Web プログラミング

第十一回：JavaScript(1)

JS(1) - JavaScript とは

JavaScript を使えば、Web ページの動作をプログラムすることができます。

これまで作ってきた HTML+CSS の Web ページは、
記述されている内容が表示されるだけの、
いわゆる**静的（Static）**なページです。

JavaScript を使えば、インタラクティブな動作や
外部リソースの読み込みなど、**動的（Dynamic）**なページを実装できるようになります。

「JavaScript」と「Java」は、名前は似ていますが
完全に別のプログラミング言語です。
JavaScript を略して Java と呼ばないように。

JS(1) - JavaScript とは

HTML：内容、文章、コンテンツ

CSS：見た目、レイアウト

JavaScript：動き、インタラクション

JS(1) - JavaScript の導入

<script> タグを使用して、HTML に JavaScript を記述できます。

<body> タグの中の最後（閉じタグ </body> の直前）に

<script> タグを書きましょう。

```
<html>
  <head>
    ...head タグの内容 ...
  </head>
  <body>
    ...body タグの内容 ...
    <script>
      console.log("hello world");
    </script>
  </body>
</html>
```

JS(1) - JavaScript の導入

css と同様に、javascript も外部ファイルを読み込む形にできます。
script.js ファイルを新規に作成し、それを読み込む形に書き換えてみましょう

index.html

```
<html>
  <head>
    ...head タグの内容 ...
  </head>
  <body>
    ...body タグの内容 ...
    <script src="script.js"></script>
  </body>
</html>
```

script.js

```
console.log("hello world");
```

JS(1) - 標準出力

console.log を使って、ブラウザのコンソールにデータを表示できます。

```
console.log("hello world");
```

キーボードの Control + Shift + i を押してデベロッパーツールを開き、
「コンソール」を選択して表示を確認しましょう。

JS(1) - 変数

変数は、いろいろなデータを格納できる「入れ物」です。
以下の例では、「data」という名前の変数に「10」というデータが入っています。

```
var data = 10;
```

変数宣言　変数の名前　格納するデータ

JS(1) - 変数

変数は、いろいろな種類のデータを格納できます。

`var data = 10;`	数値
`var data = 100 * 1.5;`	計算結果（100×1.5 なので、150 が入る）
`var data = " テキスト ";`	文字列（ダブルクォーテーションで囲む）
`var data = [" 東 ", " 西 "," 南 "," 北 "];`	配列（4 つの文字列データが入っている）
`var data = true;`	真偽値（true もしくは false）
`var data1 = 10;` `var data2 = data1;`	変数に変数を格納 （data1、data2、両方 10 が入る）

JS(1) - 配列

配列はデータの種類の一種で、複数のデータの塊です。

カッコ [ ] の中にカンマ , で区切って、複数のデータを格納します。

以下の例では、data という変数に 4 つの文字列データ（東、西、南、北）が入っています。

```
var data = [" 東 ", " 西 "," 南 "," 北 "];
```

0 番目のデータ　1 番目のデータ　2 番目のデータ　3 番目のデータ

「data という変数の何番目か」を指定することで、配列の特定の一つにアクセスできます。

番号は 0 から始まることに注意してください。

```
console.log( data[0] );
```

→ コンソールに「東」が表示される

JS(1) - 配列

配列の特定の位置にデータを入れることもできます。

```
data[1] = " にし";
```

→ 配列データは [" 東 ", " にし "," 南 "," 北 "] になる

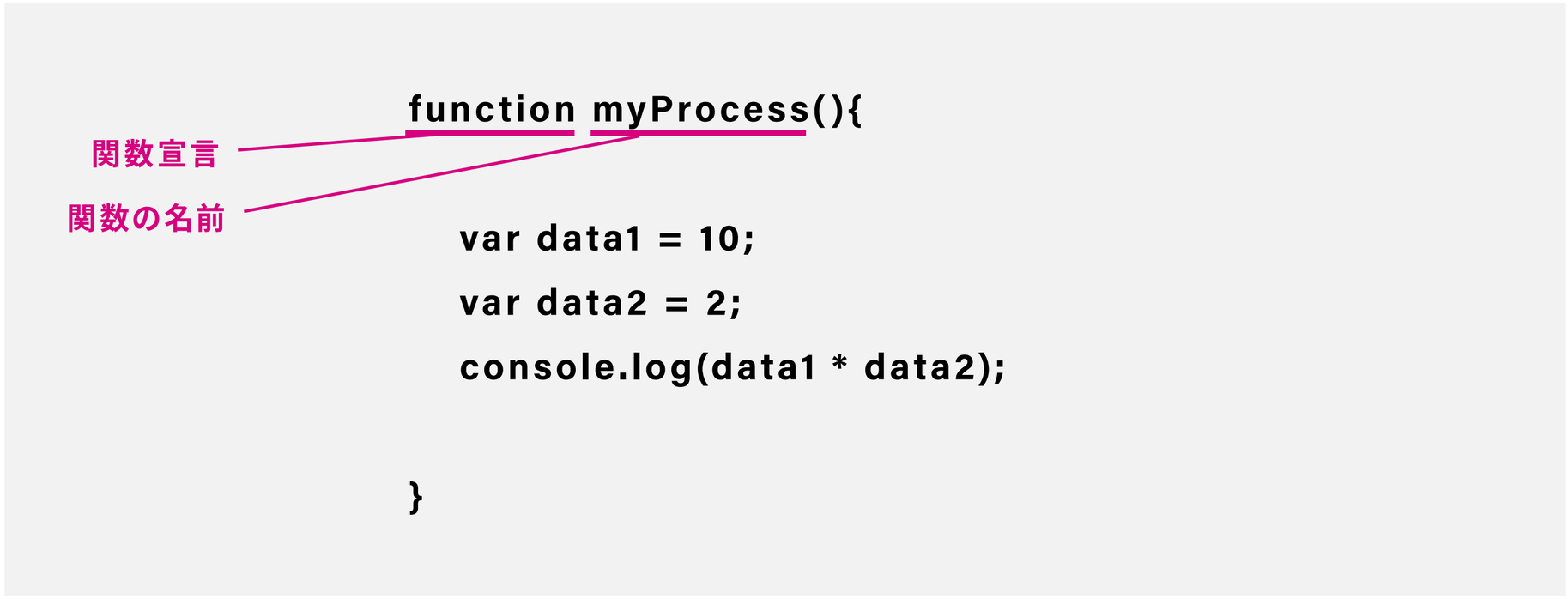
配列に入っているデータの量を「長さ」といいます。
配列データの length を見れば、その配列の長さを得ることができます。

```
console.log( data.length );
```

→ 配列には 4 つのデータが入っているので、コンソールに「4」が表示される

JS(1) - 関数

関数は、プログラム（処理）の一塊です。
一連の作業に名前をつけて、呼び出すことができます。



関数を作った後、関数名 () と書くことで実行できます。

```
myProcess();
```

→ 関数の作業内容が実行され、コンソールに「20」が表示される

JS(1) - if 文

if 文は、「もし〜だったら実行する」のように、処理に条件をつけることができます。

```
var data = 10;
if( data<100 ){
  console.log( "hello" );
}
```

判定条件

実行される作業内容

→ 変数 data が 100 以下だったら実行する。data は 10 なので実行され、コンソールに「hello」と表示される

JS(1) - if 文

if 文と組み合わせて else 文を使うと、
「もし〜だったらこっち、そうでなければそっちを実行する」という処理にできます

```
var data = 1000;
if( data<100 ){
    console.log( "100 以下 " );
}else{
    console.log( "100 以上 " );
}
```

→ 変数 data は 100 以上なため、条件に合致せず、else 文の中に書かれている処理が実行される
コンソールには「100 以上」と表示される。

JS(1) - if 文

判定条件には、以下のような種類がある

`if( data < 100 ){ ... }`	data が 100 より少なかったら実行
`if( data > 100 ){ ... }`	data が 100 より多かったら実行
`if( data <= 100 ){ ... }`	data が 100 より少ないか、同じだったら実行
`if( data >= 100 ){ ... }`	data が 100 より多いか、同じだったら実行
`if( data == 100 ){ ... }`	data が 100 だったら実行

条件判定の結果は、真偽値の true もしくは false となります。
if 文は、true の時は実行され、false の時は実行されません。

```
var flag = data < 100;
if( flag ){ ... }
```

「data が 100 より少なかったら実行」と同じ意味

JS(1) - for 文

for 文は、繰り返しの処理を定義できます。

```
for( var i=0; i<10; i=i+1 ){
     ①        ②      ③
    var data1 = 10;
    console.log( data1 * i );

}
```

繰り返し実行される作業内容

→ コンソールに 10 回数値が表示される。各回、「10 x 繰り返し回数」の数値が表示される。

①は、最初の 1 度だけ行われる処理です。繰り返しの制御のための変数を宣言しています。

②は、繰り返しの終了条件です。変数 i が 10 以下である限り繰り返されます。

③は、繰り返す度に行われる処理です。i に i+1 を入れています。つまり i を 1 増やしています。

JS(1) - 例題 1

以下のプログラムで、何が起こるかを確かめてみましょう。

```
for( var i=0; i<10; i=i+1 ){
    if(i > 5){
        console.log( i )
    }
}
```

JS(1) - 例題 2

以下のプログラムで、何が起こるかを確かめてみましょう。

```
var data = [10, 20, 30, 40, 50];
for( var i=data.length-1; i>=0; i=i-1 ){
        console.log(data[i])
}
```

JS(1) - HTML の操作 - 要素の取得

JavaScript で web ページを操作するために、
document.querySelector を使って、操作したい HTML 要素を取得します。

index.html

```
<html>
  <head>
    ...head タグの内容 ...
  </head>
  <body>
    <div class="block">ブロック </div>
    <script src="script.js"></script>
  </body>
</html>
```

script.js

```
var target = document.querySelector(".block")
console.log( target );
```

→「block」というクラス名の HTML 要素を target 変数に入れている。
コンソールには取得した HTML そのものが表示される

JS(1) - HTML の操作 - 要素の取得

document.querySelector は、セレクタに合致する HTML 要素を一つだけ取得します。
もし複数セレクタに合致した場合、最初の 1 つめを取得します。

index.html

```
<html>
  <head>
    ...head タグの内容 ...
  </head>
  <body>
    <div class="block">ブロック 1</div>
    <div class="block">ブロック 2</div>
    <script src="script.js"></script>
  </body>
</html>
```

script.js

```
var target = document.querySelector(".block")
console.log( target );
```

→「block」というクラス名の HTML 要素は複数あるが、1 つめが取得される。
target 変数には「ブロック 1」の HTML 要素が格納される。

JS(1) - HTML の操作 - 要素の取得

document.querySelectorAll を使えば、
セレクタに合致する全ての要素を配列の形で取得できます。

index.html

```
<html>
  <head>
    ...head タグの内容 ...
  </head>
  <body>
    <div class="block">ブロック 1</div>
    <div class="block">ブロック 2</div>
    <script src="script.js"></script>
  </body>
</html>
```

script.js

```
var target = document.querySelectorAll(".block")
console.log( target );
```

→ target 変数には「ブロック 1」「ブロック 2」が配列に入った形で格納される。

JS(1) - HTML の操作 - 書き換え

取得した HTML 要素の innerHTML をいじることで、
中身を書き換えることができます。

script.js

```
var target = document.querySelector(".block");
target.innerHTML = "javascript から書き換え ";
console.log( target );
```

→「block」というクラス名の要素の中に書いてあるテキストが、javascript から書き換えられる。
Web ページに表示されている内容が変わる。

JS(1) - HTML の操作 - css の操作

取得した HTML 要素の style をいじることで、
その要素に直接 css 付与することができます。

script.js

```
var target = document.querySelector(".block");
target.innerHTML = "javascript から書き換え ";
target.style.backgroundColor = "#FFFFFF";
console.log( target );
```

→「block」というクラス名の要素に、背景色（background-color）が指定される

JS(1) - HTML の操作 - css の操作

javascript から css を指定する時は、
これまで css では - で区切っていたところを、
- をつかわず大文字ににする形で書きます。

css の書き方

```
background-color : #FFF;
font-size : 14px;
padding-top : 40px
position : relative;
```

js の書き方

```
target.style.backgroundColor = "#FFFFFF";
target.style.fontSize = "14px";
target.style.paddingTop = "40px";
target.style.position = "relative";
```

JS(1) - HTML の操作 - クラスの操作

HTML 要素の classList から、その要素に新たなクラス名を追加したり、
クラス名を削除したりできます。

script.js

```
var target = document.querySelector(".block");
target.classList.add("newClass");
target.classList.remove("block");
console.log( target );
```

JS(1) - HTML の操作 - クラスの操作

HTML 要素の classList の contains() 関数を使うと、
その要素が指定されたクラス名を持っているかどうかが true / false の形でわかります。

script.js

```
var target = document.querySelector(".block");
target.classList.add("newClass");
target.classList.remove("block");

if( target.classList.contains("block") ){
    console.log("block クラスあり ");
}
```

→taget 変数に入っている要素は、直前にクラス名 block が remove されているので、
if 文判定は false となり、コンソールには何も表示されない。

JS(1) - イベント

addEventListener をつかって、いろいろな「イベント」をきっかけに、
javascript の処理を実行させることができます。

index.html

```
<html>
  <head>
    ...head タグの内容 ...
  </head>
  <body>
    <button class="btn"> ボタン </button>
    <script src="script.js"></script>
  </body>
</html>
```

script.js

```
var target = document.querySelector(".btn");

function btnClick(){
   console.log(" ボタンがクリックされました ");
}

target.addEventListener("click", btnClick);
```

JS(1) - イベント

```
var target = document.querySelector(".btn");

function btnClick(){
    console.log(" ボタンがクリックされました ");
}

target.addEventListener("click", btnClick);
```

あらかじめ関数を用意しておく

対象の HTML 要素が入った変数

イベントの種類

そのイベントが起こったときに
実行される関数名

JS(1) - イベント

以下は、ボタンをするクリックする度に
.block 要素にクラス名「clicked」をつけたり外したりするサンプルコードです。

```
var target = document.querySelector(".block");
var button = document.querySelector(".btn");

function btnClick(){
  if( target.classList.contains("clicked") ){
     target.classList.remove("clicked");
  }else{
     target.classList.add("clicked");
  }
}

button.addEventListener("click", btnClick);
```