウンリストを作成するためのタグです。選択肢は子要素の**<option>**タグで記述します。

なお、**<select>**タグでは、**name** 属性は指定できますが、**value** 属性は指定しません。**value** 属性は**<option>**タグで指定します。

## <option>タグ

**<option>**タグは、ドロップダウンリストの選択肢を作成するもので、**<select>**タグの子要素として記述します。

プログラムに送る値を **value** 属性で指定し、画面に表示されるテキストは要素の内容として記述します。**selected** 属性を指定すると、あらかじめその選択肢が選択された状態になります。

属性名	値	説明
selected	selected	その選択肢があらかじめリストで選択されていることを指定する。

## フォームの作成

それでは簡単なフォームを作成してみましょう。

HTML ソースコード(26\_入力フォーム/index.html)

```
<form action="xxx">
  <p>お名前<br>
    <input type="text" name="name" /></p>
  <p>お住まいの地域<br />
    <select name="area">
      <option value="1">北海道・東北</option>
      <option value="2">関東</option>
      <option value="3">中部</option>
      <option value="4">関西</option>
      <option value="5">中国・四国</option>
      <option value="6">九州・沖縄</option>
```

```
<select>

</p>
<p>電話番号<br />

    <input type="number" name="phone" /></p>
<p>eメール<br />

    <input type="e-mail" name="mail" /></p>
<p>お問い合わせ種類<br />

    <label><input type="radio" name="type" value="1" />商品について</label>
    <label><input type="radio" name="type" value="2" />当サイトについて</label>
</p>
<p>お問い合わせ内容<br>

    <textarea name="detail" cols="30" rows="5"></textarea></p>
<p><input type="submit" value="送信" /></p>
</form>
```

## 実行画面

入力フォーム

お名前

お住まいの地域

北海道・東北

電話番号

eメール

お問い合わせ種類

☐ 商品について ☐ 当サイトについて

お問い合わせ内容

送信