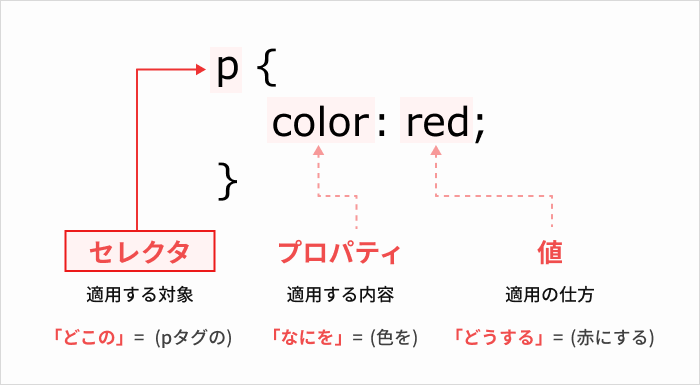
## **疑似要素**

**疑似要素**とは簡単にいえば、「要素の**一部**に対してスタイルを適用できるセレクタ」のことです。

ここで、もう一度セレクタについて復習しておきましょう。セレクタとは簡単にいえば、スタイルを適用する「**対象**」のことでした。



上記の例であればセレクタ（p要素）に対してcolor: red;を指定しているため、例えば <p>段落です</p>であれば「段落です」という要素全体の文字色が赤色になります。

では要素全体ではなく、要素の**一部**にスタイルを適用したいときはどうすればよいでしょうか。

HTMLファイルを直接編集したり、タグを追加してそれにスタイルを適用したりするのも1つの手ですが、より簡単な方法があります。それが疑似要素です。

疑似要素を使うことで、HTMLファイルを編集せずとも要素の一部に対してスタイルを適用できるようになるため、デザインの幅が大きく広がります。

実際に使ってみたほうがわかりやすいので、代表的な4つの疑似要素を例にして理解を深めていきましょう。

* ::before：要素の**直前**にコンテンツを追加する
* ::after：要素の**直後**にコンテンツを追加する
* ::first-letter：要素の**最初の文字**を装飾する
* ::first-line：要素の**最初の行**を装飾する

あらかじめ、HTML\_CSSフォルダ内に新しくpseudo-elements.htmlとpseudo-elements.cssという2つのファイルを作成し、pseudo-elements.htmlを以下のように編集しておいてください。pseudo-elements.cssはまだ何も編集しなくてOKです。

* pseudo＝よく似た、疑似の（※覚える必要はありません）

+<!DOCTYPE html>

+<html>

+ <head>

+ <title>タイトルです</title>

+ <meta charset="utf-8">

+ <link rel="stylesheet" href="pseudo-elements.css">

+ </head>

+ <body>

+ <h1>見出し1です</h1>

+ <p>段落1の1行目です<br>段落1の2行目です</p>

+ <h1>見出し2です</h1>

+ <p>段落2の1行目です<br>段落2の2行目です</p>

+ </body>

+</html>

### **before**

::beforeは、要素の**直前**にコンテンツ（文字や記号、アイコンなど）を追加する疑似要素です。

::beforeを使うには、以下のようにセレクタ（要素やクラス、id）の後に::beforeをつけるだけでOKです。これは他の疑似要素や疑似クラスでも同様です。

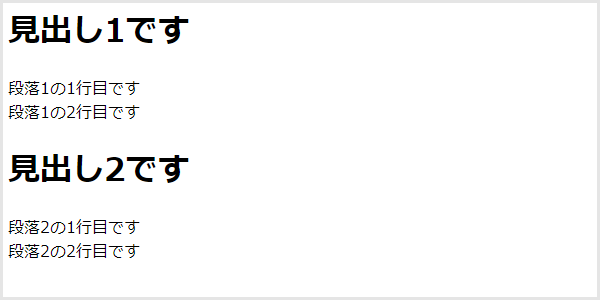
p::before {

content: "この中に追加するコンテンツを書きます";

}

::beforeの場合は、contentプロパティの値に追加したいコンテンツを指定します。

* contentプロパティ：要素の直前または直後に挿入するコンテンツを指定する。疑似要素の::beforeまたは::afterのみに使われる



ここで、::beforeを使ってh1要素の**直前**にひし形の記号（◆）を追加してみましょう。

pseudo-elements.cssを以下のように編集したあと、pseudo-elements.htmlをブラウザで開いて出力結果を確認してみてください。

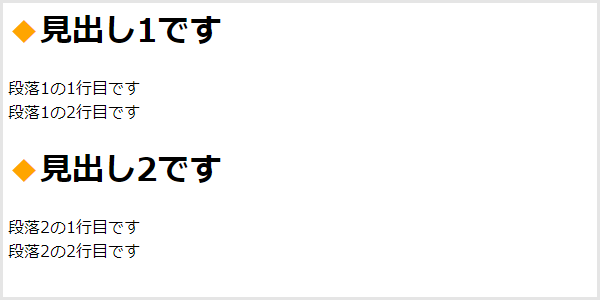
+h1::before {

+ content: "◆";

+ color: orange;

+}

以下のように、h1要素の**直前**にひし形の記号（◆）が追加されました。また、同時にcolor: orange;を指定したのでオレンジ色になっています。



### **::after**

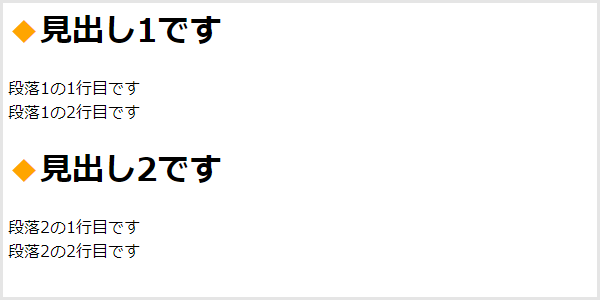
::afterは、要素の**直後**にコンテンツ（文字や記号、アイコンなど）を追加する疑似要素です。::afterの使い方は::beforeと同じです。

p::after {

content: "この中に追加するコンテンツを書きます";

}

では実際にやってみましょう。現状では、pseudo-elements.htmlをブラウザで開くと以下のように表示されます。



ここで、::afterを使ってh1要素の**直後**に四角の記号（■）を追加してみましょう。

pseudo-elements.cssを以下のように編集したあと、pseudo-elements.htmlをブラウザで開いて出力結果を確認してみてください。

h1::before {

content: "◆";

color: orange;

}

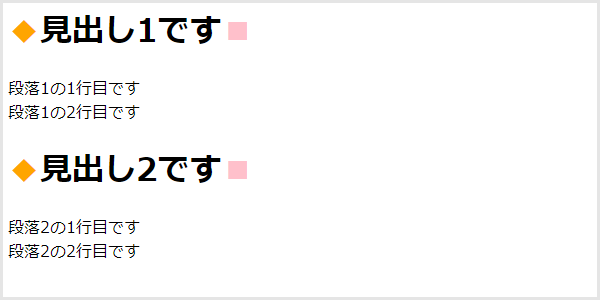
+h1::after {

+ content: "■";

+ color: pink;

+}

以下のように、h1要素の**直後**に四角の記号（■）が追加されました。また、同時にcolor: pink;を指定したのでピンク色になっています。



### **::first-letter**

::first-letterは、要素の**最初の文字**を装飾する疑似要素です。

::first-letterの使い方は基本的に::beforeや::afterと一緒ですが、::first-letterでは[13章](https://terakoya.sejuku.net/programs/51/chapters/588)で学んだ文字装飾のプロパティを使います。

p::first-letter {

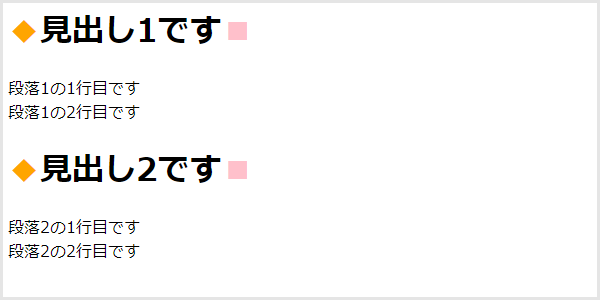
/\* 文字装飾のプロパティを使います（以下はあくまで一例です） \*/

font-size: 24px;

font-weight: bold;

}

現状では、pseudo-elements.htmlをブラウザで開くと以下のように表示されます。



ここで、::first-letterを使ってp要素の**最初の文字**のフォントサイズを24pxにし、太字にしてみましょう。

pseudo-elements.cssを以下のように編集したあと、pseudo-elements.htmlをブラウザで開いて出力結果を確認してみてください。

h1::before {

content: "◆";

color: orange;

}

h1::after {

content: "■";

color: pink;

}

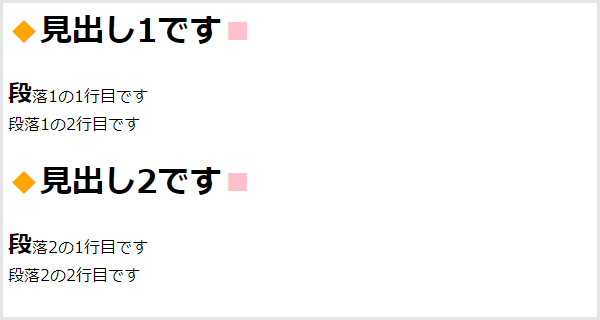
+p::first-letter {

+ font-size: 24px;

+ font-weight: bold;

+}

以下のように、p要素の**最初の文字**のフォントサイズが24pxになり、太字になりました。



### **::first-line**

::first-lineは、要素の**最初の行**を装飾する疑似要素です。::first-lineの使い方は::first-letterと同じです。

p::first-line {

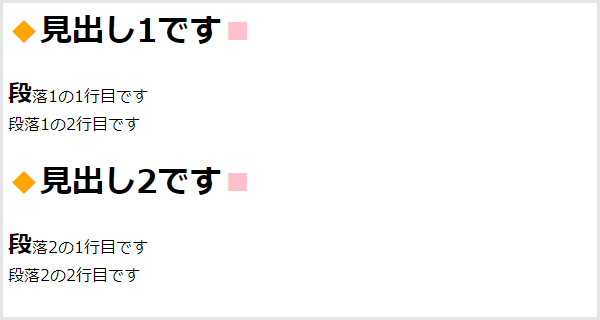
/\* 文字装飾のプロパティを使います（以下はあくまで一例です） \*/

font-size: 24px;

font-weight: bold;

}

現状では、pseudo-elements.htmlをブラウザで開くと以下のように表示されます。



ここで、::first-lineを使ってp要素の**最初の行**のフォントサイズを24pxにし、太字にして文字色をグレーにしてみましょう。

pseudo-elements.cssを以下のように編集したあと、pseudo-elements.htmlをブラウザで開いて出力結果を確認してみてください。

h1::before {

content: "◆";

color: orange;

}

h1::after {

content: "■";

color: pink;

}

+p::first-line {

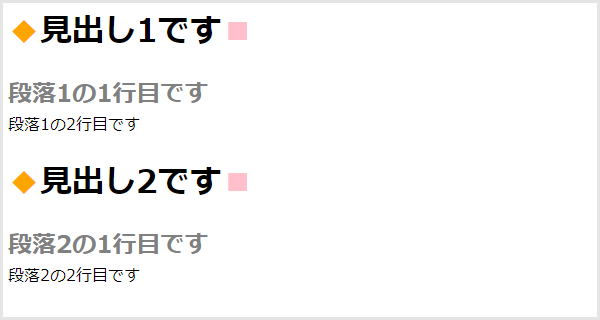
+ font-size: 24px;

+ font-weight: bold;

+ color: gray;

+}

以下のように、p要素の**最初の行**のフォントサイズが24pxになり、太字になって文字色がグレーになりました。



まとめ

* **疑似要素**とは「要素の**一部**に対してスタイルを適用できるセレクタ」のことである
* 代表的な4つの疑似要素
  + ::before：要素の**直前**にコンテンツを追加する
  + ::after：要素の**直後**にコンテンツを追加する
  + ::first-letter：要素の**最初の文字**を装飾する
  + ::first-line：要素の**最初の行**を装飾する

## **疑似クラス**

**疑似クラス**とは簡単にいえば、「要素が**特定の状態**のときに適用できるセレクタ」のことです。

特定の状態というのは、例えば「そのリンクにアクセス済みのとき」「ボタンの上にマウスのカーソルが乗っているとき」などです。

疑似クラスを使うことで、要素の状態によって適用するスタイルを変えられるので、デザインの幅が大きく広がります。

実際に使ってみたほうがわかりやすいので、代表的な7つの疑似クラスを例にして理解を深めていきましょう。

* :link：**未訪問**のリンク（a要素など）にスタイルを適用する
* :visited：**訪問済み**のリンク（a要素など）にスタイルを適用する
* :hover：要素の上に**マウスのカーソルが乗っているとき**にスタイルを適用する
* :active：要素が**アクティブ状態**（マウスを左クリックしてから離すまで）のときにスタイルを適用する
* :nth-child(n)：親要素内にある**すべての要素**の中で、**n番目**の要素にスタイルを適用する
* :nth-of-type(n)：親要素内にある**同じ要素**の中で、**n番目**の要素にスタイルを適用する
* :not(A)：親要素内にあるAという要素**以外に**スタイルを適用する

なお、疑似要素は::beforeのように:（コロン）が2つありましたが、疑似クラスは:linkのように:（コロン）が**1つ**です。覚えておきましょう。

あらかじめ、HTML\_CSSフォルダ内に新しくpseudo-classes.htmlとpseudo-classes.cssという2つのファイルを作成し、以下のように編集しておいてください。

* pseudo＝よく似た、疑似の（※覚える必要はありません）

<!DOCTYPE html>

2 <html>

3 <head>

4 <title>タイトルです</title>

5 <meta charset="utf-8">

6 <link rel="stylesheet" href="pseudo-classes.css">

7 </head>

8 <body>

9 <div class="parent">

10 <h1>見出し1です</h1>

11 <a href="#">リンクです</a>

12 <p>段落1です</p>

13 <p>段落2です</p>

14 <p>段落3です</p>

15 <h2>見出し2です</h2>

16 <p>段落4です</p>

17 <p>段落5です</p>

18 <p>段落6です</p>

19 </div>

20 </body>

21 </html>

1.parent {

2 background-color: whitesmoke;

3 }

4

### **:link**

:linkは、**未訪問**のリンク（a要素など）にスタイルを適用する疑似クラスです。なお未訪問とは、「そのリンクにまだアクセスしたことがない状態」のことを指します。

:linkを使うには、疑似要素と同じようにセレクタ（要素やクラス、id）の後に:linkをつけるだけでOKです。これは他の疑似クラスでも同様です。

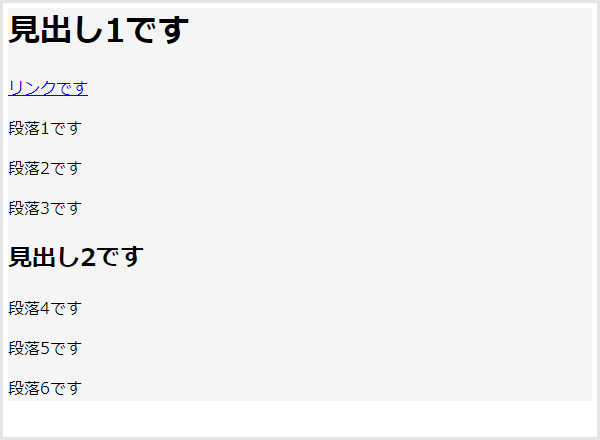
a:link {

color: orange;

}

なお、:linkは基本的に未訪問のリンクの**文字色**を変更するために使われます。

では実際にやってみましょう。現状では、pseudo-classes.htmlをブラウザで開くと以下のように表示されます。なお、一度でも「リンクです」というテキストをクリックしてしまうと、未訪問ではなく**訪問済み**の状態になってしまうのでクリックしないようにしてください。



ここで、:linkを使って**未訪問**のリンク（a要素）の文字色をオレンジ色に変更してみましょう。

pseudo-classes.cssを以下のように編集したあと、pseudo-classes.htmlをブラウザで開いて出力結果を確認してみてください。

.parent {

background-color: whitesmoke;

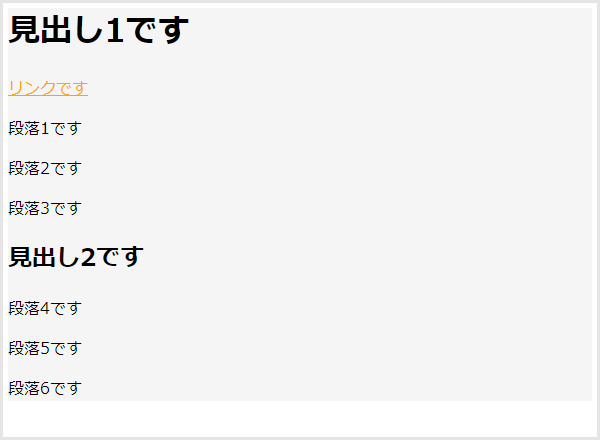
}

+a:link {

+ color: orange;

+}

以下のように、**未訪問**のリンク（a要素）の文字色がオレンジ色になりました。



### **:visited**

:visitedは、**訪問済み**のリンク（a要素など）にスタイルを適用する疑似クラスです。なお訪問済みとは、「そのリンクにすでにアクセスしたことがある状態」のことを指します。

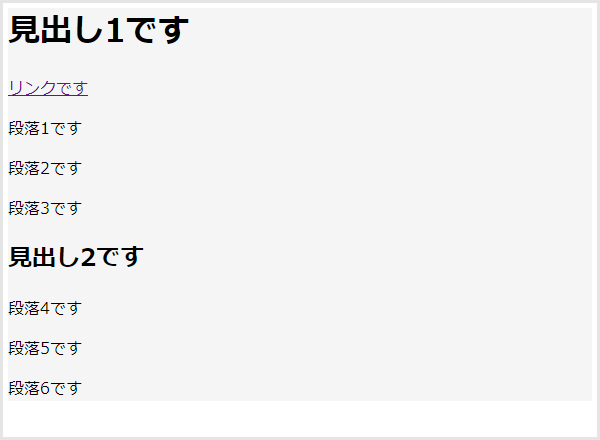
:visitedの使い方は:linkと一緒です。

a:visited {

color: orange;

}

では実際にやってみましょう。まずはpseudo-classes.htmlをブラウザを開き、「リンクです」というテキストをクリックしてみてください。すると、テキストの色が変わります。これが**訪問済み**の状態です。



ここで、:visitedを使って**訪問済み**のリンク（a要素）の文字色を緑色に変更してみましょう。

pseudo-classes.cssを以下のように編集したあと、pseudo-classes.htmlをブラウザで開いて出力結果を確認してみてください。

.parent {

background-color: whitesmoke;

}

a:link {

color: orange;

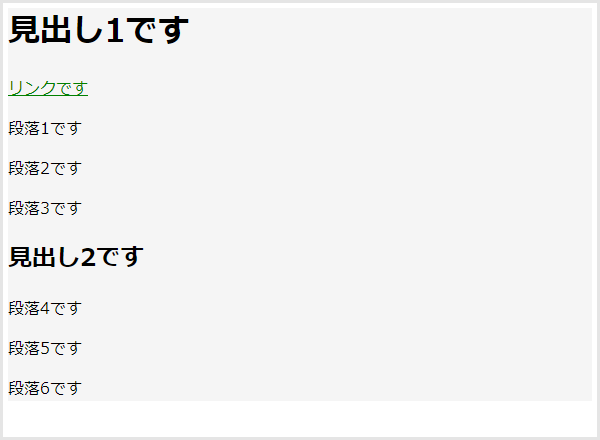
}

+a:visited {

+ color: green;

+}

以下のように、**訪問済み**のリンク（a要素）の文字色が緑色になりました。



### **:hover**

:hoverは、要素の上に**マウスのカーソルが乗っているとき**にスタイルを適用する疑似クラスです。

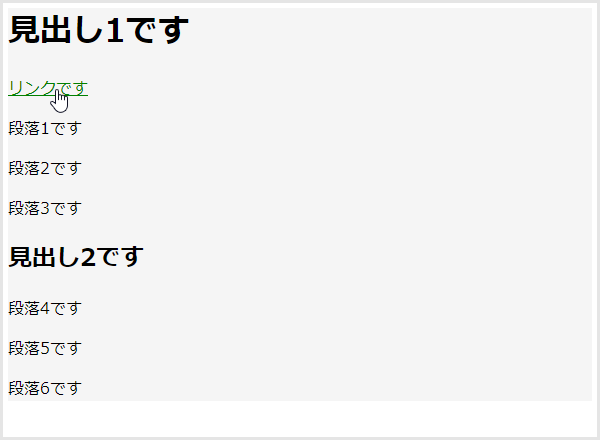
使い方は:linkや:visitedと一緒ですが、:hoverはボタンなどリンク以外にも使われます。

a:hover {

color: orange;

}

では実際にやってみましょう。まずはpseudo-classes.htmlをブラウザを開き、「リンクです」というテキストの上にマウスのカーソルを乗せてみてください。現状では何も変化しません。



ここで、:hoverを使って**マウスのカーソルが乗っているとき**の文字色を紫色に変更してみましょう。

pseudo-classes.cssを以下のように編集したあと、pseudo-classes.htmlをブラウザで開いて出力結果を確認してみてください。

.parent {

background-color: whitesmoke;

}

a:link {

color: orange;

}

a:visited {

color: green;

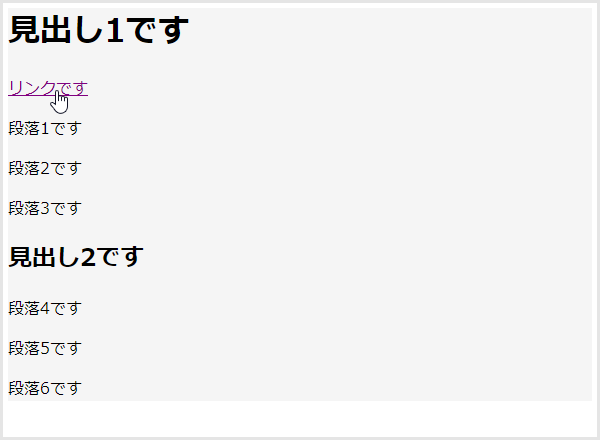
}

+a:hover {

+ color: purple;

+}

「リンクです」というテキストの上にマウスのカーソルを乗せてみましょう。以下のように、**マウスのカーソルが乗っているとき**の文字色が紫色になりました。



### **:active**

:activeは、要素が**アクティブ状態**のときにスタイルを適用する疑似クラスです。

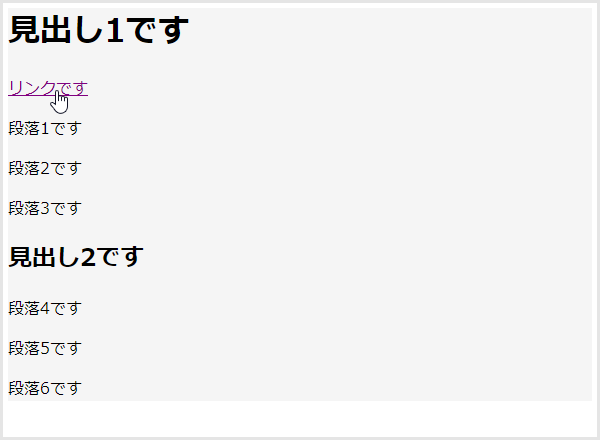
なお、アクティブ状態とは基本的に「マウスを左クリックしてから離すまで」の状態のことを指します。:activeはa要素やbutton要素などで使われます。

a:active {

color: orange;

}

では実際にやってみましょう。まずはpseudo-classes.htmlをブラウザを開き、「リンクです」というテキストの上で、マウスを左クリックしたままにしてみてください。現状では何も変化しません。



ここで、:activeを使ってリンク（a要素）が**アクティブ状態**のときの文字色を赤色に変更してみましょう。

pseudo-classes.cssを以下のように編集したあと、pseudo-classes.htmlをブラウザで開いて出力結果を確認してみてください。

.parent {

background-color: whitesmoke;

}

a:link {

color: orange;

}

a:visited {

color: green;

}

a:hover {

color: purple;

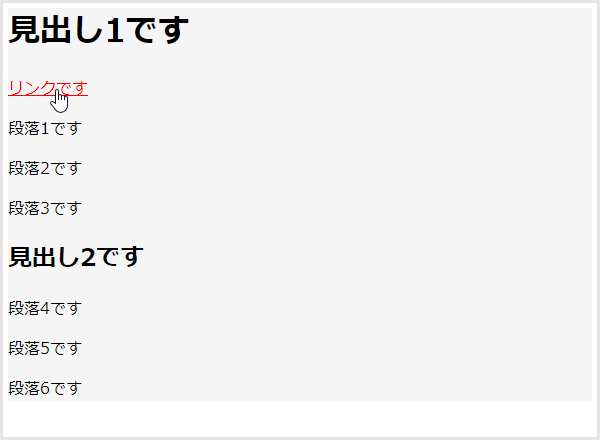
}

+a:active {

+ color: red;

+}

リンクです」というテキストの上で、マウスを左クリックしたままにしてみましょう。以下のように、**アクティブ状態**のときの文字色が赤色になりました。



### **:nth-child(n)**

:nth-child(n)は、親要素内にある**すべての要素**の中で、**n番目**の要素にスタイルを適用する疑似クラスです。

なお、nthは4th、5thなどに使われる接尾辞thを使うことで、「n番目」を表しています。つまり、nth-childは「n番目の子要素」という意味です。

例えば以下のようなHTMLファイルがあったとします。親要素であるdiv要素の中に、子要素としてh1要素が1つ、p要素が3つ入っている状態です。

<div>

<h1>見出しです</h1>

<p>段落1です</p>

<p>段落2です</p>

<p>段落3です</p>

</div>

div p:nth-child(2) {

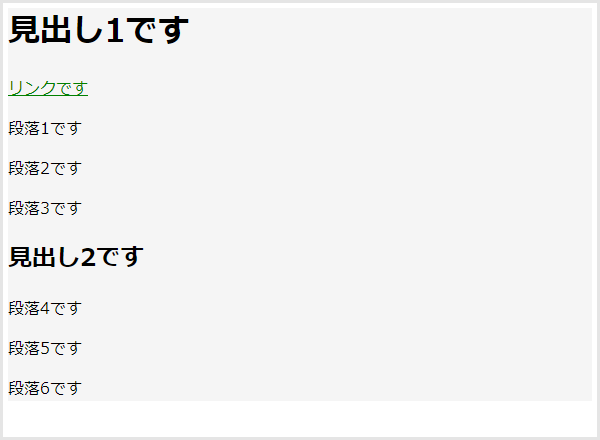
color: red;

}

この場合にスタイルが適用されるのは、h1要素から数えて2番目にある「段落1です」というp要素です。

このように、:nth-child(n)において「n」を指定するときは、親要素内にある**すべての要素**を数に入れます。

では実際にやってみましょう。現状では、pseudo-classes.htmlをブラウザで開くと以下のように表示されます。



ここで、:nth-child(n)を使って「段落2です」というテキストを赤色にしてみましょう。まずは自分の力でpseudo-classes.cssを編集してみてください。

なおpseudo-classes.htmlでは、親要素であるdiv要素（class="parent"）の中に、以下の順番で子要素が入っています。

1. h1要素
2. a要素
3. p要素（段落1です）
4. p要素（段落2です）
5. p要素（段落3です）
6. h2要素
7. p要素（段落4です）
8. p要素（段落5です）
9. p要素（段落6です）

:nth-child(n)を使って「段落2です」というテキストにスタイルを適用するには、どのようにセレクタを指定すればよいでしょうか。

正解は以下のとおりです。

.parent {

background-color: whitesmoke;

}

a:link {

color: orange;

}

a:visited {

color: green;

}

a:hover {

color: purple;

}

a:active {

color: red;

}

+.parent p:nth-child(4) {

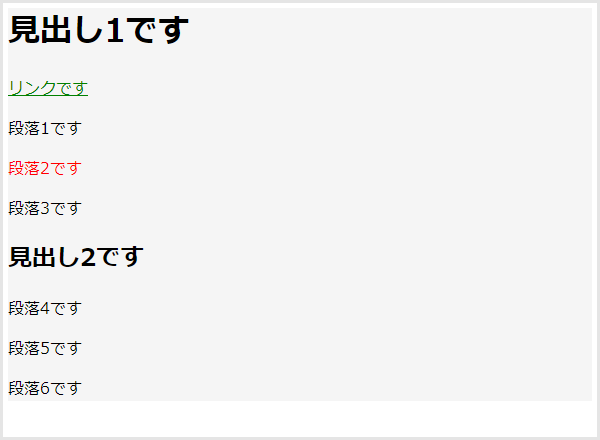
+ color: red;

+}

前述のとおり、:nth-child(n)において「n」を指定するときは、親要素内にある**すべての要素**を数に入れます。p要素（段落2です）はh1要素から数えて4番目にあるため、「n＝4」となります。

pseudo-classes.cssを上記のように編集したあと、pseudo-classes.htmlをブラウザで開いて出力結果を確認してみてください。

以下のように、「段落2です」というテキストが赤色になりました。



### **:nth-of-type(n)**

:nth-of-type(n)は、親要素内にある**同じ要素**の中で、**n番目**の要素にスタイルを適用する疑似クラスです。

例えば以下のようなHTMLファイルがあったとします。親要素であるdiv要素の中に、子要素としてh1要素が1つ、p要素が3つ入っている状態です。

<div>

<h1>見出しです</h1>

<p>段落1です</p>

<p>段落2です</p>

<p>段落3です</p>

</div>

ここで、以下のようにdiv要素内のp:nth-of-type(2)をセレクタに指定したとします。

div p:nth-of-type(2) {

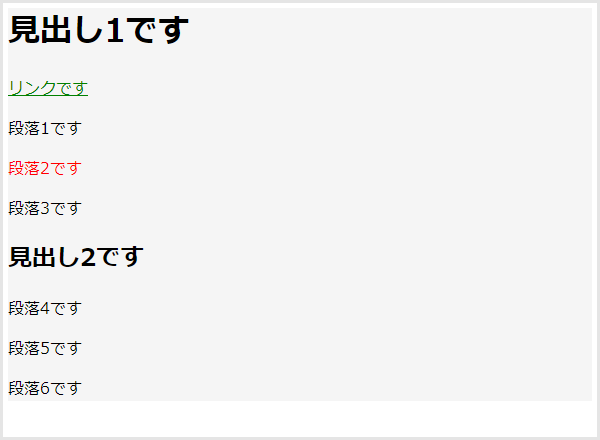
color: blue;

}

この場合にスタイルが適用されるのは、p要素の中で2番目にある「段落2です」というp要素です。

このように、:nth-of-type(n)において「n」を指定するときは、親要素内にある**同じ要素**だけを数に入れます。

では実際にやってみましょう。現状では、pseudo-classes.htmlをブラウザで開くと以下のように表示されます。



ここで、:nth-of-type(n)を使って「段落4です」というテキストを青色にしてみましょう。まずは自分の力でpseudo-classes.cssを編集してみてください。

なおpseudo-classes.htmlでは、親要素であるdiv要素（class="parent"）の中に、以下の順番で子要素が入っています。

1. h1要素（見出し1です）
2. a要素（リンクです）
3. p要素（段落1です）
4. p要素（段落2です）
5. p要素（段落3です）
6. h2要素（見出し2です）
7. p要素（段落4です）
8. p要素（段落5です）
9. p要素（段落6です）

:nth-of-type(n)を使って「段落4です」というテキストにスタイルを適用するには、どのようにセレクタを指定すればよいでしょうか。

正解は以下のとおりです。

.parent {

background-color: whitesmoke;

}

a:link {

color: orange;

}

a:visited {

color: green;

}

a:hover {

color: purple;

}

a:active {

color: red;

}

.parent p:nth-child(4) {

color: red;

}

+.parent p:nth-of-type(4) {

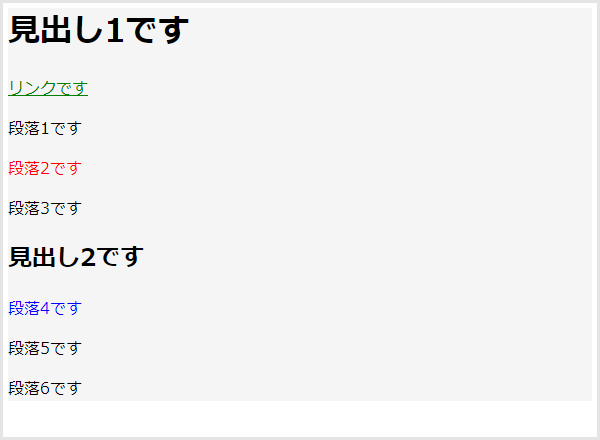
+ color: blue;

+}

前述のとおり、:nth-of-type(n)において「n」を指定するときは、親要素内にある**同じ要素**だけを数に入れます。p要素（段落4です）は同じ要素であるp要素の中で4番目にあるため、「n＝4」となります。

pseudo-classes.cssを上記のように編集したあと、pseudo-classes.htmlをブラウザで開いて出力結果を確認してみてください。

以下のように、「段落4です」というテキストが青色になりました。



### **:not(A)**

:not(A)は、親要素内にあるAという要素**以外に**スタイルを適用する疑似クラスです。

例えば以下のようなHTMLファイルがあったとします。親要素であるdiv要素の中に、子要素としてh1要素が1つ、p要素が3つ入っている状態です。

<div>

<h1>見出しです</h1>

<p>段落1です</p>

<p>段落2です</p>

<p>段落3です</p>

</div>

cssファイル

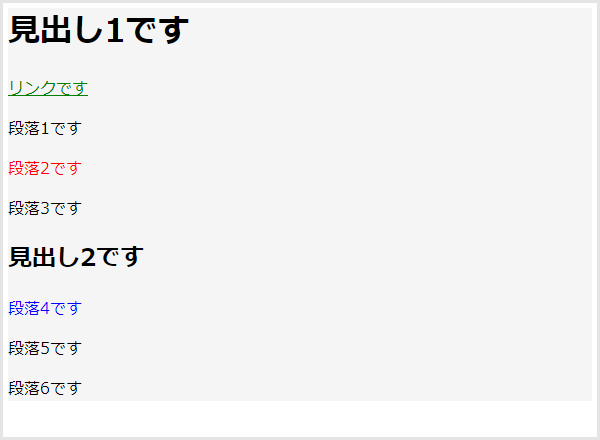
div :not(p) {

color: orange;

}

この場合にスタイルが適用されるのは、親要素であるdiv要素の中においてp要素**以外**、つまりh1要素です。

では実際にやってみましょう。現状では、pseudo-classes.htmlをブラウザで開くと以下のように表示されます。



ここで、p要素以外のh1、a、h2要素に取り消し線を引いてみましょう。

pseudo-classes.cssを以下のように編集したあと、pseudo-classes.htmlをブラウザで開いて出力結果を確認してみてください。

.parent {

background-color: whitesmoke;

}

a:link {

color: orange;

}

a:visited {

color: green;

}

a:hover {

color: purple;

}

a:active {

color: red;

}

.parent p:nth-child(4) {

color: red;

}

.parent p:nth-of-type(4) {

color: blue;

}

+.parent :not(p) {

+ text-decoration: line-through;

+}

以下のように、p要素以外のh1、a、h2要素に取り消し線が引かれました。



まとめ

* **疑似クラス**とは「要素が**特定の状態**のときに適用できるセレクタ」のことである
* 代表的な7つの疑似クラス
  + :link：**未訪問**のリンク（a要素など）にスタイルを適用する
  + :visited：**訪問済み**のリンク（a要素など）にスタイルを適用する
  + :hover：要素の上に**マウスのカーソルが乗っているとき**にスタイルを適用する
  + :active：要素が**アクティブ状態**のときにスタイルを適用する
  + :nth-child(n)：親要素内にある**すべての要素**の中で、**n番目**の要素にスタイルを適用する
  + :nth-of-type(n)：親要素内にある**同じ要素**の中で、**n番目**の要素にスタイルを適用する
  + :not(A)：親要素内にあるAという要素**以外に**スタイルを適用する

## **レスポンシブデザイン**

パソコンとスマートフォンでは画面サイズが異なります。そのため、パソコンではキレイに表示されてもスマートフォンでは見切れてしまったりすることがあります。



そこで重要になるのがレスポンシブデザインです。

レスポンシブデザインとは、パソコンやスマートフォンなど、デバイスの画面幅に応じて**表示が最適化されるデザイン**のことです。

本章でこれから学びますが、レスポンシブデザインに対応させるには以下2つの作業が必要です。

* CSSファイルで**メディアクエリ**を使う
* HTMLファイルで**ビューポート**を設定する

レスポンシブデザインに対応したWebサイトでは、以下のようにデバイスに応じて最適化されたデザインを表示できます。



## **メディアクエリ**

メディアクエリとは、画面幅に応じて**適用するスタイルを切り替えるCSSの機能**のことを指します。

メディアクエリを使えば、「横幅が1024px以上のデバイスだったらこのスタイルを適応してね」と画面サイズごとにあらかじめ命令しておくことで、それぞれの画面サイズに適したデザインを表示させられます。

つまり、メディアクエリを使えばレスポンシブデザインを簡単に実現できるということです。

メディアクエリを使うには、CSSファイル内で以下のように@mediaを記述します。

/\* 以下の例では、画面幅が1024px以下のときにスタイルが適用されます \*/

@media screen and (max-width: 1024px) {

/\* ここに通常のCSSを書きます。以下は一例です \*/

div {

display: inline-block;

}

}

@media screen and (max-width: 1024px)の部分を、以下の2つに分けて解説します。

* @media screen and
* (max-width: 1024px)

### **@media screen and**

このscreenの部分は**メディアタイプ**といって、「どのデバイスにCSSを適用させるか」を指定します。Webサイトの制作時は基本的にscreen（PC、タブレット、スマートフォン）しか指定しないので、@media screen andの部分はコピペでOKです。

### **(max-width: 1024px)**

この部分には**メディア特性**（画面幅など）を指定します。

つまり、「**横幅が1024px以上だったら**このスタイルを適応してね」「**横幅が767px以下だったら**このスタイルを適応してね」という**条件**を指定する部分です。

@media screen andの部分は基本的に固定なので、メディアクエリの書き方としては以下の2パターンを覚えればOKです。

* @media screen and (min-width: ○○px)：画面幅が○○px**以上**のとき
* @media screen and (max-width: ○○px)：画面幅が○○px**以下**のとき

### **補足：ブレークポイント**

レスポンシブデザインにおけるブレークポイントとは、**デザインが切り替わる画面幅**のことです。

先ほどのCSSファイル（見本）でいうと、1024pxがブレークポイントです。よって、1024pxの画面幅を境にしてデザインが切り替わります。

ブレークポイントは複数指定できるので、例えば768pxと1024pxをブレークポイントにすれば、「0～768px」「768～1024px」「1024px～」でデザインを切り替えることもできます。

## **21.4 メディアクエリを使ってみよう**

では実際にメディアクエリを使ってレスポンシブデザインを体験してみましょう。

今回は以下のレスポンシブデザインに挑戦します。

* 768pxをブレークポイントにする
* ブレークポイントより画面幅が広いときはdiv要素を横並びにし、ブレークポイントより画面幅が狭いときは縦並びにする

まずはメディアクエリを使わない場合だとどうなるか、ブラウザの画面を確認しましょう。

HTML\_CSSフォルダ内に新しくmedia-queries.htmlとmedia-queries.cssという2つのファイルを作り、以下のように編集してください。

media-queries.html

+<!DOCTYPE html>

+<html>

+ <head>

+ <title>タイトルです</title>

+ <meta charset="utf-8">

+ <link rel="stylesheet" href="media-queries.css">

+ </head>

+ <body>

+ <div></div>

+ <div></div>

+ <div></div>

+ <div></div>

+ </body>

+</html>

media-queries.css

+div {

+ display: inline-block;

+ width: 100px;

+ height: 100px;

+ margin: 10px;

+ background-color: orange;

+}

上記のコードでは、display: inline-block;を指定して4つのdiv要素を横並びにしています。

media-queries.htmlをブラウザで開き、画面幅を768pxにしてみてください。デベロッパーツールを使えば画面幅の値を確認しながら調整できます。



では次に、メディアクエリを使ってみましょう。media-queries.cssを以下のように編集してください。

div {

display: inline-block;

width: 100px;

height: 100px;

margin: 10px;

background-color: orange;

}

+@media screen and (max-width: 768px) {

+ div {

+ display: block;

+ }

+}

上記のコードでは画面幅の最大が768px、つまり画面幅が768px以下のときにdisplay: block;を指定してdiv要素を縦並びにしています。

media-queries.htmlをブラウザで開いたら、先ほどと同じようにデベロッパーツールを使って画面幅を768pxにしてみましょう。

以下のように、画面幅が768pxのときにdiv要素が縦並びになりました。



画面幅を広くしたり狭くしたりして、768pxがブレークポイントになっていることを確認してください。

## **ビューポート**

ビューポートとは、**Webページが表示されている領域**のことです。Webページが表示されている領域なので、ブラウザのアドレスバーやタブなどはビューポートに**含まれません**。

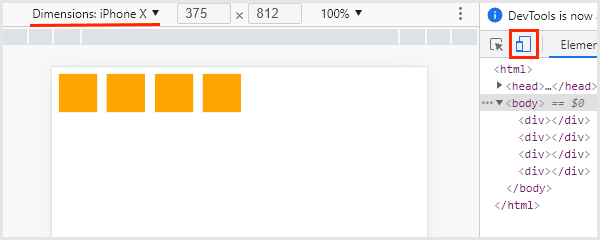


レスポンシブデザインに対応させるためにはビューポートの設定が必須です。

ビューポートを設定しなかった場合、PC向けのページをスマートフォンで表示すると要素が縮小されて見づらくなったり、メディアクエリが効かなくなったりします。

実際に確かめてみましょう。先ほどのmedia-queries.htmlをもう一度ブラウザで開き、デベロッパーツールを起動してください。

続いて以下の赤枠のアイコンをクリックしたあと下線部をクリックし、どれでもよいのでiPhoneを選択してください。こうすることで、PC上でiPhoneの画面表示を確認できます。



3節でメディアクエリを使い、画面幅が768px以下のときにdiv要素が縦並びになるように指定したのを覚えているでしょうか。

しかし、iPhoneの画面表示を確認すると、画面幅が768px以下であるにもかかわらずdiv要素が縦並びになっていません。つまり、メディアクエリが効いていないということです。

3節で確認したのはPC画面なので、メディアクエリを使っただけでdiv要素が縦並びになりました。一方で、スマートフォンで確実にメディアクエリを効かせるにはビューポートの設定も必要です。

ビューポートの設定はHTMLのmeta要素で行います。

ではここで、ビューポートを設定してみましょう。media-queries.htmlを以下のように編集してください。

media-queries.html

<!DOCTYPE html>

<html>

<head>

<title>タイトルです</title>

<meta charset="utf-8">

+ <meta name="viewport" content="width=device-width, initial-scale=1.0">

<link rel="stylesheet" href="media-queries.css">

</head>

<body>

<div></div>

<div></div>

<div></div>

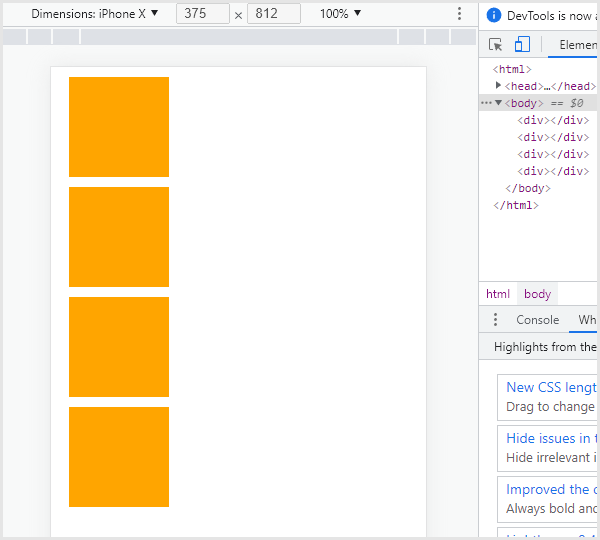
<div></div>

</body>

</html>

ビューポートにはさまざまな設定方法がありますが、基本的には上記のように<meta name="viewport" content="width=device-width, initial-scale=1.0">と書けばOKです。

もう一度media-queries.htmlをブラウザで開き、iPhoneの画面表示を確認してみましょう。以下のように、きちんとメディアクエリが効いてdiv要素が縦並びになっています。



メディアクエリを使うときは同時にビューポートも設定するようにしましょう。

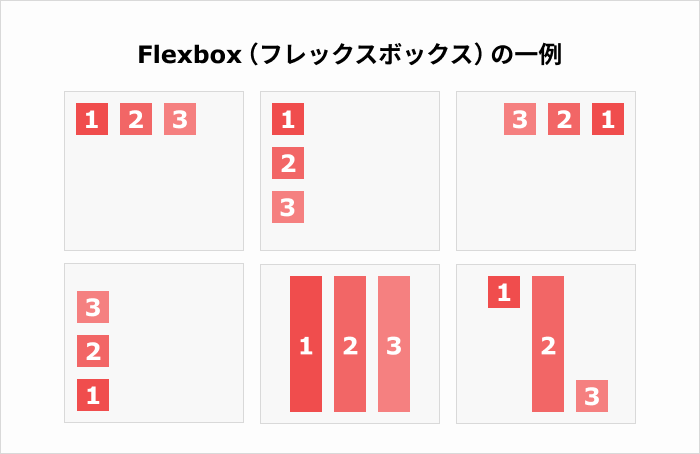
覚える必要はありませんが、参考としてビューポートの設定項目（プロパティ）一覧を掲載しておきます。目を通しておいてください。

| **設定項目（プロパティ）** | **指定できる値** |
| --- | --- |
| width（横幅） | device-width（デバイスの横幅）または数値 |
| height（高さ） | device-height（デバイスの高さ）または数値 |
| initial-scale（ズームの初期倍率） | 数値 |
| minimum-scale（ズームの最小倍率） | 数値 |
| maximum-scale（ズームの最大倍率） | 数値 |
| user-scalable（ズームできるかどうか） | yesまたはno |

まとめ

* レスポンシブデザインとは、デバイスの画面幅に応じて**表示が最適化されるデザイン**のことである
* メディアクエリとは、画面幅に応じて**適用するスタイルを切り替えるCSSの機能**のことである
  + メディアクエリを使うにはCSSに@mediaを記述する
  + 例：@media screen and (max-width: 1024px) { /\* 通常のCSS \*/ }
  + min-width: ○○px：画面幅が○○px**以上**のとき
  + max-width: ○○px：画面幅が○○px**以下**のとき
* レスポンシブデザインにおけるブレークポイントとは、**デザインが切り替わる画面幅**のことである
* ビューポートとは、**Webページが表示されている領域**のことである
  + ビューポートはHTMLのmeta要素で設定できる
  + 例：<meta name="viewport" content="width=device-width, initial-scale=1.0">
  + レスポンシブデザインを実現するにはビューポートの設定が必須である
  + メディアクエリを使うときは同時にビューポートも設定する

## **Flexbox（フレックスボックス）**



Flexbox（フレックスボックス）とは、**柔軟にレイアウトを組める**CSSのレイアウト手法のことです。なお、flexはflexibleの略で、「柔軟な」という意味です。

「Flexboxを利用すれば簡単にレイアウトを組める」と覚えておきましょう。

レスポンシブデザインを作るときに大変なのが、「PCとスマートフォンそれぞれに最適なレイアウトを考えて調整しなければならない」という点です。

例えば「PCでは横3列のレイアウトにしたいけどスマートフォンでは横1列にしたい」というとき、自力でこれを実現するのは大変です。しかし、Flexboxを利用すればこのようなレスポンシブデザインを簡単に実現できます。

## **22.3 Flexboxの使い方**

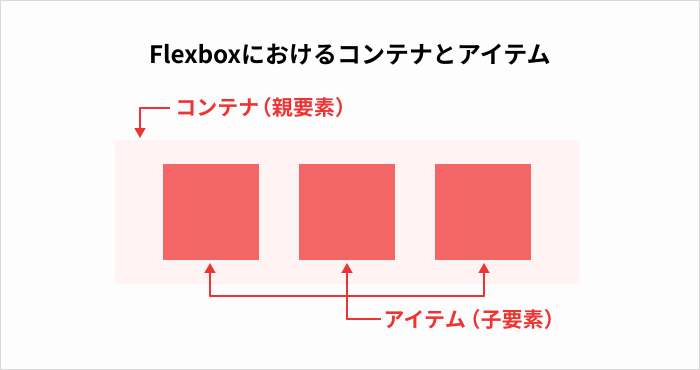
Flexboxの基本的な使い方は以下のとおりです。

1. HTMLファイルでコンテナ（親要素）とアイテム（子要素）を作成する
2. CSSファイルでコンテナにdisplay: flex;を指定する
3. CSSファイルでコンテナとアイテムそれぞれに各種プロパティを指定して、レイアウトを調整する

ここで、Flexboxにおけるコンテナとアイテムという概念について解説します。簡単にいえば **「コンテナ＝親要素」「アイテム＝子要素」** です。

Flexboxを利用するときは親要素を入れ物にたとえてコンテナと呼び、その中に入っている子要素をアイテムと呼びます。なお、アイテムは「項目」や「品物」を意味します。

以下のように、入れ物であるコンテナの中にアイテムが入っているイメージです。コンテナの中に入っているアイテムは自由自在にレイアウトを組めます。



ではまず、1と2でFlexboxを利用する準備を整えましょう。HTMLファイルを編集し、以下2つの要素をセットで作成します。

* コンテナ（親要素）
* アイテム（子要素）

実際に作成してみましょう。HTML\_CSSフォルダ内に新しくflexbox.htmlというファイルを作成し、flexbox.htmlを以下のように編集してください。

flexbox.html

+<!DOCTYPE html>

+<html>

+ <head>

+ <title>タイトルです</title>

+ <meta charset="utf-8">

+ <link rel="stylesheet" href="flexbox.css">

+ </head>

+ <body>

+ <div class="flexbox-container">

+ <div id="item01" class="flexbox-item">

+ <p>アイテム01<br>（子要素）</p>

+ </div>

+ <div id="item02" class="flexbox-item">

+ <p>アイテム02<br>（子要素）</p>

+ </div>

+ <div id="item03" class="flexbox-item">

+ <p>アイテム03<br>（子要素）</p>

+ </div>

+ <div id="item04" class="flexbox-item">

+ <p>アイテム04<br>（子要素）</p>

+ </div>

+ <div id="item05" class="flexbox-item">

+ <p>アイテム05<br>（子要素）</p>

+ </div>

+ <div id="item06" class="flexbox-item">

+ <p>アイテム06<br>（子要素）</p>

+ </div>

+ <div id="item07" class="flexbox-item">

+ <p>アイテム07<br>（子要素）</p>

+ </div>

+ <div id="item08" class="flexbox-item">

+ <p>アイテム08<br>（子要素）</p>

+ </div>

+ </div>

+ </body>

+</html>

上記のコードでは、Flexboxを利用するために以下の要素を作成しました。

* コンテナ（親要素）：1つのdiv要素（class="flexbox-container"）
* アイテム（子要素）：8つのdiv要素（class="flexbox-item"）

続いてCSSファイルで、コンテナにdisplay: flex;を指定します。

なお、displayプロパティに指定できる値については、[17章](https://terakoya.sejuku.net/programs/51/chapters/592)でblock、inline、inline-block、noneの4つを学びました。しかし、これら代表的な4つ以外にもdisplayプロパティに指定できる値はたくさんあります（すべて覚える必要はありません）。

その中の1つがflexで、コンテナにdisplay: flex;を指定するとFlexboxが利用できるようになります。

ではまずdisplay: flex;を指定しない場合、つまりFlexboxを利用しない場合の表示を確認しておきましょう。

HTML\_CSSフォルダ内に新しくflexbox.cssというファイルを作成し、以下のように編集してください。

flexbox.css

+.flexbox-container {

+ padding: 5px;

+ background-color: lightcyan;

+}

+.flexbox-item {

+ padding: 5px;

+ margin: 5px;

+ background-color: orange;

+}

上記のコードでは、要素を見やすくするために背景色を指定したり、paddingプロパティやmarginプロパティを使って余白を設けたりしています。

ではflexbox.htmlをブラウザで開き、出力結果を確認してみましょう。

[17章](https://terakoya.sejuku.net/programs/51/chapters/592)で学んだとおり、div要素のdisplayプロパティの初期値はblockなので、以下のように要素が縦に並んでいます。



続いて、コンテナにdisplay: flex;を指定した場合の表示を確認しましょう。

flexbox.cssを以下のように編集したあと、flexbox.htmlをブラウザで開いて出力結果を確認してください。

flexbox.css

.flexbox-container {

+ display: flex;

padding: 5px;

background-color: lightcyan;

}

.flexbox-item {

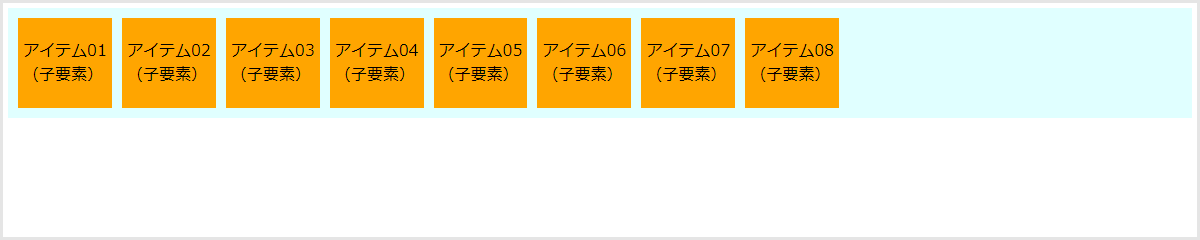
padding: 5px;

margin: 5px;

background-color: orange;

}

以下のように、アイテムが横並びになりました。



ここからコンテナやアイテムに各種プロパティを指定していくことで、柔軟にレイアウトを組むことができます。

### **補足：アイテムが横並びになる理由**

コンテナにdisplay: flex;を指定すると、アイテムが横並びになる理由は以下のとおりです。

* コンテナにdisplay: flex;を指定した時点でFlexboxが有効になる
* Flexboxのflex-directionプロパティの初期値がrowなので、指定しなくてもアイテムが左から右に配置される

flex-directionプロパティについては後ほど詳しく解説します。

## **22.4 コンテナ（親要素）に指定できるプロパティ**

コンテナにdisplay: flex;を指定したので、Flexboxを利用する準備は完了です。

次はコンテナとアイテムそれぞれに各種プロパティを指定していくのですが、まずはコンテナに指定できるプロパティを見ていきましょう。

コンテナにFlexboxの各種プロパティを指定することで、その子要素であるアイテムすべてにスタイルが適用されます。

コンテナに指定できるプロパティは以下のとおりです。

* flex-directionプロパティ：アイテムの**並び順**を指定
* flex-wrapプロパティ：アイテムの**折り返し**を指定
* flex-flowプロパティ：アイテムの**並び順と折り返し**を一括で指定
* justify-contentプロパティ：アイテムの**横方向（主軸）の揃え方**を指定
* align-itemsプロパティ：アイテムの**縦方向（交差軸）の揃え方**を指定
* align-contentプロパティ：アイテムを**複数行にしたときの縦方向（交差軸）の揃え方**を指定

なおこれらのプロパティを暗記する必要はありません。

本章は「Flexboxを利用すればこんなことができる」を知ることが目標です。

Webサイトの制作中に必要になったら「そういえばこんなプロパティがあったな」と調べてコピー&ペーストすればOKです。

では順番に見ていきましょう。

なお、本章では表示の変化を確認するために画面幅を調整する必要が出てきます。その際は直接ブラウザの幅を変えるのではなく、デベロッパーツールを使って画面幅を調整してください。

### **flex-directionプロパティ：アイテムの並び順を指定**

flex-directionは、アイテムの**並び順**を指定するプロパティです。

flex-directionプロパティに指定できる値は以下のとおりです。

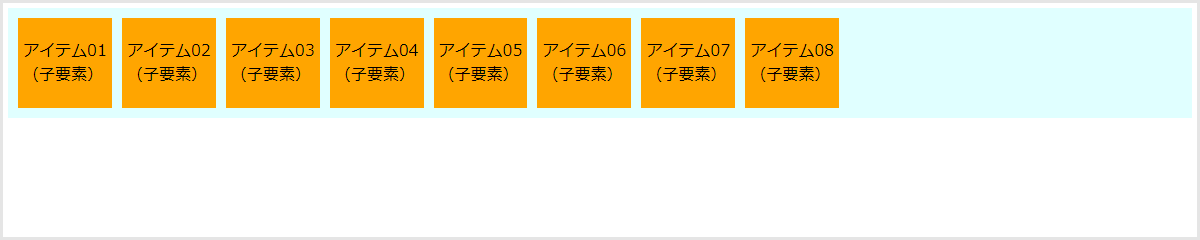
* row（初期値）：アイテムを左から右に配置する
* row-reverse：アイテムを右から左に配置する
* column：アイテムを上から下に配置する
* column-reverse：アイテムを下から上に配置する

なお、flex-directionプロパティで指定した値によって、主軸が**横方向**になるか**縦方向**になるか決まります。

主軸とはアイテムを配置していく方向のことで、row（初期値）の場合はアイテムが左から右に配置されるため、主軸は**横方向**に決定されます。

主軸が**横方向**の場合、主軸に対して交差する方向は**縦方向**となりますが、これを交差軸と呼びます。

では実際にやってみましょう。現時点では以下のように、アイテムが左から順番に並んでいます。



ではflex-direction: row-reverse;を指定して、これらのアイテムを右から順番に並べてみましょう。

flexbox.cssを以下のように編集したあと、flexbox.htmlをブラウザで開いて出力結果を確認してください。

flexbox.css

.flexbox-container {

display: flex;

+ flex-direction: row-reverse;

padding: 5px;

background-color: lightcyan;

}

.flexbox-item {

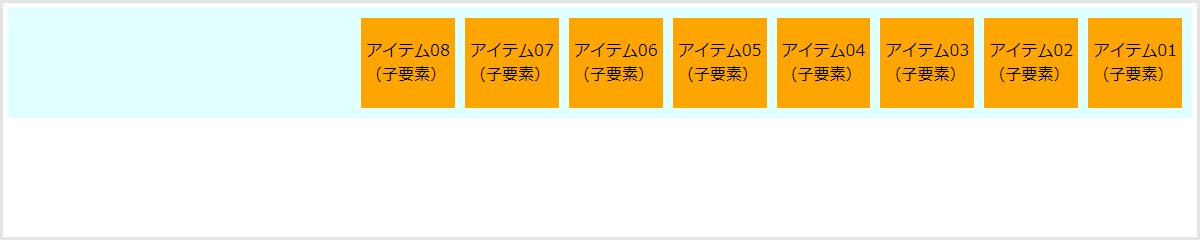
padding: 5px;

margin: 5px;

background-color: orange;

}

以下のように、アイテムの並び順が右から左になりました。



### **flex-wrapプロパティ：アイテムの折り返しを指定**

flex-wrapは、アイテムの**折り返し**を指定するプロパティです。

* 折り返し＝幅が足りないときにはみ出すのではなく改行すること

flex-wrapプロパティに指定できる値は以下のとおりです。

* nowrap（初期値）：アイテムを折り返しせず、1行に並べる
* wrap：アイテムを折り返し、上から下に並べる
* wrap-reverse：アイテムを折り返し、下から上に並べる

では実際にやってみましょう。現時点では以下のように、画面幅を狭めてもアイテムが折り返されず、1行に並んでいます。



ではflex-wrap: wrap;を指定して、アイテムが折り返されるようにしてみましょう。

flexbox.cssを以下のように編集したあと、flexbox.htmlをブラウザで開いて出力結果を確認してください。

flexbox.css

.flexbox-container {

display: flex;

- flex-direction: row-reverse;

+ flex-wrap: wrap;

padding: 5px;

background-color: lightcyan;

}

.flexbox-item {

padding: 5px;

margin: 5px;

background-color: orange;

}

画面幅を狭めると、以下のようにアイテムが折り返されるようになりました。



### **flex-flowプロパティ：アイテムの並び順と折り返しを一括で指定**

flex-flowは、アイテムの**並び順と折り返し**を一括で指定するプロパティです。

flex-flowプロパティに指定できる値はflex-directionプロパティやflex-wrapプロパティと同じですが、以下のように半角スペースを空けて一括で指定します。なお、flex-flowプロパティの初期値はrow nowrapです。

CSSファイル（見本）

.flexbox-container {

display: flex;

/\* 「flex-directionプロパティの値 flex-wrapプロパティの値」の順で一括指定します \*/

flex-flow: row wrap;

}

すでに学んだ内容なので実践は割愛しますが、ぜひ自分で値を変えながら試してみてください。

### **justify-contentプロパティ：アイテムの横方向（主軸）の揃え方を指定**

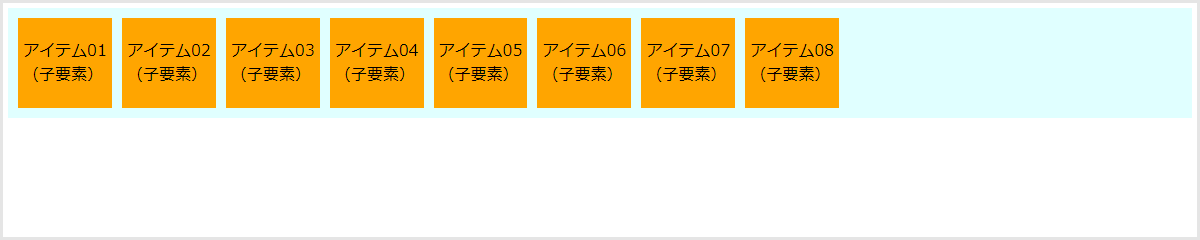
justify-contentは、アイテムの**横方向（主軸）の揃え方**を指定するプロパティです。

* justify＝（行間・字間を）整える、（文章・行などの両端を）揃える　（※覚える必要はありません）

justify-contentプロパティに指定できる値は以下のとおりです。

* flex-start（初期値）：アイテムを左揃えで配置する
* flex-end：アイテムを右揃えで配置する
* center：アイテムを中央揃えで配置する
* space-between：一番左と一番右のアイテムを両端に配置し、残りのアイテムを均等な間隔で配置する
* space-around：一番左と一番右のアイテムも含め、均等な間隔で配置する

では実際にやってみましょう。現時点では以下のように、アイテムが左揃えで配置されています



ではjustify-content: center;を指定して、アイテムを中央揃えで配置してみましょう。

flexbox.cssを以下のように編集したあと、flexbox.htmlをブラウザで開いて出力結果を確認してください。

flexbox.css

.flexbox-container {

display: flex;

+ justify-content: center;

padding: 5px;

background-color: lightcyan;

}

.flexbox-item {

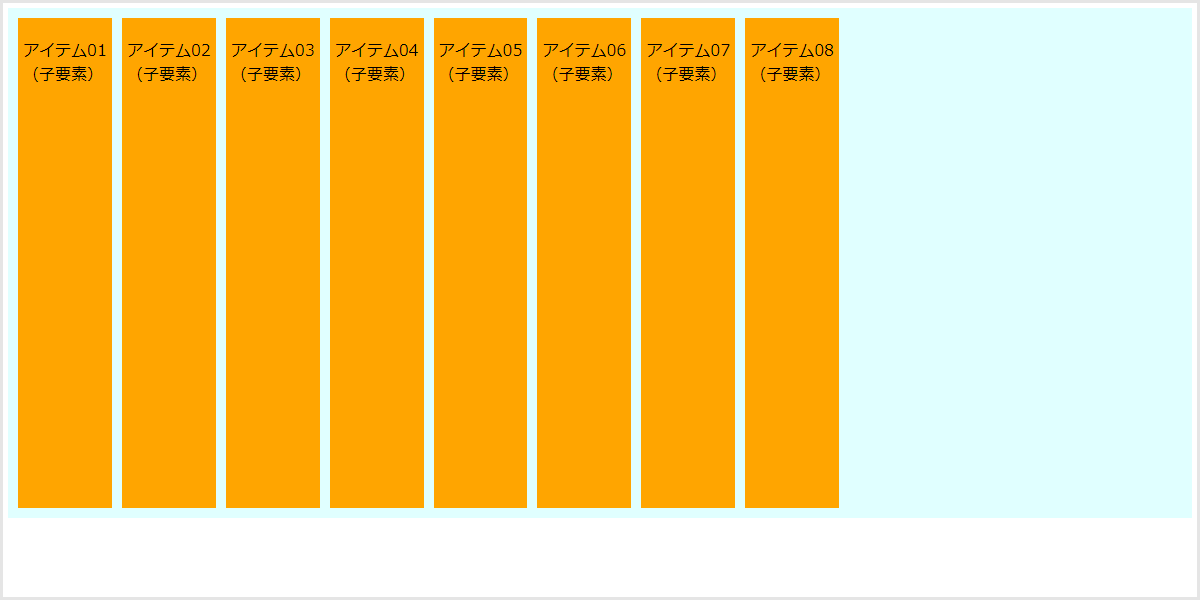
padding: 5px;

margin: 5px;

background-color: orange;

}

align-itemsプロパティの初期値はstrechなので、以下のようにアイテムがコンテナの高さいっぱいに広がっています。



ではalign-items: flex-end;を指定して、アイテムを下揃えで配置してみましょう。

flexbox.cssを以下のように編集したあと、flexbox.htmlをブラウザで開いて出力結果を確認してください。

flexbox.css

.flexbox-container {

display: flex;

+ align-items: flex-end;

height: 500px;

padding: 5px;

background-color: lightcyan;

}

.flexbox-item {

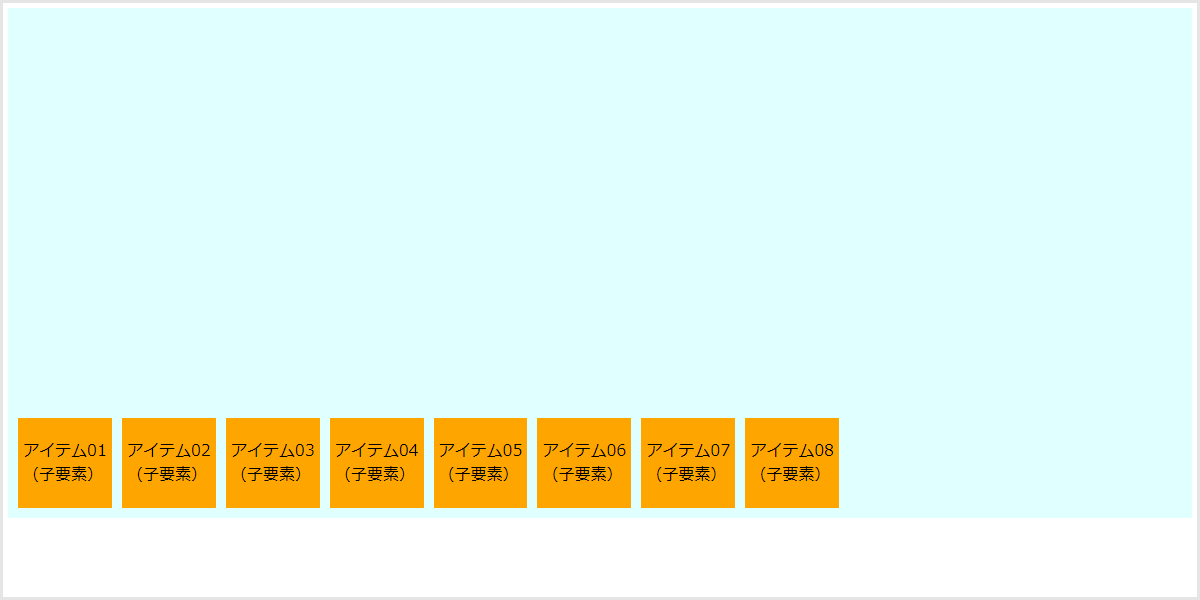
padding: 5px;

margin: 5px;

background-color: orange;

}

以下のように、アイテムが下揃えで配置されました。

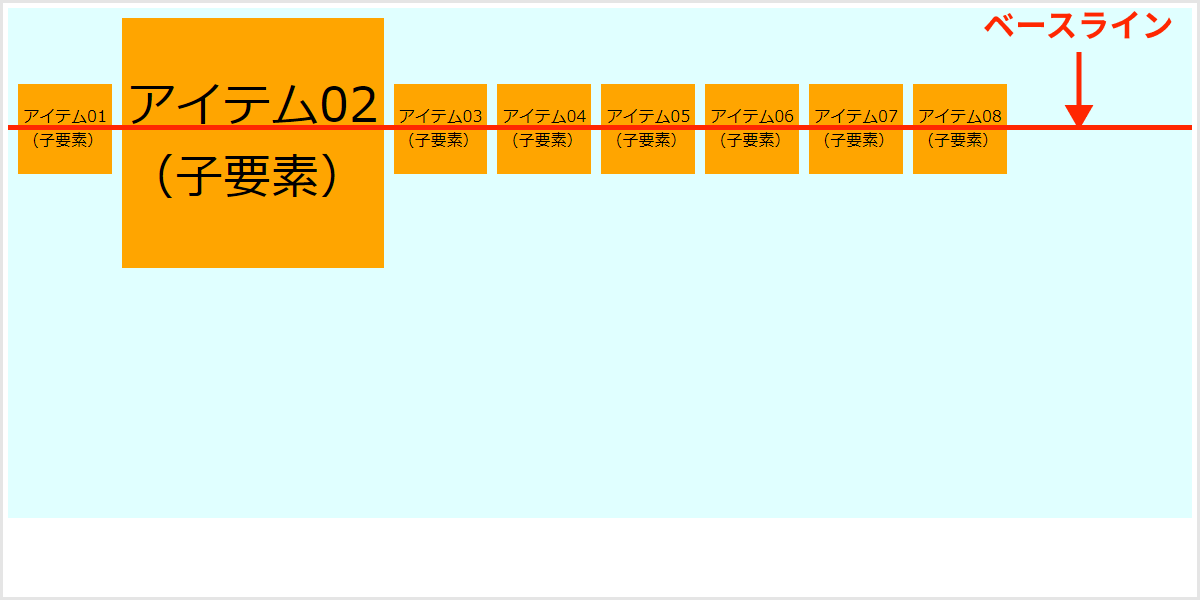


#### **補足：align-items: baseline;**

現在編集しているflexbox.htmlとflexbox.cssでは、align-items: baseline;を指定してもalign-items: flex-start;と同じように上揃えで表示されます。

これは、アイテム内の文字の内容や大きさが同じ、つまり文字のベースラインが同じだからです。

ここで、例えば2つ目のアイテムの文字を大きくすると、文字のベースラインが変わるので以下のように表示されます。



### **align-contentプロパティ：アイテムを複数行にしたときの縦方向（交差軸）の揃え方を指定**

align-contentは、アイテムを**複数行にしたときの縦方向（交差軸）の揃え方**を指定するプロパティです。

align-contentプロパティに指定できる値は以下のとおりです。

* stretch（初期値）：アイテムの行をコンテナの高さに合わせて広げて配置する
* flex-start：アイテムの行を上揃えで配置する
* flex-end：アイテムの行を下揃えで配置する
* center：アイテムの行を中央揃えで配置する
* space-between：一番上と一番下のアイテムを上下の端に配置し、残りのアイテムを均等な間隔で配置する
* space-around：一番上と一番下のアイテムも含め、均等な間隔で配置する

では実際にやってみましょう。まずは初期値であるstretchの表示を確認するため、flexbox.cssを以下のように編集してください。

flexbox.css

.flexbox-container {

display: flex;

+ flex-wrap: wrap;

height: 500px;

padding: 5px;

background-color: lightcyan;

}

.flexbox-item {

padding: 5px;

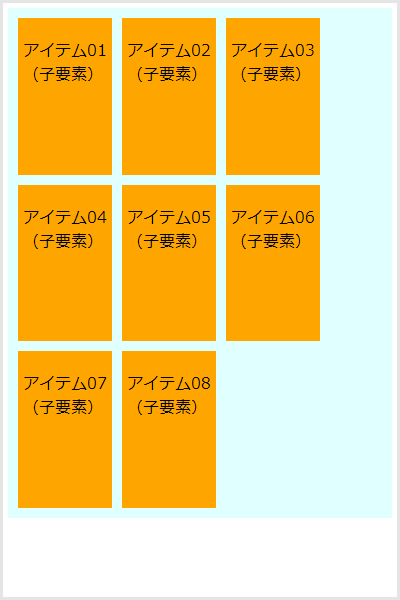
margin: 5px;

background-color: orange;

}

上記のコードでは、ブラウザを狭めてアイテムを複数行にするために、flex-wrap: wrap;を指定しています。

ではflexbox.htmlをブラウザで開き、出力結果を確認してください。画面幅を狭めて複数行にすると、以下のようにアイテムの行がコンテナの高さに合わせて広がります。



ではalign-content: space-between;を指定して、どのように表示されるか確認してみましょう。

flexbox.cssを以下のように編集したあと、flexbox.htmlをブラウザで開いて出力結果を確認してください。

flexbox.css

.flexbox-container {

display: flex;

flex-wrap: wrap;

+ align-content: space-between;

height: 500px;

padding: 5px;

background-color: lightcyan;

}

.flexbox-item {

padding: 5px;

margin: 5px;

background-color: orange;

}

画面幅を狭めて複数行にすると、以下のように一番上と一番下のアイテムが上下の端に配置され、残りのアイテムは均等な間隔で配置されます。



## **アイテム（子要素）に指定できるプロパティ**

これまで学んできたのはコンテナ（親要素）に指定できるプロパティなので、そのコンテナ内にあるすべてのアイテムにスタイルが適用されました。

一方で、これから学ぶのはアイテム（子要素）に指定できるプロパティなので、アイテム一つ一つに個別のスタイルを適用させられます。

アイテムに指定できるプロパティは以下のとおりです。

* orderプロパティ：アイテムの**並び順**を指定
* flex-growプロパティ：アイテムの伸長係数（**どれくらい伸びるか**）を指定
* flex-shrinkプロパティ：アイテムの縮小係数（**どれくらい縮むか**）を指定
* flex-basisプロパティ：アイテムの**幅の初期サイズ**を指定
* flexプロパティ：アイテムの**伸長係数**、**縮小係数**、**幅の初期サイズ**を一括で指定
* align-selfプロパティ：アイテムの**縦方向（交差軸）の揃え方**を指定

コンテナに指定できるプロパティと同様、これらも暗記する必要はありません。「コンテナに指定できるプロパティとアイテムに指定できるプロパティの2種類がある」という点を押さえておくことが大切です。

では順番に見ていきましょう。

### **orderプロパティ：アイテムの並び順を指定**

orderは、アイテムの**並び順**を指定するプロパティです。

コンテナに指定できるflex-directionプロパティでは、「左から右」「上から下」など、大まかな並び順の指定しかできませんでした。

しかし、orderプロパティを使ってそれぞれのアイテムに具体的な数値を指定すれば、より細かく並び順を決められます。

では実際にやってみましょう。flexbox.cssを以下のように編集したあと、flexbox.htmlをブラウザで開いて出力結果を確認してください。

flexbox.css

.flexbox-container {

display: flex;

padding: 5px;

background-color: lightcyan;

}

.flexbox-item {

padding: 5px;

margin: 5px;

background-color: orange;

}

+ #item01 {

+ order: 5;

+ }

+ #item02 {

+ order: 6;

+ }

+ #item03 {

+ order: 7;

+ }

+ #item04 {

+ order: 8;

+ }

+ #item05 {

+ order: 1;

+ }

+ #item06 {

+ order: 2;

+ }

+ #item07 {

+ order: 3;

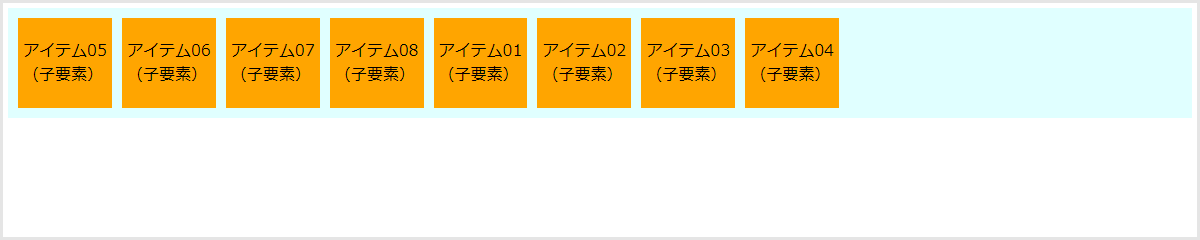
+ }

+ #item08 {

+ order: 4;

+ }

以下のように、指定したとおりの順番でアイテムが配置されました。



### **lex-growプロパティ：アイテムの伸長係数（どれくらい伸びるか）を指定**

flex-growは、アイテムの伸長係数（**どれくらい伸びるか**）を指定するプロパティです。

flex-growプロパティには数値を指定できます。初期値は0です。

では実際にやってみましょう。flexbox.cssを以下のように編集したあと、flexbox.htmlをブラウザで開いて出力結果を確認してください。

flexbox.css

.flexbox-container {

display: flex;

padding: 5px;

background-color: lightcyan;

}

.flexbox-item {

padding: 5px;

margin: 5px;

background-color: orange;

}

#item01 {

}

#item02 {

}

#item03 {

}

#item04 {

}

#item05 {

}

#item06 {

}

#item07 {

+ flex-grow: 1;

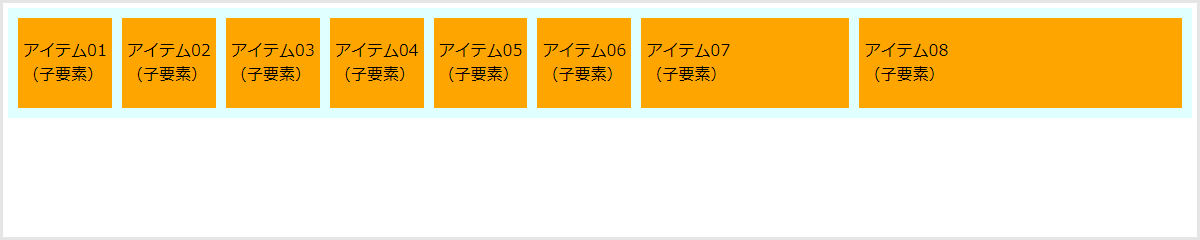
}

#item08 {

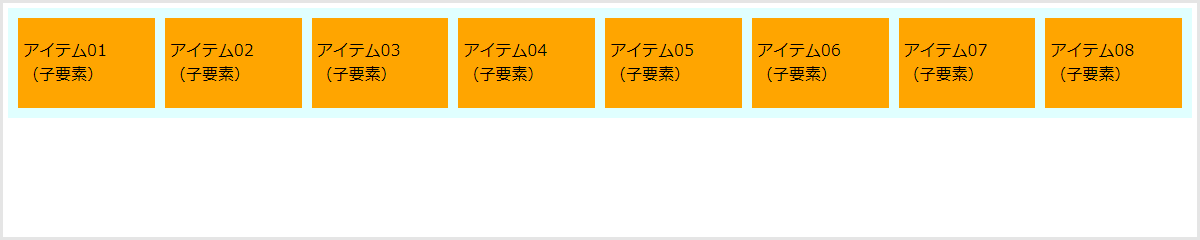
+ flex-grow: 2;

}

以下のように、アイテム07とアイテム08の幅が1：2の比率で伸びました。



なお、すべてのアイテムにflex-grow: 1;を指定すれば、それぞれのアイテムの幅が均等に伸びます。ぜひ試してみてください。



### **flex-shrinkプロパティ：アイテムの縮小係数（どれくらい縮むか）を指定**

flex-shrinkはflex-growプロパティとは逆に、アイテムの縮小係数（**どれくらい縮むか**）を指定するプロパティです。

flex-growプロパティと同じように数値を指定しますが、flex-shrinkプロパティの初期値は1です。

では実際にやってみましょう。flexbox.cssを以下のように編集したあと、flexbox.htmlをブラウザで開いて出力結果を確認してください。

flexbox.css

.flexbox-container {

display: flex;

padding: 5px;

background-color: lightcyan;

}

.flexbox-item {

padding: 5px;

margin: 5px;

background-color: orange;

}

#item01 {

}

#item02 {

}

#item03 {

}

#item04 {

}

#item05 {

}

#item06 {

}

#item07 {

}

#item08 {

+ flex-shrink: 2;

}

画面幅を狭めると、以下のようにアイテム08の幅がより大きく縮むようになりました。



### **flex-basisプロパティ：アイテムの幅の初期サイズを指定**

flex-basisは、アイテムの**幅の初期サイズ**を指定するプロパティです。

flex-basisプロパティの値には、widthプロパティと同じようにpxや%などを指定できます。なお、初期値はwidthプロパティと同じくautoです。

では実際にやってみましょう。flexbox.cssを以下のように編集したあと、flexbox.htmlをブラウザで開いて出力結果を確認してください。

flexbox.css

.flexbox-container {

display: flex;

padding: 5px;

background-color: lightcyan;

}

.flexbox-item {

padding: 5px;

margin: 5px;

background-color: orange;

}

#item01 {

+ flex-basis: 150px;

}

#item02 {

+ flex-basis: 25%;

}

#item03 {

}

#item04 {

}

#item05 {

}

#item06 {

}

#item07 {

}

#item08 {

}

以下のように、アイテム01とアイテム02の幅の初期サイズが指定したとおりになりました。



### **flexプロパティ：アイテムの伸長係数、縮小係数、ベースとなる幅を一括で指定**

flexはアイテムの**伸長係数**、**縮小係数**、**幅の初期値**を一括で指定するプロパティです。

flexプロパティに指定できる値はflex-grow、flex-shrink、flex-basisプロパティと一緒ですが、以下のように半角スペースを空けて一括で指定します。なお、flexプロパティの初期値は0 1 autoです。

CSSファイル（見本）

.flexbox-item {

/\* 「flex-growの値 flex-shrinkの値 flex-basisの値」の順で一括指定します \*/

flex: 1 1 200px;

}

### **align-selfプロパティ：アイテムの縦方向（交差軸）の揃え方を指定**

align-selfは、アイテムの**縦方向（交差軸）の揃え方**を指定するプロパティです。

4節で学んだalign-itemsプロパティの「個別のアイテムに指定できるバージョン」と考えるとよいでしょう。align-itemsはコンテナに指定するプロパティでしたが、align-selfはアイテムに指定するプロパティです。

align-selfプロパティに指定できる値は以下のとおりです。

* auto（初期値）：コンテナのalign-itemsプロパティの値を継承する
* stretch：アイテムをコンテナの高さいっぱいに広げて配置する
* flex-start：アイテムを上揃えで配置する
* flex-end：アイテムを下揃えで配置する
* center：アイテムを中央揃えで配置する
* baseline：アイテムを文字のベースラインに合わせて配置する

では実際にやってみましょう。flexbox.cssを以下のように編集したあと、flexbox.htmlをブラウザで開いて出力結果を確認してください。

flexbox.css

.flexbox-container {

display: flex;

+ height: 500px;

padding: 5px;

background-color: lightcyan;

}

.flexbox-item {

padding: 5px;

margin: 5px;

background-color: orange;

}

#item01 {

+ align-self: flex-start;

}

#item02 {

}

#item03 {

+ align-self: center;

}

#item04 {

}

#item05 {

+ align-self: flex-end;

}

#item06 {

}

#item07 {

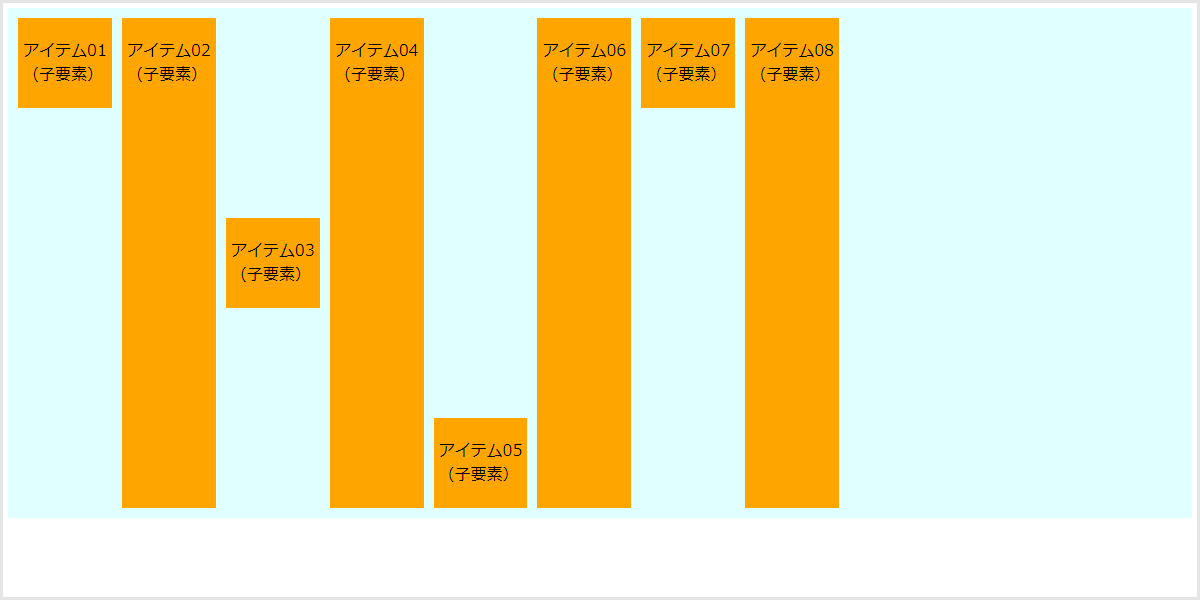
+ align-self: baseline;

}

#item08 {

}

以下のように、アイテムの縦方向の揃え方が指定したとおりになりました。



なお、align-selfプロパティを指定していないアイテムは初期値のautoなので、コンテナのalign-itemsプロパティの値を継承します。

そして、align-itemsプロパティの初期値はstrechなので、それを継承したアイテムはコンテナの高さいっぱいに広がります。

値を自由に変えて、表示の変化を確認してみてください。

まとめ

* Flexboxとは、**柔軟にレイアウトを組める**CSSのレイアウト手法のことである
* Flexboxの基本的な使い方
  + HTMLファイルでコンテナ（親要素）とアイテム（子要素）を作成する
  + CSSファイルでコンテナにdisplay: flex;を指定する
  + CSSファイルでコンテナとアイテムそれぞれに各種プロパティを指定して、レイアウトを調整する
* コンテナ（親要素）に指定できるプロパティは、コンテナ内にあるすべてのアイテムに適用される
* アイテム（子要素）に指定できるプロパティは、それぞれのアイテムに対して個別に適用される

| **コンテナ（親要素）に指定できるプロパティ** | **説明** |
| --- | --- |
| flex-directionプロパティ | アイテムの**並び順**を指定 |
| flex-wrapプロパティ | アイテムの**折り返し**を指定 |
| flex-flowプロパティ | アイテムの**並び順と折り返し**を一括で指定 |
| justify-contentプロパティ | アイテムの**横方向（主軸）の揃え方**を指定 |
| align-itemsプロパティ | アイテムの**縦方向（交差軸）の揃え方**を指定 |
| align-contentプロパティ | アイテムを**複数行にしたときの縦方向（交差軸）の揃え方**を指定 |

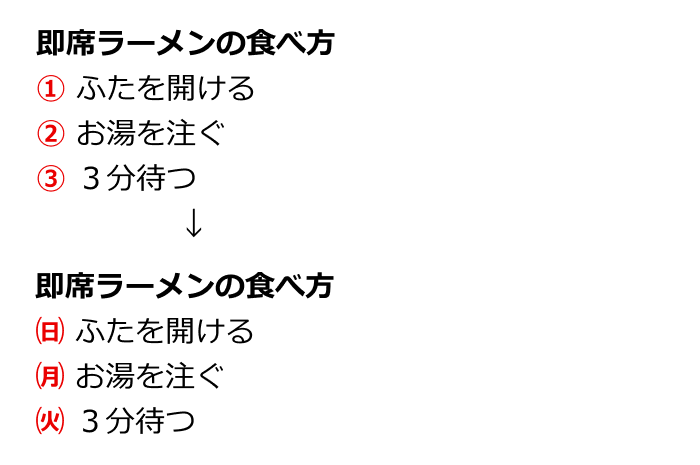
| **アイテム（子要素）に指定できるプロパティ** | **説明** |
| --- | --- |
| orderプロパティ | アイテムの**並び順**を指定 |
| flex-growプロパティ | アイテムの伸長係数（**どれくらい伸びるか**）を指定 |
| flex-shrinkプロパティ | アイテムの縮小係数（**どれくらい縮むか**）を指定 |
| flex-basisプロパティ | アイテムの**幅の初期サイズ**を指定 |
| flexプロパティ | アイテムの**伸長係数**、**縮小係数**、**幅の初期サイズ**を一括で指定 |
| align-selfプロパティ | アイテムの**縦方向（交差軸）の揃え方**を指定 |

## **機種依存文字**

機種依存文字とは、特定の機種のパソコンや端末でしか表示できない文字のことをいいます。

例えば、Windowsでしか表示できない文字をMacで表示しようとすると、文字が表示されなかったり、入力した文字とは異なる文字が表示されてしまうことがあります。

具体的には、丸囲み文字の数字を表示したかったのに、かっこで囲まれた曜日に変わってしまうようなことが起こります。

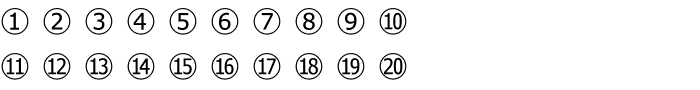


Webサイトは世界中のたくさんの種類のパソコンやスマートフォンからアクセスされます。そのため、機種依存文字は使わないようにしましょう。

ではここからは、代表的な機種依存文字を紹介していきます。

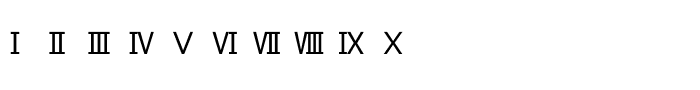
### **丸囲み数字**

丸囲み数字はかっこに置き換えるか、もしくは、後述する特殊文字を使いましょう。



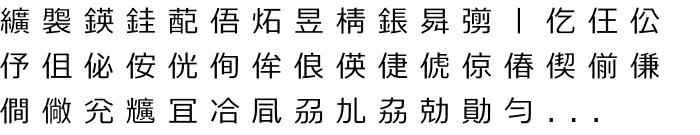
### **ローマ数字**

ローマ数字も機種依存文字なので、アルファベットの「I」「V」「X」の組み合わせか、特殊文字で代用しましょう。



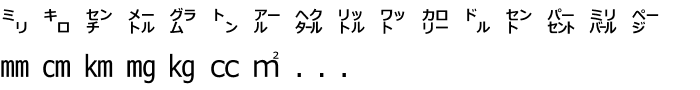
### **旧字体漢字**

旧字体の一部の漢字は機種依存文字です。使うケースは少ないですが、ひらがなにするなどして対応しましょう。



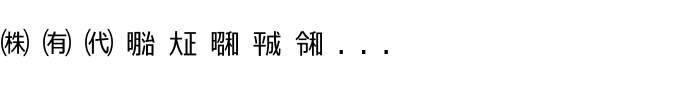
### **単位**

単位の文字を変換すると以下のような小さい文字の表記が出てきます。これらも機種依存文字ですので、全角のカタカナで入力しましょう。



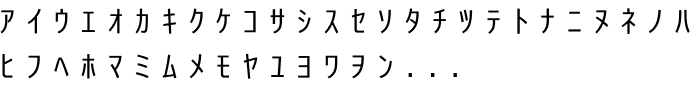
### **（）つき文字、元号**

カッコつきの年号や文字も機種依存文字ですので、全角で入力しましょう。



### **半角カタカナ**

半角のカタカナも機種によって文字化けします。全角のカタカナで入力しましょう。



### **記号**

これらの記号も機種によって使えません。後述する特殊文字で代用できるものがあるので、そちらを代用しましょう。



## **特殊文字（文字実体参照、数値文字参照）**

では機種依存文字を使いたい場合はどうすればいいのでしょうか。その場合は**特殊文字**で代用しましょう。

特殊文字とは、コピーライトや丸囲み文字などの記号のことを指します。機種依存文字の代替として特殊文字が使えるものがあります。

特殊文字をWebサイトで表示するには、**文字参照**という「別の文字や数値に置き換えて表示させる方法」を利用します。

具体的には文字参照には2種類の方法があります。キーワードとなる文字を使う**文字実体参照**と、数字を使う**数値文字参照**です。それぞれ簡単に説明します。

**文字実体参照**は、&で始まり;で終わる&文字;の形で書きます。例えば、登録商標記号（Rを丸で囲んだ記号）は&reg;と書きます。

**数値文字参照**は、&#で始まり;で終わる&#数字;の形で書きます。例えば、登録商標記号は&#174;と書きます。

イメージがまだ掴めていないと思うので、実際にWebサイト制作においてどのように使われるのか、実例を見てみましょう。

### **特殊文字の使い方**

例としてコピーライトの記述方法を紹介します。

**OK：文字実体参照を用いたコピーライトの記述例**

<p>&copy; 2021 SAMURAI.inc</p>

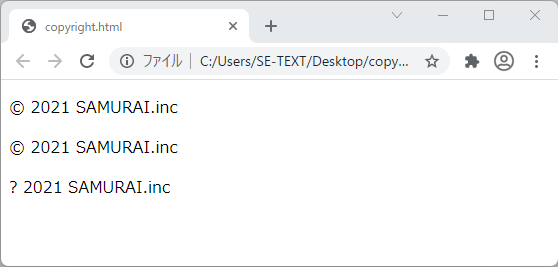
**OK：数値文字参照を用いたコピーライトの記述例**

**<p>&#169; 2021 SAMURAI.inc</p>**

**NG：全角のコピーライト記号を用いた記述例**

**<p>© 2021 SAMURAI.inc</p>**

**実行例**

****

実行すると、全角のコピーライト記号をそのまま書いた場合のみが正しく表示されませんでした。

このように自分のパソコンでは正しく表示されても、他のパソコンでは表示できないということがありますので注意しましょう。

### **よく使う特殊文字一覧**

コピーライト以外にもよく使う特殊文字があります。覚える必要はないので「このようなものがあるのか」と参考程度に見てみましょう。

| **文字** | **文字実体参照** | **数値文字参照** | **文字** | **文字実体参照** | **数値文字参照** |
| --- | --- | --- | --- | --- | --- |
| ① | - | &#9312; | ② | - | &#9313; |
| ③ | - | &#9314; | ④ | - | &#9315; |
| ⑤ | - | &#9316; | ⑥ | - | &#9317; |
| ⑦ | - | &#9318; | ⑧ | - | &#9319; |
| ⑨ | - | &#9320; | ⑩ | - | &#9321; |
| ⑪ | - | &#9322; | ⑫ | - | &#9323; |
| ⑬ | - | &#9324; | ⑭ | - | &#9325; |
| ⑮ | - | &#9326; | ⑯ | - | &#9327; |
| ⑰ | - | &#9328; | ⑱ | - | &#9329; |
| ⑲ | - | &#9330; | ⑳ | - | &#9331; |
| Ⅰ | - | &#8544; | Ⅱ | - | &#8545; |
| Ⅲ | - | &#8546; | Ⅳ | - | &#8547; |
| Ⅴ | - | &#8548; | Ⅵ | - | &#854; |
| Ⅶ | - | &#8550; | Ⅷ | - | &#8551; |
| Ⅸ | - | &#8552; | Ⅹ | - | &#8553; |
| ㎜ | - | &#13212; | ㎝ | - | &#13213; |
| ㎞ | - | &#13214; | ㎠ | - | &#13216; |
| ㎡ | - | &#13217; | ㎢ | - | &#13218; |
| ® | &reg; | &#174; | © | &copy; | &#169; |
| ™ | &trade; | &#153; | ℃ | - | &#8451; |
| ㈱ | - | &#12849; | ㈲ | - | &#12850; |
| < | &lt; | &#0060; | > | &gt; | &#0062; |
| " | &quot; | &#0034; | ' | &apos; | &#0039; |

**まとめ**

* 機種依存文字という機種によっては表示できない文字がある
* 機種依存文字の代わりに特殊文字を使う
* 特殊文字には文字実体参照と数値文字参照がある

## **HTML Living Standardとは**

**HTML Living Standard**とは、Apple・Mozilla・Operaの開発者によって設立されたWHATWG（Web Hypertext Application Technology Working Group：ワットダブルジー）という団体が策定するHTMLの仕様です。現在、HTMLの標準仕様となっています。

これまではW3C（World Wide Web Consortium）という団体が策定した「HTML5 / HTML5.1 / HTML5.2」がHTMLの標準仕様でした。W3Cに変わってWHATWGが策定した仕様が標準仕様になりました。

****

HTML Living Standardの仕様は、公式サイト（英語）と日本語訳サイトにて公開されています。英語が苦手な方は日本語訳サイトを参照してください。

* [HTML Standard](https://html.spec.whatwg.org/)
* [HTML Standard 日本語訳](https://momdo.github.io/html/)

ただし日本語訳サイトは有志により翻訳されており、最新の公式仕様に翻訳が追いついていない旨が記載されています。そのため、最新仕様は公式サイトであることを理解しておいてください。

### **コンピューター言語の仕様とは**

ここで、たびたび登場する「仕様」という言葉について説明します。

コンピューター言語には、「そのコンピューター言語の書き方や意味を規定したドキュメント」があり、これが仕様となります。仕様は日々更新され、バージョン（版）により何回目の更新のものか識別します。

HTMLの場合、詳細は後述しますがHTML1.0から始まりHTML5.2となり、そして今回のHTML Living Standardへと変遷しました。この仕様の話はHTMLに限らずどのコンピューター言語にもある話なので覚えておきましょう。

## **24.3 HTML5廃止の経緯**

HTMLの標準仕様の歴史について紹介します。HTML 1.0～2.0 は IETF（Internet Engineering Task Force）という団体が、HTML 3.2～5.2 は W3Cという団体が標準仕様を策定してきました。

| **西暦** | **仕様** | **団体** |
| --- | --- | --- |
| 1993 | HTML1.0 | IETF |
| 1995 | HTML2.0 | IETF |
| 1997 | HTML3.2 | W3C |
| 1997 | HTML4.0 | W3C |
| 1999 | HTML4.01 | W3C |
| 2014 | HTML5 | W3C |
| 2016 | HTML5.1 | W3C |
| 2017 | HTML5.2 | W3C |

一方、WHATWGという団体が2004年に設立され、HTMLの標準化を進めてきました。2019年にW3CがHTMLの標準化を断念し、WHATWG任せることを決定し、2021年にHTMLの標準仕様はHTML Living Standardになりました。

| **西暦** | **出来事** |
| --- | --- |
| 2004 | Apple、Opera、Mozillaの開発者がWHATWGという団体を設立 |
| 2011 | HTML Living Standard 開始 |
| 2017 | Microsoftが参加 |
| 2019 | W3CがHTMLの標準化を断念し、WHATWG任せることを決定 |
| 2021 | W3CのHTMLの仕様は廃止され、WHATWG の HTML Living Standardに完全統一 |

## **24.4 HTML5.2とHTML Living Standardの違い**

HTML5.2とHTML Living Standardとの違いは、ほぼありません。「[HTML Standard 日本語訳](https://momdo.github.io/html/)」によると、HTML Living StandardとHTML5は同じである旨が記載されています。

HTML Living Standardが標準仕様になって、一部の要素や属性が追加・変更・削除されましたが、それらを使っていないサイトの場合は影響がありません。

ここで、要素と属性の違いについて復習しておきましょう。詳細は[2章](https://terakoya.sejuku.net/programs/51/chapters/576)で確認してください。



### **追加・廃止された要素**

それでは、HTML Living Standardが標準仕様になって追加・廃止された要素について紹介します。なお以下は現時点（2022年11月）での内容であり、日々更新されるので最新版は公式サイトで適宜確認しましょう。

#### **追加された要素**

| **要素名** | **用途** |
| --- | --- |
| hgroup | 複数の見出し(h1〜h6要素)をグループ化する |
| slot | slot(ここでは説明は省略します) |

#### **廃止された要素**

| **要素名** | **用途** |
| --- | --- |
| rb | 「Ruby Base」の略。文字にルビをふる際に使用する |
| rtc | ルビテキスト（ふりがなの文字列）のコンテナを表す |

上記以外にも細かな「変更された要素」や「追加・変更・廃止された属性」もあります。ただし、初学者があまり利用することがない要素・属性の変更であり、説明しても混乱するので割愛します。

興味がある人はインターネットで「HTML Living Standard HTML5 変更点」と検索してみましょう。

## **24.5 HTML Living Standardに準拠しているか確認する方法**

HTML Living Standardの仕様に準拠して作成することで、WebサイトやWebアプリケーションなどのコンテンツの品質を保つことができます。

では、HTML Living Standardの仕様に準拠しているか否かを確認するにはどうすればよいでしょうか。以下の2種類の方法で確認します。

1. 公式サイトで気になる要素と属性を調べる
2. 適合性チェッカーを使う

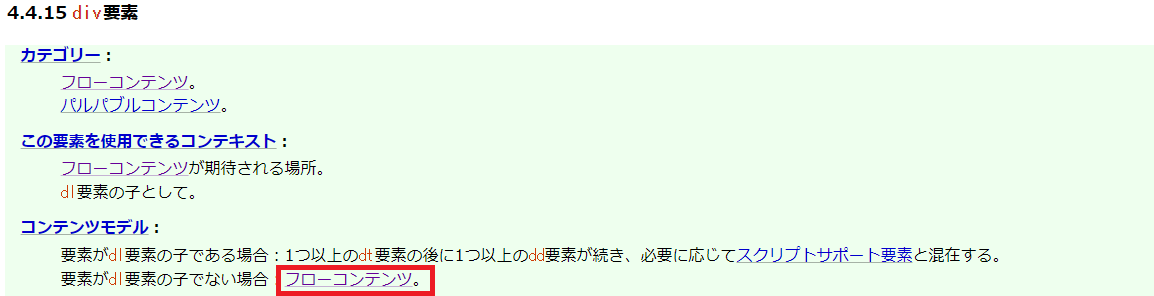
ひとつは気になった要素・属性に関して地道に1つずつ確認する方法です。もうひとつはWebサイトやWebアプリのアドレス（URL）を指定してまとめてチェックする方法です。

順番に説明します。

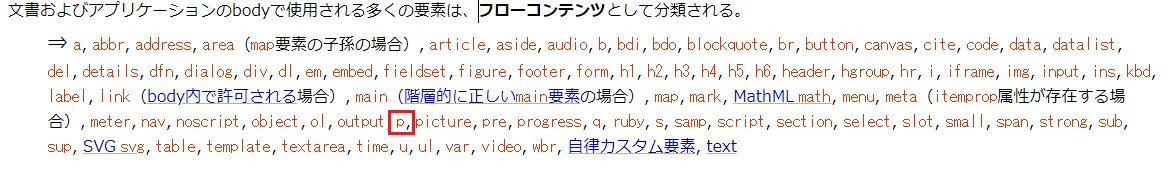
### **（1）仕様を確認する**

1つめの方法は、気になった要素・属性に関して、日本語訳サイトの「[4 HTMLの要素](https://momdo.github.io/html/#toc-semantics)」に記載されている仕様を1つ1つ調べることです。

例えばdiv要素に関して確認したい場合、「[4.4.15 div要素](https://momdo.github.io/html/grouping-content.html#the-div-element)」にアクセスすると以下のように表示されます。

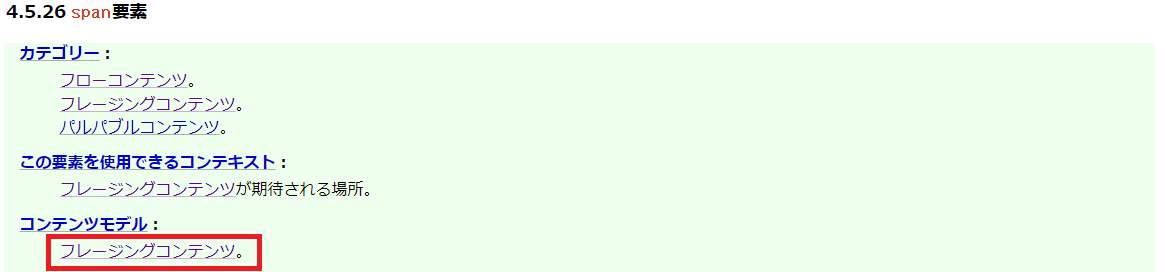


「コンテンツモデル」に記述されているコンテンツをdivの子要素に指定できます。赤枠の「フローコンテンツ」をクリックすると以下のように表示されます。

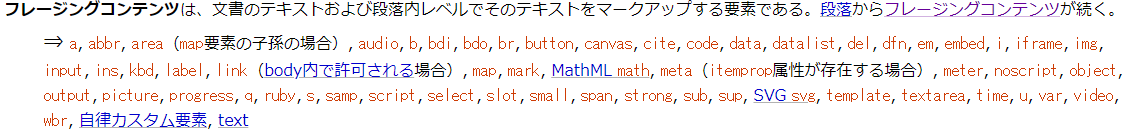


ここに記載されている要素がdivの子要素に指定できます。上図の場合、p要素をdivの子要素にできることがわかります。

次に、span要素を見てみましょう。[4.5.26 span要素](https://momdo.github.io/html/text-level-semantics.html#the-span-element)にアクセスすると、以下のように表示されます。



「コンテンツモデル」に記述されているコンテンツをspanの子要素に指定できます。赤枠の「フレージングコンテンツ」をクリックすると、以下のように表示されます。

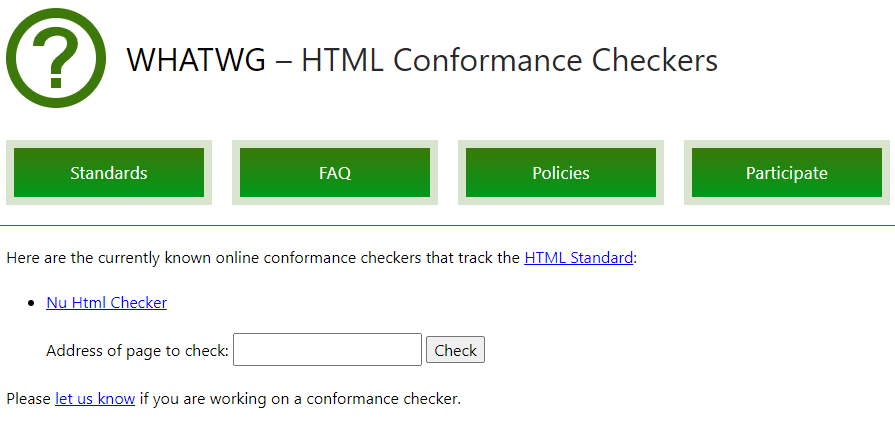


spanの子要素にはp要素を指定できないということがわかります。つまり、spanの子要素にp要素があるWebサイトは、HTML Living Standardに準拠していないことになります。

このように、HTML Living Standardに準拠しているか否かを確認してください。

### **（2）適合性チェッカーを利用する**

2つめの方法は、チェッカーでサイトをまとめてチェックする方法です。公式サイト日本語訳「[1.10.3 HTMLを記述するときに誤りを見つける方法：バリデーターと適合性チェッカー](https://momdo.github.io/html/introduction.html#how-to-catch-mistakes-when-writing-html:-validators-and-conformance-checkers)」にアクセスすると「適合性チェッカー」があります。



「Address of page to check:」の箇所にWebサイトやWebアプリケーションのURL（アドレス）を入力して「Check」ボタンをクリックすると、HTML Living Standardに準拠しているか否かをチェックできます。

WebサイトやWebアプリケーションを公開する前にチェックするとよいでしょう。

まとめ

* HTML Living StandardはHTMLの標準仕様であること
* HTML5.2とHTML Living Standardはほぼ同じであること
* HTML Living Standardに準拠しているか否か確認する方法は以下の2つ
  + 仕様を確認する
  + 適合性チェッカーを利用する