Javascript & HTML5
ｻﾝﾌﾟﾙｺｰﾄﾞ集

haseham

ｼﾝﾌﾟﾙなHTML5文書

```
<!doctype html> <!-- このﾃｷｽﾄﾌｧｲﾙが、html文書であることを明示する。 -->

<html>

 <head>
  <title> シンプルなHTML5文書 </title> <!-- webﾍﾟｰｼﾞのﾀｲﾄﾙ -->
 </head>

  <body>

   <p>タグで配置した文字列でぃす</p> <!-- 文字列を配置する。 -->

    <script>

       document.write( "JSで書き込んだ文字列でぃす" ); // この文書に、文字列を書き込む。

       var hensuu1 = 4649; // 変数を宣言して、初期化する。( 数値を代入 )
       document.write( hensuu1 ); // 変数の値を書き込む。

    </script>

  </body>

</html>
```

【 説明 】

・青色の箇所は、**HTML**タグです。
・赤色の箇所は、**javascript**です。

・緑色の箇所は、コメントです。

・ 1行目の **<!doctype html>** ﾀｸﾞ は、決まり文句です。

・その後をみていくと、**<html>** ﾀｸﾞ の中には、
<head> ﾀｸﾞ と **<body>** ﾀｸﾞ が配置されています。

・**<head>** ﾀｸﾞ は、「ページヘッダ」のことであり、
webﾍﾟｰｼﾞがwebﾌﾞﾗｳｻﾞ上に読み込まれる際に、
最初に設定しておきたいことを書くところです。

・**<body>** ﾀｸﾞ は、「ページボディ」のことで、
ﾍﾟｰｼﾞ に表示される内容を、ここに書いて行きます。

・**<script>** ﾀｸﾞ は、**javascript** などのｽｸﾘﾌﾟﾄを書くところで、
昔は、**<head>** ﾀｸﾞ 内に書くことが慣例となっていましたが、
最近では、ﾍﾟｰｼﾞ が表示されるまでの体感速度を上げるために、
<body> ﾀｸﾞ 内の最後に書くようになりました。

HTML5の基本ﾀｸﾞ

さて、それでは次に、
HTML5文書で登場する
標準的なﾀｸﾞを、一通り書いてみます。↓

```
<!doctype html> <!-- このﾃｷｽﾄﾌｧｲﾙが、html文書であることを明示する。 -->

<html>

 <head>

   <title> HTML5の基本ﾀｸﾞ </title> <!-- webﾍﾟｰｼﾞのﾀｲﾄﾙ -->

   <!-- 外部のｽﾀｲﾙｼｰﾄをﾘﾝｸさせる。 -->
   <link rel="stylesheet" href="default.css">

   <!-- 文字ｺｰﾄﾞ。( 原則的に utf-8 を指定する。) -->
   <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
   <!-- <meta charset="UTF-8"> でもよい。 -->

   <!-- このﾍﾟｰｼﾞで使用されている言語を明示する。( 日本語なら、ja ) -->
   <meta http-equiv="content-language" content="ja">

   <!-- ﾃﾞﾌｫﾙﾄの cssﾌｧｲﾙ をﾘﾝｸさせる。 -->
   <meta http-equiv="default-style" content="default.css">

   <!-- 検索ｴﾝｼﾞﾝに対して、ｷｰﾜｰﾄﾞを提示する。 -->
   <meta name="keywords" content="ｷｰﾜｰﾄﾞ1,ｷｰﾜｰﾄﾞ2">

   <!-- 検索ｴﾝｼﾞﾝに対して、このﾍﾟｰｼﾞの簡単な説明を提示する。 -->
   <meta name="description" content="このﾍﾟｰｼﾞの概略">

   <!-- 検索ｴﾝｼﾞﾝに対して、要求を提示する。( 無効なﾌﾞﾗｳｻﾞもある ) -->
   <meta name="robots" content="noindex,nofollow">
```

```
  <!-- noindex  ... URL を index しないように要求する。
       nofollow ... このﾍﾟｰｼﾞ内のﾘﾝｸをたどらないように要求する。 -->

  <!-- このﾍﾟｰｼﾞの著者を明示する。-->
  <meta name="author" content="山田太郎">

</head>

 <body>

  <!-- ﾘﾝｸを配置する。 -->
  <a href="https://www.youtube.com/" target="_blank">
     別ｳｨﾝﾄﾞｳを開いて、ﾘﾝｸ先のﾍﾟｰｼﾞを表示します。</a><br/>

  <!-- 画像を埋め込む。-->
  <img src="http://m9.ideamans.com/images/lena.jpg"
          alt="画像の代替文" title="画像のﾀｲﾄﾙ" width="100" height="100"/><br/>

  <!-- ﾌﾟﾗｸﾞｲﾝにより実行されるﾏﾙﾁﾒﾃﾞｨｱﾌｧｲﾙを埋め込む。( Flash など ) -->
  <embed src="wave.swf" width="100" height="100"/><br/>

  <!-- ﾌﾟﾗｸﾞｲﾝが不要なものも含めて、外部のﾌｧｲﾙを埋め込む。 -->
  <object data="table.html" width="100" height="100"><br/>
     未対応の場合に表示される代替文
     <param name="" value="" /> <!-- ﾊﾟﾗﾒｰﾀの値を渡すこともできる。 -->
  </object>

  <!-- 音声を埋め込む。( chrome と firefox は対応している。ﾌﾟﾗｸﾞｲﾝ不要。) -->
  <audio src="http://www.ne.jp/asahi/music/myuu/wave/cat1.wav" controls>
    未対応の場合に表示される代替文
  </audio><br/>
  <!-- preload 属性     音声を読み込んでおく。
       autoplay 属性    自動再生する。
       loop 属性        ループ再生する。
       controls 属性    再生ﾊﾟﾈﾙを表示する。-->

  <!-- ｷｬﾝﾊﾞｽ (描画領域) を埋め込む。-->
```

```
    <canvas id="canvas1" style="background-color:black;"
                width="100" height="100">
      未対応の場合に表示される代替文
    </canvas><br/>

    <!-- 段落 -->
    <p>
あいうえお
かきくけこ
    </p>

     <!-- ﾎﾞｰﾀﾞｰﾗｲﾝ -->
    <hr>

    <!-- 改行 -->
    <br />

  </body>

</html>
```

・「**web**ｻｲﾄ」というのは、
世界のどこかにあるｻｰﾊﾞｰ上で動作しているﾌﾟﾛｸﾞﾗﾑであり、
その一部である「**web**ﾍﾟｰｼﾞ」というのは、
そのｻｰﾊﾞｰとあなたのﾊﾟｿｺﾝとが通信をすることで、
あなたの目の前のﾌﾞﾗｳｻﾞ上へと送信されてくるものです。

・この時に送信されるwebﾍﾟｰｼﾞのﾌｧｲﾙ形式は、**HTML**形式であり、
HTMLの文法で書かれています。

・この「**HTML**ﾌｧｲﾙ」のことを、
「**HTML**ﾄﾞｷｭﾒﾝﾄ(文書)」ともいいますが、
この文書をﾃｷｽﾄｴﾃﾞｨﾀで開いてみると、前回に説明したように、
要素タグ、テキスト、コメント、などで構成されていることが
おわかりいただけるかと思います。

・そして、この要素ﾀｸﾞの中には、「属性」を付けることができ、
　この中には、各要素ﾀｸﾞ固有の情報が書かれています。

・上の例でみると、茶色で示された箇所が、属性名と、その値です。

・さて、**HTML5**の、目玉機能とも言えるのが、「キャンバス」です。

・従来の**HTML**では、ﾍﾟｰｼﾞ上の絵を変更することが難しく、
　絵を変更する場合は、そのつど、ｻｰﾊﾞｰと通信をして、
　新しいﾍﾟｰｼﾞと差し替えるといったことを行っていましたが、
　今回導入された「ｷｬﾝﾊﾞｽ」を使えば、ｻｰﾊﾞｰと通信をしなくても、
　表示されているﾍﾟｰｼﾞ上に、絵を描くことができます。

・ｷｬﾝﾊﾞｽを使うには、**<canvas>** ﾀｸﾞのところで、
　識別用の**id**と、描画領域のｻｲｽﾞを設定しておきます。

・ﾍﾟｰｼﾞへの描画処理は、**javascript**で書きます。

・**javascript**側では、この**id**を指定して、ｷｬﾝﾊﾞｽｵﾌﾞｼﾞｪｸﾄを取得します。

HTML5のﾃｰﾌﾞﾙ

```
<!doctype html> <!-- このﾃｷｽﾄﾌｧｲﾙが、html文書であることを明示する。 -->

<html>

 <head>
  <title>HTML5のﾃｰﾌﾞﾙ</title> <!-- webﾍﾟｰｼﾞのﾀｲﾄﾙ -->
  </head>

  <body>

    <table> <!-- 表の開始 -->

      <caption> <!-- 表のﾀｲﾄﾙの開始 -->
        <details>
          <summary>表のタイトル</summary>
          <!-- ﾀｲﾄﾙの右端のﾎﾞﾀﾝを押すと、下記の説明文が表示される。 -->
          <p>表の説明文</p> <!-- ﾃﾞﾌｫﾙﾄでは非表示となっている。-->
        </details>
      </caption> <!-- 表のﾀｲﾄﾙの終了 -->

      <thead> <!-- 列のﾍｯﾀﾞ行の開始 -->
        <tr> <!-- 1行目の開始 -->
          <th></th> <!-- 行・列の見出しが交差しているｾﾙ。 -->
          <th>1列目の見出し</th> <!-- 1列目の見出し -->
          <th>2列目の見出し</th> <!-- 2列目の見出し -->
        </tr> <!-- 1行目の終了 -->
      </thead> <!-- 列のﾍｯﾀﾞ行の終了 -->

      <tbody> <!-- ここからが表の内容 -->

        <tr> <!-- 1行目の開始 -->
```

```
      <th>1行目の見出し列</th> <!-- 1行目の見出し列 -->
      <td>1行目の1列目</td> <!-- 1列目 -->
      <td>1行目の2列目</td> <!-- 2列目 -->
    </tr> <!-- 1行目の終了 -->

    <tr> <!-- 2行目の開始 -->
      <th>1行目の見出し列</th> <!--2行目の見出し列 -->
      <td>2行目の1列目</td> <!-- 1列目 -->
      <td>2行目の2列目</td> <!-- 2列目 -->
    </tr> <!-- 2行目の終了 -->

  </tbody> <!-- ここまでが表の内容 -->

  <tfoot> <!-- 列のﾌｯﾀ行の開始 -->
    <tr> <!-- 1行目の開始 -->
      <th></th> <!-- 行・列の見出しが交差しているｾﾙ。 -->
      <td>1列目の補足説明</td> <!-- 1列目の補足説明 -->
      <td>2列目の補足説明</td> <!-- 2列目の補足説明 -->
    </tr> <!-- 1行目の終了 -->
  </tfoot> <!-- 列のﾌｯﾀ行の終了 -->

</table> <!-- 表の終了 -->

</body>

</html>
```

・<table>ﾀｸﾞの中には、次の**4**つのﾀｸﾞがあります。↓

<caption>　表のﾀｲﾄﾙ(説明文を表示するﾎﾞﾀﾝ付き)
<thead>　ﾍｯﾀﾞ行　(列の見出しが表示される行)
<tbody>　表の内容
<tfoot>　ﾌｯﾀ行　(各列の補足説明などを表示する行)

・さらに、下**3**つのﾀｸﾞの中には、次のﾀｸﾞがあります。↓

<tr>　行

・そして、その1行の中には、次の2種類のﾀｸﾞがあり、
これが各列のｾﾙとなります。↓

<td>　列ｾﾙ(通常版)
<th>　列ｾﾙ(見出し用であり、ﾃｷｽﾄは太字)

・ **<tbody>**以外の**3**つのﾀｸﾞは省略できます。

・行や列の中に、別の表を配置することもできます。

ﾌｫｰﾑとｺﾝﾄﾛｰﾙ

・まず、ﾃｷｽﾄｴﾃﾞｨﾀを起動して、下記のﾃｷｽﾄを貼り付け、
「**form1.html**」というﾌｧｲﾙ名で保存してください。

・そして、そのｱｲｺﾝをｸﾘｯｸすると、
webﾌﾞﾗｳｻﾞが起動して、**web**ﾍﾟｰｼﾞが表示されたはずです。

・**web**ｻｰﾋﾞｽでは、会員登録の画面など、
ﾕｰｻﾞｰからの入力が必要な際には、
このような入力画面を表示します。

・これを「ﾌｫｰﾑ」といい、その内側には、このﾍﾟｰｼﾞのように、
ﾃﾞｰﾀ入力用のｺﾝﾄﾛｰﾙが、いくつか配置されます。

・さて、ﾕｰｻﾞｰは、入力を終えると、送信ﾎﾞﾀﾝを押します。

・このとき、入力されたﾃﾞｰﾀは、webｻｰﾋﾞｽの本体であるｻｰﾊﾞｰに送られます。

・そして、その結果として、ｻｰﾊﾞｰから、別の**html**ﾌｧｲﾙが送信されて、
ﾌﾞﾗｳｻﾞ上に表示されるのです。(ｱﾝｹｰﾄの集計結果など)

```
<!doctype html> <!-- このﾃｷｽﾄﾌｧｲﾙが、html文書であることを明示する。 -->

<html>

 <head>

  <title>ﾌｫｰﾑと入力ｺﾝﾄﾛｰﾙ</title> <!-- webﾍﾟｰｼﾞのﾀｲﾄﾙ -->

   <script>

   </script>

  </head>
```

```
<body>

 <!-- 属性などはこちら。http://www.htmq.com/html5/input.shtml -->

 <!-- 入力ﾌｫｰﾑ -->
 <form action="ﾃﾞｰﾀの送信先であるｻｰﾊﾞｰのURL" method="post">
  <!-- method は、送信方法で、getだと、送信先URL?入力ﾃﾞｰﾀが送信される。
      post だと、入力内容だけが送信され、URLのところに入力ﾃﾞｰﾀが表示されない。-->
  <!-- enctype は、送信するﾃﾞｰﾀの形式で、次のうちのいずれか。↓
      application/x-www-form-urlencoded は、
          「ﾌｨｰﾙﾄﾞ名=入力値」を、& でつないだ形式のﾃﾞｰﾀ。（ﾃﾞﾌｫﾙﾄ値）
      multipart/form-data は、ﾌｧｲﾙ名を含むﾃﾞｰﾀ。
      text/plain は、ﾌﾟﾚｰﾝﾃｷｽﾄのみ。 -->
  <!-- target 属性には、ﾘﾝｸ先のﾍﾟｰｼﾞを表示するﾌﾚｰﾑを指定する。
          https://helpx.adobe.com/jp/legacy/kb/222191.html -->
  <!-- ｺﾝﾄﾛｰﾙに、autofocus 属性をつけると、起動時にﾌｫｰｶｽされる。 -->

  <!-- ﾎﾞﾀﾝ -->
  <p>
   <input type="button" name="button1" value="ﾎﾞﾀﾝです"/>
  </p>

  <!-- 画像ﾎﾞﾀﾝ -->
  <p>
   <input type="image" src="image1.png" alt="画像ﾎﾞﾀﾝです"/>
  </p>

  <!-- ﾃｷｽﾄﾎﾞｯｸｽ -->
  <p>
   <label>見出し:</label> <!-- ﾗﾍﾞﾙ -->
   <input type="text" name="textbox1"
          size="30" maxlength="20" value="入力文字列"/>
  </p>
  <!-- sizeは、表示する文字数。maxlengthは、入力可能な文字数。
      required を付けると、必須入力。readonly を付けると、入力不可。-->
  <!-- placeholder="文字列" で、背景にうすく指示ﾒｯｾｰｼﾞを表示する。 -->
```

```
<!-- type=number で、数値入力。
     ( 右端に、増減ﾎﾞﾀﾝが表示される。step 属性で、増減値を指定可。) -->
<!-- type=password で、ﾊﾟｽﾜｰﾄﾞ入力。-->
<!-- 日付を入力する場合は、ｶﾚﾝﾀﾞｰを表示するﾎﾞﾀﾝが右端に付く。-->
<!-- type=date で、日付入力。( 表示は、2015/12/31　値は、2015-12-31 ) -->
<!-- type=time で、時間入力。( 表示は、12:38 ) 値は、12:38 -->
<!-- type=datetime-local で、ﾛｰｶﾙ日時入力。
     ( 表示は、2015/12/31 12:38　値は、2015-12-31T12:38 ) -->
<!-- type=month で、年月入力。( 表示は、2015/12 値は、2015-12 ) -->
<!-- type を url email tel にした場合は、入力値の検証が行われる。
   そして、その検証結果が無効である場合は、ﾃﾞｰﾀは送信されない。
   ( novalidate 属性を付けると、検証をOFFにできる。)-->
<!-- pattern="" で、入力可能な文字列形式を、正規表現で指定できる。-->

<!-- ｺﾝﾎﾞﾎﾞｯｸｽ ( ﾘｽﾄﾎﾞｯｸｽから選べるｵｰﾄｺﾝﾌﾟﾘｰﾄ機能が付いたﾃｷｽﾄﾎﾞｯｸｽ ) -->
<p>
 <label>見出し:</label> <!-- ﾗﾍﾞﾙ -->
 <input type="text" name="combobox1"
          autocomplete="on" list="input_list1"/>
 <datalist id="input_list1">
     <option value="選択肢1">
     <option value="選択肢2">
  </datalist>
</p>

 <!-- ﾘｽﾄﾎﾞｯｸｽ -->
<label>見出し:</label> <!-- ﾗﾍﾞﾙ -->
<select id="listbox1" name="listbox1" size="2">
    <option value="item1" selected>選択肢1</option>
    <option value="item2">選択肢2</option>
</select>
<!-- multiple 属性を付けると、複数選択可になる。-->
<!-- size 属性は、表示される選択肢の数。-->

<!-- 複数行ﾃｷｽﾄﾎﾞｯｸｽ -->
<p>
 <label>見出し:</label> <!-- ﾗﾍﾞﾙ -->
```

```
 <textarea id="textarea1" name="textarea1"
                cols="40" rows="4" maxlength="20">初期値</textarea>
</p>
<!-- wrap 属性を soft にすると、自動折り返し + 送信ﾃﾞｰﾀに反映しない
hard にすると、自動折り返し + 送信ﾃﾞｰﾀに反映する
off にすると、自動折り返しをしない -->

<!-- ﾗｼﾞｵﾎﾞﾀﾝ -->
 <fieldset name="fieldset1"> <!-- ｸﾞﾙｰﾌﾟﾎﾞｯｸｽ ( 省略可 ) -->
    <legend>見出し:</legend> <!-- ｸﾞﾙｰﾌﾟﾎﾞｯｸｽの見出し ( 省略可 ) -->
    <label>選択肢1</label>
    <input type="radio" name="radio1" value="item1" checked="true"/>
    <label>選択肢2</label>
    <input type="radio" name="radio1" value="item2"/>
 </fieldset>

<!-- ﾁｪｯｸﾎﾞｯｸｽ -->
<fieldset name="fieldset2"> <!-- ｸﾞﾙｰﾌﾟﾎﾞｯｸｽ ( 省略可 ) -->
     <legend>見出し:</legend> <!-- ｸﾞﾙｰﾌﾟﾎﾞｯｸｽの見出し ( 省略可 ) -->
     <label>選択肢1</label>
     <input type="checkbox" name="checkbox1"
               value="item1" checked="true"/>
     <label>選択肢2</label>
     <input type="checkbox" name="checkbox1"
               value="item2"/>
 </fieldset>

<!-- ﾌｧｲﾙ選択ﾎﾞﾀﾝ ( ﾎﾞﾀﾝの右側に、選択したﾌｧｲﾙ名が表示される。) -->
<p>
 <label>見出し:</label> <!-- ﾗﾍﾞﾙ -->
 <input type="file" name="file_select_button1"/>
</p>

<!-- 調節つまみ ( min で最小値を、max で最大値を、それぞれ指定可。) -->
<p>
 <label>見出し:</label> <!-- ﾗﾍﾞﾙ -->
 <input type="range" name="thumb1"/>
```

```
    </p>

    <!-- 色選択ﾎﾞﾀﾝ -->
    <p>
     <label>見出し:</label> <!-- ﾗﾍﾞﾙ -->
     <input type="color" name="color_select_button1"/>
    </p>

    <!-- 進捗ﾊﾞｰ -->
    <p>
     <label>見出し:</label> <!-- ﾗﾍﾞﾙ -->
     <progress max="100" value="50"></progress>
    </p>

    <!-- 横向きの棒ｸﾞﾗﾌ -->
    <p>
      <label>見出し:</label> <!-- ﾗﾍﾞﾙ -->
      <meter max="100" value="50"></meter>
    </p>

    <!-- このﾌｫｰﾑの、 送信ﾎﾞﾀﾝ -->
    <p>
      <input type="submit" value="送信"/>
      <input type="reset" value="リセット"/>
    </p>

   </form>

  </body>

</html>
```

---

javascript の基本構文

```
<!doctype html> <!-- このﾃｷｽﾄﾌｧｲﾙが、html文書であることを明示する。 -->

<html>

 <head>

  <title> javascript の基本構文 </title> <!-- webﾍﾟｰｼﾞのﾀｲﾄﾙ -->

   <!-- javascript -->
    <script src="import.js"> <!-- 外部のjsを呼び出す。 -->
    </script>

   <!-- javascript -->
    <script> <!-- 直接書き込む。 -->

     var v1;       // 変数の宣言。( ｸﾞﾛｰﾊﾞﾙ変数は、window のﾌﾟﾛﾊﾟﾃｨとして参照できる。)
     var v2 = 1;  // 数値で初期化する
      var v3,v4;          // 複数の変数を、同時に宣言する。
      var v5 = "あああ"; // 変数を文字列で初期化する。
      var v6 = "長い文は、 \
            折り返せる";  // ﾊﾞｯｸｽﾗｯｼｭで折り返せる。
      let v7;              // ﾛｰｶﾙ変数の宣言。( ﾌﾞﾛｯｸ内で宣言した場合は、外では無効。)
      const v8 =120;   // 定数の宣言。
       // 真偽値     true     false
      // 数値       100     0.01
      // 文字列   "あいうえお"
      // null              null値
      // undefined   未定義値
      // NaN           無効な値     if ( isNaN( v1 ) ) return;
      // 数値の文字列は、数値に変換できる。 parseInt()   parseFloat()
      // ( p1は文字列。p2は数記法の基数。失敗したら、NaNを返す。)

     var array1 = ["山形", "富山", "岡山"];// 配列の宣言。
     // = [0,1,,3]  省略した要素は、undefined になる。
```

```
var v9 = array1[0]; // 配列の要素を参照する。
// var length1 = "あいう".length; // 文字列は、Stringｵﾌﾞｼﾞｪｸﾄのﾒﾝﾊﾞが使える。

var obj1 = { namae: "太郎", nenrei: 23 }; // ｵﾌﾞｼﾞｪｸﾄを初期化する。

var v10 = obj1.namae; // ｵﾌﾞｼﾞｪｸﾄのﾌﾟﾛﾊﾟﾃｨを参照する。

var v11 = typeof v2; // 変数のﾃﾞｰﾀ型名を取得する。 ( "string" "number" )


if ( v1 == true ) {} // if 文
if ( v1 != false ) {} else {} // if else 文

switch ( v1 ) { case 0: break; default: break; } // switch 文

for ( var i = 0; i < 10; i++ ){} // for 文 ( break; continue; も使える。)
for ( var cur in array1 ) {} // for each 文

while ( v1 ) {} // while 文
do {} while( v1 ); // do while 文

); // ｺﾝｿｰﾙに出力する。

var result_bool = confirm( "ﾀｲﾄﾙというか質問" ); // OKﾀﾞｲｱﾛｸﾞを表示する。

var result_text = prompt( "ﾀﾞｲｱﾛｸﾞのﾀｲﾄﾙ", "初期値" ); // 入力ﾀﾞｲｱﾛｸﾞを表示する。

function Tasu( p1 , p2 ) // 関数を実装する。
{
  return p1 + p2;
}

// 関数式 ( 関数名は省略できるが、この関数内で呼び出す場合には必要。 )
var Hiku = function ( p1 , p2 ){ return p1 - p2; };
var sa = Hiku( 2 , 1 ); // 関数を呼び出す。

function TashiteWaru( p1 , p2 )
```

```
  {
    var v1 = p1 + p2;

    // ｸﾛｰｼﾞｬ ( 関数の実装内に、別の関数を実装する。)
    // 呼び出し元のﾛｰｶﾙ変数は、まだあるので使える。
    return function () { return v1 / 2; }
  }

    // 例外処理
  try {}
  catch ( ex ) { throw ex; }
  finally {}

 </script>


 <noscript>
   <p>このページでは、JavaScript を使用しています。</p>
 </noscript>

</head>

<body>
```

javascript のｸﾗｽ構文

```
<!doctype html> <!-- このﾃｷｽﾄﾌｧｲﾙが、html文書であることを明示する。 -->

<html>

 <head>

  <title> javascript のｸﾗｽ構文 </title> <!-- webﾍﾟｰｼﾞのﾀｲﾄﾙ -->

    <!-- javascript -->
     <script> <!-- 直接書き込む。 -->

      // -------------------------
      // ｸﾗｽの宣言

      class Class1
      {

       constructor( name ) // ｺﾝｽﾄﾗｸﾀ
       {
          this.name = name; // ﾌﾟﾛﾊﾟﾃｨを初期化する。
       }

       set name( name ) // ﾌﾟﾛﾊﾟﾃｨ の Setter
       {
          this.name = name;
       }

       get name() // ﾌﾟﾛﾊﾟﾃｨ の Getter
       {
          return this.name;
       }

       method1() // ﾒｿｯﾄﾞ
       {
          console.log( this.name );
```

```
    }

  } // end class

  // -------------------------
  // Class1 を ｲﾝｽﾀﾝｽ化 する。

  var instance1 = new Class1( '太郎' );
  var instance2 = new Class1( '花子' );
  instance1.method1(); // 太郎
  instance2.method1(); // 花子

  // -------------------------
  // 子ｸﾗｽの宣言 ( ｸﾗｽ継承 )

  class Class2 extends Class1
  {
    constructor( name ) // ｺﾝｽﾄﾗｸﾀ
    {
      super( name ); // 親ｸﾗｽのｺﾝｽﾄﾗｸﾀを呼び出す。
    }
  } // end class

    // -------------------------

 </script>

</head>

<body>
```

ﾍﾟｰｼﾞの要素を動的に更新する

```
<!doctype html> <!-- このﾃｷｽﾄﾌｧｲﾙが、html文書であることを明示する。 -->

<html>

 <head>

  <title> ﾍﾟｰｼﾞの要素を動的に更新する </title> <!-- webﾍﾟｰｼﾞのﾀｲﾄﾙ -->

    <!-- javascript -->
     <script>

      // ----------------
      // window がｸﾘｯｸされた時のｲﾍﾞﾝﾄ処理を設定する。

      window.onclick = function()
       {

       //「div」要素ﾀｸﾞを新規作成する。
       var div1 = document.createElement( "div" );

       //「strong」要素ﾀｸﾞを新規作成する。
       var strong1 = document.createElement( "strong" );

       // それを「div」要素ﾀｸﾞに追加する。
       div1.append( strong1 );


       // ﾃｷｽﾄを「strong」要素ﾀｸﾞに追加する。
       strong1.append( "クリックされました。" );

       //「div」要素ﾀｸﾞ「view1」を、取得し、
       var div2 = document.getElementById( "view1" );
```

```
        // 新規作成したものと入れ替える。
          div2.innerHTML = div1.innerHTML;

      };

      // ----------------

  </script>

 </head>

 <body>
  <div  id="view1">クリックしてみてください。</div>
 </body>

</html>
```

・ getElementById ﾒｿｯﾄﾞの引数1は、「id」属性の値で、
  この引数に渡された値を持つ要素ﾀｸﾞを返します。

・ この他にも、「name」属性の値を指定して
    要素ﾀｸﾞを取得する getElementByName ﾒｿｯﾄﾞや、
    要素ﾀｸﾞ名を指定して要素ﾀｸﾞを取得する
    getElementByTagName ﾒｿｯﾄﾞもあります。

```
var div1 = getElementsByTagName( "div" ); // ﾀｸﾞ名で取得する。
var form1 = getElementsByName( "form1" );  // name属性の値で取得する。
var form1 = getElementById( "form1" );        // id属性の値で取得する。
var pain1 = getElementsByClassName( "pain1" ); // class属性の値で取得する。
var my_class1 = querySelector( ".myClass" ); // cssｾﾚｸﾀで取得する。(最初の一つ)

var my_class1 = querySelectorAll( ".myClass" ); // cssｾﾚｸﾀで取得する。(すべて)
```

・ｺﾝﾄﾛｰﾙなど、id属性を設定してある要素ﾀｸﾞについては、
ｵﾌﾞｼﾞｪｸﾄは、内部的にｸﾞﾛｰﾊﾞﾙ変数として宣言されているため、
上記のﾒｿｯﾄﾞで取得しなくても、そのまま使うことができます。

・このように、XML文書の要素ﾀｸﾞを
ｵﾌﾞｼﾞｪｸﾄとしてﾒﾓﾘ上に配置し、ﾌﾟﾛｸﾞﾗﾑによって編集することを、
「**DOM**」(**D**ocument **O**bject **M**odel) といいます。

「body」要素ﾀｸﾞに追加する

```
<!doctype html> <!-- このﾃｷｽﾄﾌｧｲﾙが、html文書であることを明示する。 -->

<html>

 <head>

  <title> 「body」要素ﾀｸﾞに追加する </title> <!-- webﾍﾟｰｼﾞのﾀｲﾄﾙ -->

    <!-- javascript -->
     <script>

      // ----------------
      // window がｸﾘｯｸされた時のｲﾍﾞﾝﾄ処理を設定する。

      var click_count = 0; // ｸﾘｯｸされた回数を記録する変数。

      window.onclick = function()
       {

       //「div」要素ﾀｸﾞを新規作成する。
       var div1 = document.createElement( "div" );

       //「strong」要素ﾀｸﾞを新規作成する。
       var strong1 = document.createElement( "strong" );

       // それを「div」要素ﾀｸﾞに追加する。
       div1.append( strong1 );

       // ﾃｷｽﾄを「strong」要素ﾀｸﾞに追加する。
       strong1.append( "クリックされました。" );

       ++click_count; // ｸﾘｯｸされたので、ｶｳﾝﾄを +1 する。
```

```
        // ﾃｷｽﾄを「strong」要素ﾀｸﾞに追加する。
        strong1.append( click_count + " 回目のクリックです。" );

        //「body」要素ﾀｸﾞに追加する。
        document.body.appendChild( div1 );

      };

      // ----------------

  </script>

 </head>

 <body>
   <div  id="view1">クリックしてみてください。</div>
 </body>

</html>
```

XMLﾌｧｲﾙから document を作成する

```
<!doctype html> <!-- このﾃｷｽﾄﾌｧｲﾙが、html文書であることを明示する。 -->

<html>

 <head>

  <title> XMLﾌｧｲﾙから document を作成する </title> <!-- webﾍﾟｰｼﾞのﾀｲﾄﾙ -->

    <!-- javascript -->
     <script>

      // ----------------
      // window が読み込まれた時のｲﾍﾞﾝﾄ処理を設定しておく。

      window.onload = function ( event )
      {

       // -------------------------
       // XMLﾌｧｲﾙを読み込む。

       var req1 = new XMLHttpRequest(); // HTTPﾘｸｴｽﾄを生成する。

       req1.responseType = "document";     // ﾚｽﾎﾟﾝｽﾀｲﾌﾟを指定しておく。
       req1.overrideMimeType( "text/xml" ); // MIMEﾀｲﾌﾟを上書きしておく。

       // HTTPﾘｸｴｽﾄが読み込まれた時のｲﾍﾞﾝﾄ処理を設定する。
       req1.onload = function ( event )
       {

        if ( req1.readyState === req1.DONE ) // 送信完了なら、
        {
```

```
        if ( req1.status === 200 ) // 送信成功なら、
        {

         // ------------------------

         // document ｵﾌﾞｼﾞｪｸﾄを取り出す。
         var xml_doc1 = req1.responseXML;

         if ( xml_doc1 ) // 取り出せたら、
         {
           //「div」要素ﾀｸﾞ「view1」を、取得し、
           var div1 = document.getElementById( "view1" );

           // XML文書側からも、ﾙｰﾄﾉｰﾄﾞの「div」要素ﾀｸﾞを取り出す。
           var div2 = xml_doc1.getElementById( "root_node" );

           // XML文書側のものと入れ替える。
           div1.innerHTML = div2.innerHTML;
         }

         // ------------------------

        } // end if ( req1.status === 200 )

      } // end  if ( req1.readyState === req1.DONE )

    }; // end req1.onload = function()

    req1.open( "get","sample.xml", true ); // xmlﾌｧｲﾙへの接続を開く。

    req1.send( null ); // HTTPﾘｸｴｽﾄを送信する。

  // ----------------

 </script>

</head>
```

```
 <body>
 </body>

</html>
```

・上記から読み込まれる「**sample.xml**」はこちらです。↓

```
<?xml version="1.0" encoding="utf-8"?>

<div id="root_node">
 <table>
  <tbody>
   <tr> <!-- 1行目の開始 -->
    <th>1行目の見出し</th> <!-- 0列目 -->
     <td>1行目の1列目</td> <!-- 1列目 -->
     <td>1行目の2列目</td> <!-- 2列目 -->
    </tr> <!-- 1行目の終了 -->
    <tr> <!-- 2行目の開始 -->
     <th>2行目の見出し</th> <!-- 0列目 -->
      <td>2行目の1列目</td> <!-- 1列目 -->
      <td>2行目の2列目</td> <!-- 2列目 -->
     </tr> <!-- 2行目の終了 -->
  </tbody>
 </table>

</div>
```

DOMによるﾉｰﾄﾞ操作まとめ

・**HTML**文書や、**XML**文書は、以下のように、
要素ﾀｸﾞの中に、要素ﾀｸﾞが入れ子になる
という「木構造」になっています。↓

```
<RootNode> <!-- ﾙｰﾄﾉｰﾄﾞ -->
  <ParentNode> <!-- 親ﾉｰﾄﾞ -->
    <ChildNode> <!-- 子ﾉｰﾄﾞ -->
    </ChildNode>
  </ParentNode>
</RootNode>
```

・ﾀｸﾞは、「要素」(Element) ばかりではありませんが、
これらの各種ﾀｸﾞは「ﾉｰﾄﾞ」(Node) と総称されています。

・**javascript**では、ﾍﾟｰｼﾞ上の要素ﾀｸﾞを、
document ｵﾌﾞｼﾞｪｸﾄから取得することができ、
これを書き換えることで、ﾕｰｻﾞｰの操作に応じて
ﾍﾟｰｼﾞに変化を加えることができます。

・**document.getElementById** ﾒｿｯﾄﾞから返されるのは、
「**Node**」ｵﾌﾞｼﾞｪｸﾄで、これには、ﾉｰﾄﾞを付け替えるための
便利なﾒｿｯﾄﾞやﾌﾟﾛﾊﾟﾃｨが実装されています。

・まず、ﾉｰﾄﾞ「**node1**」に、ﾉｰﾄﾞ「**node2**」を
子ﾉｰﾄﾞとして追加する場合をみて行きましょう。↓

```
node1.appendChild( node2 ); // 追加する。
```

・このﾉｰﾄﾞ「node2」は、ﾉｰﾄﾞ「node1」の子ﾉｰﾄﾞﾘｽﾄの
末尾に追加されます。

・ちなみに、すでに子ﾉｰﾄﾞであるﾉｰﾄﾞを追加した場合は、
一旦削除されてから、再度、追加されます。(つまり、位置が変化する。)

・同じ子ﾉｰﾄﾞを複製する場合は、cloneNode ﾒｿｯﾄﾞを呼び出します。↓

```
var clone_node1 = node1.cloneNode( true ); // ﾉｰﾄﾞ以下を複製する。
```

・配下の子ﾉｰﾄﾞも含めて、すべて複製する場合は、
上記のように、引数1に **true** を渡します。

・ node1の配下に、node2が存在するかを判定するには、次のように書きます。↓

```
if ( node1.contains( node2 ) ){} // 存在するなら、true。
```

・node1 が、子ﾉｰﾄﾞを持っているかを判定する場合は、次のように書きます。↓

```
if ( node1.firstChild != null ) {}          // 判定方法　その1
```

```
if ( node1.childNodes.length > 0 ) {}  // 判定方法　その2
```

```
if ( node1.hasChildNodes() ) {}        // 判定方法　その3
```

・node1 の子ﾉｰﾄﾞﾘｽﾄにおいて、target_node1 の直前に、
new_node1 を追加するには、次のように書きます。 ↓

```
node1.insertBefore( new_node1, target_node1 );
```

・よく使いそうな事例として、
最初の子ﾉｰﾄﾞの手前に、子ﾉｰﾄﾞを追加していく場合では、
引数2に、node1.firstChild ﾌﾟﾛﾊﾟﾃｨを指定するとよいでしょう。

・node1 と node2 が同じ内容であるかを判定するには、次のように書きます。 ↓

```
if ( node1.isEqualNode( node2 ) ) {}
```

・このﾒｿｯﾄﾞで比較・判定されるのは、ﾉｰﾄﾞのﾀｲﾌﾟや
idなどの属性の値、子ﾉｰﾄﾞの総数など、総じて内容です。

・参照しているｵﾌﾞｼﾞｪｸﾄが同じものであるかを判定する場合は、
isSameNode ﾒｿｯﾄﾞを使います。

・node1 の子ﾉｰﾄﾞﾘｽﾄから、node2を除外するには、次のように書きます。 ↓

```
var removed_node = node1.removeChild( node2 );
```

・node2は、ﾘｽﾄから除外されただけで、ﾒﾓﾘ上から削除されたわけではありません。

・この他にも、次のような方法がよく使われています。 ↓

```
// ﾉｰﾄﾞ自身を削除する。
node1.parentNode.removeChild( node1 );
```

```
// 子ﾉｰﾄﾞをすべて除外する。
while ( node1.firstChild ) node1.removeChild( node1.firstChild );
```

```
// 子ﾉｰﾄﾞをすべて削除する。

node1.textContent = null;
```

・target_node を、new_node に置き換えるには、次のように書きます。↓

```
 var replaced_node = parent_node1.replaceChild( new_node, target_node );
```

・node1 の子ﾉｰﾄﾞを列挙するには、次のように書きます。↓

```
var childs = node1.childNodes;       // 子ﾉｰﾄﾞ一覧を取り出す。

for ( let i = 0; i < childs.length; i++ )  // 子ﾉｰﾄﾞﾘｽﾄのﾙｰﾌﾟ。
    console.log( childs[i].nodeValue );  // 子ﾉｰﾄﾞの値を出力する。
```

・ちなみに、子ﾉｰﾄﾞの総数は、node1.length ﾌﾟﾛﾊﾟﾃｨでも取得できます。

・関係性のあるﾉｰﾄﾞを取得するには、以下の読み取り専用ﾌﾟﾛﾊﾟﾃｨを使います。↓

```
var doc = node1.ownerDocument;          // document ｵﾌﾞｼﾞｪｸﾄを返す。
```

```
var root_node = node1.getRootNode();        // ﾙｰﾄﾉｰﾄﾞを返す。
var parent_node = node1.parentNode;         // 親ﾉｰﾄﾞを返す。
var parent_element = node1.parentElement; // 親要素を返す。( 親が要素でない場合も null。)
var first_child = node1.firstChild;                // 最初の子ﾉｰﾄﾞを返す。
var last_child  = node1.lastChild;                 // 最後の子ﾉｰﾄﾞを返す。
var prev_bros_node = node1.prevSibling;     // 直前の兄弟ﾉｰﾄﾞを返す。
var next_bros_node = node1.nextSibling;     // 直後の兄弟ﾉｰﾄﾞを返す。
```

・ｵｰﾅｰﾉｰﾄﾞの ownerDocument ﾌﾟﾛﾊﾟﾃｨは **null** です。

・ﾉｰﾄﾞの内側にあるﾃｷｽﾄは、書き換えることもできます。↓

```
var text1 = node1.textContent;   // 要素内にあるﾃｷｽﾄを返す。

node1.textContent = "かきくけこ"; // 要素内にあるﾃｷｽﾄを書き換える。
```

・要素内にある**HTML**を読み書きする場合は、innerText ﾌﾟﾛﾊﾟﾃｨを使います。

・しかし、この innerHTML ﾌﾟﾛﾊﾟﾃｨは、ﾊﾟﾌｫｰﾏﾝｽがあまり良くない上に、
XSS攻撃に対する脆弱性が存在しており、利用に際しては、注意が必要です。

・ﾉｰﾄﾞの値を取得、変更するには、次のように書きます。↓

```
var value1 = node1.nodeValue; // ﾉｰﾄﾞの値を取得する。
node1.nodeValue = value1;       // ﾉｰﾄﾞの値を変更する。
```

・この、nodeValue ﾌﾟﾛﾊﾟﾃｨで取得できる値は、ﾉｰﾄﾞのﾀｲﾌﾟによって異なります。↓

・CDATASection Comment Text ... 内容文字列。

・**ProcessingInstruction**　... ﾀｰｹﾞｯﾄを除く全てのｺﾝﾃﾝﾂ。

・**Document　DocumentFragment　DocumentType**
Element　NamedNodeMap　EntityReference　Notation ... null

・ちなみに、**node1** のﾉｰﾄﾞﾀｲﾌﾟを取得するには、次のように書きます。↓

```
var node_type_num = node1.nodeType; // ﾉｰﾄﾞﾀｲﾌﾟ番号を返す。
```

・取得したﾉｰﾄﾞﾀｲﾌﾟは、次の定数と比較することで判定します。↓

```
Node.ELEMENT_NODE                1... Element
Node.TEXT_NODE                   3... Text
Node.PROCESSING_INSTRUCTION_NODE 7... ProcessingInstruction ( <?xml-stylesheet ... ?> など )
Node.COMMENT_NODE                8... Comment
Node.DOCUMENT_NODE               9... Document
Node.DOCUMENT_TYPE_NODE         10... DocumentType  ( <!DOCTYPE html> など )
Node.DOCUMENT_FRAGMENT_NODE     11... DocumentFragment
```

ﾉｰﾄﾞ「**node1**」の持つ属性「**attri1**」の値を取得、変更するには、次のように書きます。↓

```
var value1 = node1.getAttribute( "attri1" ); // 値を取得する。
node1.setAttribute( "attri1", value1 );       // 値を変更する。
```

ﾉｰﾄﾞ「**node1**」が、属性「**attri1**」を持っているかを判定するには、次のように書きます。↓

```
if ( node1.hasAttribute( "attri1" ) ){} // 持っていたら、trueが返る。
```

ﾉｰﾄﾞ「node1」から、属性「attri1」を削除するには、次のように書きます。↓

```
node1.removeAttribute( "attri1" ); // 削除する。
```

・node1 の「ﾍﾞｰｽURI」を取得するには、次のように書きます。↓

```
var base_uri = node1.baseURI; // ﾉｰﾄﾞのURIを返す。( 絶対ﾊﾟｽ )
```

・ちなみに、この「ﾍﾞｰｽURI」は、
HTML文書の場合は、「base」ﾀｸﾞの「href」属性で設定します。↓

```
<base href="http://www.example.com/">
```

・XML文書の場合は、「doc」ﾀｸﾞの、「xml:base」属性で設定します。↓

```
<doc xml:base="http://example.com/"
     xmlns:xlink="http://www.w3.org/1999/xlink">
```

・ﾉｰﾄﾞのｸﾗｽを変更することもできます。

```
// ｸﾗｽ「class1」を追加する。
```

```
node1.classList.add( "class1" );

// ｸﾗｽ「class1」を削除する。
node1.classList.remove( "class1" );

// ｸﾗｽ「class1」をﾄｸﾞﾙする。
node1.classList.toggle( "class1" );

// ｽﾀｲﾙを直接設定する。
node1.style.backgroundColor = '#ff0000';
```

・ここでいう「ｸﾗｽ」とは、ｽﾀｲﾙｼｰﾄの「ｸﾗｽ」のことです。

・うまくいかないときは、こちらのｿｰｽも参考にしてみて下さい。↓

http://ariradne.web.fc2.com/index.html
http://ariradne.web.fc2.com/search.js

ﾎﾞﾀﾝのｸﾘｯｸ処理を書いてみょう

・さて、ﾕｰｻﾞｰがﾎﾞﾀﾝｺﾝﾄﾛｰﾙをｸﾘｯｸしたら、
javascriptで何か処理をさせたい、と思うことがあると思います。

・ﾕｰｻﾞｰがﾎﾞﾀﾝをｸﾘｯｸすると、ｸﾘｯｸｲﾍﾞﾝﾄという
「イベント」が発生しています。

・この時に、「イベントハンドラ」という関数が呼ばれます。

・自作した関数を、ｵﾌﾞｼﾞｪｸﾄのｲﾍﾞﾝﾄﾊﾝﾄﾞﾗとして設定する方法としては、
次の2通りがあります。↓

・ｵﾌﾞｼﾞｪｸﾄの、**onclick** などのﾌﾟﾛﾊﾟﾃｨに、関数を代入する。
・ｵﾌﾞｼﾞｪｸﾄの、**addEventListener**ﾒｿｯﾄﾞに、関数を渡す。

・簡単な例を見ていきましょう。↓

```
<!doctype html> <!-- このﾃｷｽﾄﾌｧｲﾙが、html文書であることを明示する。-->

<html>

 <head>
  <title> ﾎﾞﾀﾝ ｺﾝﾄﾛｰﾙ </title> <!-- webﾍﾟｰｼﾞのﾀｲﾄﾙ -->
 </head>

 <body>

    <!-- 入力ﾌｫｰﾑ ( action ) -->
    <form id="form1" action="" method="" onsubmit="return false">

      <!-- ﾎﾞﾀﾝ -->
      <input type="button" id="button1" value="ﾎﾞﾀﾝです">

    </form>
```

```
<script>

// ------------------------------------------------------
// ﾍﾟｰｼﾞが読み込まれた時のｲﾍﾞﾝﾄ処理。( load ｲﾍﾞﾝﾄ ﾊﾝﾄﾞﾗ )

window.onload = function ( event )
{

    // ｸﾘｯｸｲﾍﾞﾝﾄのﾊﾝﾄﾞﾗを設定する。
    button1.addEventListener( "click",
                              function ( event ) // click ｲﾍﾞﾝﾄ ﾊﾝﾄﾞﾗ
                              {
                                 document.write( "ｸﾘｯｸされました。" );
                              },
                              false );

}; // end window.onload = function ()

// ------------------------------------------------------

</script>

</body>

</html>
```

・さて、上の例では、addEventListener ﾒｿｯﾄﾞが呼ばれていますが、
これは、window の load ｲﾍﾞﾝﾄﾊﾝﾄﾞﾗの中で呼ばれています。

・これはなぜかというと、window がﾛｰﾄﾞされてからでないと、
その上に配置されるﾎﾞﾀﾝｺﾝﾄﾛｰﾙが、有効にならないからです。

・ﾌﾟﾛﾊﾟﾃｨへ代入することによって、ﾊﾝﾄﾞﾗを設定する方法としては、

次のようにして、要素ﾀｸﾞに書くこともできます。↓

```
<body onclick="OnClick();">
```

・それから、引数3の「false」についてですが、
ｲﾍﾞﾝﾄが発生すると、ﾌﾞﾗｳｻﾞは、
どの要素で発生したのか、発生元の「イベントターゲット」を調べるために、
親要素から子要素へとたどっていきます。

・これを「キャプチャーフェーズ」といいますが、
この時に、同じｲﾍﾞﾝﾄのﾊﾝﾄﾞﾗを持っている親要素があった場合、
この引数3を**true**にしていると、そのﾊﾝﾄﾞﾗが先に呼び出されます。

・さて、ｲﾍﾞﾝﾄがどの要素で発生したのかが判明すると、
その要素のﾊﾝﾄﾞﾗが呼び出されるのですが、(「ターゲットフェーズ」という)
そのあとﾌﾞﾗｳｻﾞは、今度は、親要素を順番にたどっていきます。

・これを「バブリングフェーズ」というんですが、
引数3が**false**だった場合は、ここで親要素のﾊﾝﾄﾞﾗが呼ばれます。

ﾃｷｽﾄﾎﾞｯｸｽの入力検知処理を書いてみょう

```
<!doctype html> <!-- このﾃｷｽﾄﾌｧｲﾙが、html文書であることを明示する。 -->

<html>

 <head>
    <title> ﾃｷｽﾄﾎﾞｯｸｽ ｺﾝﾄﾛｰﾙ </title> <!-- webﾍﾟｰｼﾞのﾀｲﾄﾙ -->
 </head>

 <body>

    <!-- 入力ﾌｫｰﾑ ( action ) -->
    <form id="form1" action="" method="" onsubmit="return false">

      <!-- ﾃｷｽﾄﾎﾞｯｸｽ -->
      <p>
        <label>見出し:</label> <!-- ﾗﾍﾞﾙ -->
        <input type="text" id="textbox1"
                size="30" maxlength="20" value="">
      </p>

    </form>

    <script>

    // ---------------------------------------------------
    // ﾍﾟｰｼﾞが読み込まれた時のｲﾍﾞﾝﾄ処理。

    var prev_value = ""; // 前回の入力値。

    window.onload = function ( event )
    {
```

```
    // 入力ｲﾍﾞﾝﾄのﾊﾝﾄﾞﾗを設定する。
    textbox1.addEventListener( "input",
                               function ( event )
                               {

                                // 入力値を取り出す。
                                let now_value = event.target.value;

                                // 前回の値とは異なっていたら、
                                if ( now_value != prev_value )
                                {

                                  // text changed ｲﾍﾞﾝﾄの処理を呼び出す。
                                     textbox1_OnTextChanged( now_value );

                                  // 前回の値を更新する。
                                  prev_value = now_value;

                                }
                               },
                               true );

}; // end window.onload = function ()

// ---------------------------------------------------------
// ﾃｷｽﾄﾎﾞｯｸｽの入力値が変更されたら、

function textbox1_OnTextChanged( changed_value )
{

    console.log( changed_value );

} // end function textbox1_OnTextChanged( changed_value )

// ---------------------------------------------------------
```

```
    </script>

  </body>

</html>
```

Canvas

```
<!doctype html> <!-- このﾃｷｽﾄﾌｧｲﾙが、html文書であることを明示する。 -->

<html>

 <head>

  <title>Canvasで描画する</title><!-- webﾍﾟｰｼﾞのﾀｲﾄﾙ -->

   <script>

    // ----------------------------------------------------
    // Load処理

    function load()
    {

     // --------------
     // ｷｬﾝﾊﾞｽを取得する。

     var canvas1 = document.getElementById( "mycanvas1" ); // ｷｬﾝﾊﾞｽを取得し、

     var context1 = null;
     context1 = canvas1.getContext( "2d" ); // その2Dｺﾝﾃｷｽﾄを取得する。
     if ( context1 == null ) return; // 未対応なら終了。

    // --------------
    // 四角形

    // 矩形を黒色で塗りつぶす。
     context1.clearRect( 0, 0, 20, 20 ); // (x,y,w,h)

    // 矩形を描画する。
     context1.fillStyle = "rgb( 255, 255, 0 )"; // 塗り色を指定する。 (r,g,b)
     context1.fillRect( 0, 0, 20, 20 );          // 矩形を塗りつぶす。(x,y,w,h)
```

```
// 矩形の輪郭を描画する。
  context1.strokeStyle = "rgb( 255, 0, 0 )"; // 線の色を指定する。(r,g,b)
context1.strokeRect( 0, 0, 20, 20 );       // 矩形の輪郭を塗る。(x,y,w,h)

  // --------------
  // 線分 (ｻﾌﾞﾊﾟｽ) をつなげて、多角形を描画する。

  // 単純な四角形 ( 輪郭のみ )
  context1.strokeStyle = "rgb( 0, 255, 255 )";// 線の色を指定する。(r,g,b)
// ﾊﾟｽを開始する。
// 四角形を指定する。 (x,y,w,h)
context1.closePath();          // ﾊﾟｽを閉じる。
context1.stroke();             // ﾊﾟｽの輪郭を塗る。

  // 線分
  context1.strokeStyle = "rgb( 255, 0, 255 )"; // 線の色を指定する。(r,g,b)
  context1.beginPath();       // ﾊﾟｽを開始する。
  context1.moveTo( 20, 20 ); // 線分の始点を指定する。 (x,y)
  context1.lineTo( 50, 20 );   // 終点を指定する。         (x,y)
  context1.closePath();       // ﾊﾟｽを閉じる。
  context1.stroke();          // ﾊﾟｽを塗る。

  // 線分による四角形 ( 輪郭のみ )
  context1.strokeStyle = "rgb(255,0,255)"; // 線の色を指定する。(r,g,b)
  context1.beginPath();       // ﾊﾟｽを開始する。
  context1.moveTo(20, 20);   // 線分の始点を指定する。     (x,y)
  context1.lineTo(50, 20);     // 終点を指定する。           (x,y)
  context1.lineTo(50, 50);     // 次の線分の終点を指定する。(x,y)
  context1.lineTo(20, 50);     // 次の線分の終点を指定する。(x,y)
  context1.closePath();       // ﾊﾟｽを閉じる。
  context1.stroke();          // ﾊﾟｽを塗る。

  // 円
  context1.beginPath(); // ﾊﾟｽを開始する。
  context1.fillStyle = "rgb( 255, 0, 255 )"; // 塗り色を指定する。(r,g,b)
  context1.arc( 70, 245, 35, 0, Math.PI << 1, false ); // (x,y,半径,開始角,終了角,ﾌﾗｸﾞ)
  // ﾌﾗｸﾞが true なら、反時計回り。
```

```
// ﾌﾗｸﾞが true で、開始角度が 2π 以上なら、円になる。
// ﾌﾗｸﾞが false で 終了角度が 2π 以上なら、円になる。
context1.fill(); // ﾊﾟｽを塗りつぶす。

// 円弧の輪郭 (角の丸み)  http://www.htmq.com/canvas/arcTo.shtml
context1.beginPath();       // ﾊﾟｽを開始する。
context1.moveTo( 20, 20 ); // 円弧の始点を指定する。(x,y)
context1.arcTo( 80, 50, 20, 80, 40 ); // ( 開始x, y, 終了x, y, 半径 )
context1.stroke();  // ﾊﾟｽを塗る。

// 2次ﾍﾞｼﾞｪ曲線   http://www.htmq.com/canvas/quadraticCurveTo.shtml
context1.strokeStyle = "rgb( 255, 0, 255 )"; // 線の色を指定する。(r,g,b)
context1.beginPath();         // ﾊﾟｽを開始する。
context1.moveTo( 50, 100 ); // 線分の始点を指定する。 (x1,y1)
context1.quadraticCurveTo( 150, 20, 250, 100 ); // (x2,y2,x3,y3)
context1.closePath();  // ﾊﾟｽを閉じる。
context1.stroke();      // ﾊﾟｽを塗る。

// 3次ﾍﾞｼﾞｪ曲線   http://www.htmq.com/canvas/bezierCurveTo.shtml
context1.strokeStyle = "rgb(0,255,255)"; // 線の色を指定する。(r,g,b)
context1.beginPath();         // ﾊﾟｽを開始する。
context1.moveTo( 50, 100 ); // 線分の始点を指定する。 (x1,y1)
context1.bezierCurveTo( 100, 20, 200, 20, 250, 100 ); // (x2,y2,x3,y3,x4,y4)
context1.closePath();  // ﾊﾟｽを閉じる。
context1.stroke();      // ﾊﾟｽを塗る。

// --------------
// ﾃｷｽﾄ

context1.font = "48px 'ＭＳ ゴシック'"; // ﾌｫﾝﾄを指定する。

context1.fillStyle = "rgb( 255, 255, 0 )"; // 塗り色を指定する。(r,g,b)
context1.fillText( "てきすと", 60, 60 );    // ﾃｷｽﾄを塗りつぶす。(x,y)

context1.strokeStyle = "rgb( 255, 0, 0 )";  // 線の色を指定する。(r,g,b)
context1.strokeText( "てきすと", 60, 60 );  // ﾃｷｽﾄの輪郭を塗る。(x,y)
```

```
// 横幅を指定して描画する場合は、次のように書く。↓
context1.strokeText( "せまいてきすと", 120, 120, 80 ); // (x,y,w)

var text_width = context1.measureText( "てきすと" ); // ﾃｷｽﾄの横幅を取得する。

// --------------
// 画像

var image1 = new Image(); // 画像ｵﾌﾞｼﾞｪｸﾄを生成せする。
image1.src = "http://bit.ly/2kwCXyW"; // 画像を指定する。
image1.onload = function()
{ context1.drawImage( image1, 100, 200 ); }; // (x,y)
// ( ※ 画像が読み込まれてからでないと、描画できない。)

// 横幅と高さを指定して、画像を描画する。(x,y,w,h)
// context1.drawImage(image1, 120, 120, 30, 30);

// 画像を部分的に描画する。( sx,sy,sw,sh を、dx,dy,dw,dh に描画する。)
// context1.drawImage( image1, 0, 0, 50, 30, 200, 200, 30, 30 );

// 画像ﾃﾞｰﾀを新規作成する。
var image_data1 = context1.createImageData( 100, 80 ); // (w,h)

// 指定した画像と同じｻｲｽﾞの画像ﾃﾞｰﾀを新規作成する。
var image_data2 = context1.createImageData( image_data1 ); // (元画像)

// ｷｬﾝﾊﾞｽ上の指定範囲を画像として切り出して取得する。( はみ出した箇所は黒色になる。)
var image_data3 = context1.getImageData( 0, 0, 100, 75 ); // (x,y,w,h)

// 画像ﾃﾞｰﾀを描画する。
context1.putImageData( image_data3, 505, 50 ); // (x,y)

// var v1 = image1.data[i]; // 色値の配列を取得できる。( RGBA... )
// var w = image1.width;   // 画像の横幅を取得する。
// var h = image1.height;   // 画像の高さを取得する。

// --------------
```

```
// ｸﾞﾗﾃﾞｰｼｮﾝ

// 線形ｸﾞﾗﾃﾞｰｼｮﾝ
context1.beginPath(); // ﾊﾟｽを開始する。
// 線形ｸﾞﾗﾃﾞｰｼｮﾝを指定する。( x0, y0, x1, y1 )
var gradient1 = context1.createLinearGradient( 20, 100, 80, 20 );
gradient1.addColorStop( 0.0, 'rgb(255,0,0)' ); // ｷｰとなる色を追加する。(始点)
gradient1.addColorStop( 0.5, 'rgb(0,255,0)' ); // ｷｰとなる色を追加する。(中央)
gradient1.addColorStop( 1.0, 'rgb(0,0,255)' ); // ｷｰとなる色を追加する。(終点)
context1.fillStyle = gradient1; // ｸﾞﾗﾃﾞｰｼｮﾝを、塗り色として指定する。
context1.rect( 420, 220, 200, 200 ); // 四角形を指定する。
context1.fill(); // 四角形を塗りつぶす。

// 円形ｸﾞﾗﾃﾞｰｼｮﾝ
context1.beginPath(); // ﾊﾟｽを開始する。
// 円形ｸﾞﾗﾃﾞｰｼｮﾝを指定する。( 中心円のx, y, 半径, 外円のx, y, 半径 )
var gradient2 = context1.createRadialGradient( 360, 80, 20, 360, 80, 80 );
gradient2.addColorStop( 0.0 , 'rgb(255,0,0)' ); // ｷｰとなる色を追加する。(中心)
gradient2.addColorStop( 0.5 , 'rgb(0,255,0)' ); // ｷｰとなる色を追加する。(中間)
gradient2.addColorStop( 1.0 , 'rgb(0,0,255)' ); // ｷｰとなる色を追加する。(外側)
context1.fillStyle = gradient2; // ｸﾞﾗﾃﾞｰｼｮﾝを、塗り色として指定する。
 // 円を指定する。 ( x, y, 半径, 開始角, 終了角, ﾌﾗｸﾞ )
context1.arc( 360, 80, 100, 45 / 180 * Math.PI,
                135 / 180 * Math.PI, true );
context1.fill(); // 円を塗りつぶす。

// --------------
// 変形

 // 移動
 context1.translate( 50, 300 );  // (x,y)
 context1.fillRect( 20, 20, 50, 50 );

 // 回転 ( ﾗｼﾞｱﾝ単位の度数 == 度数 / 180 * Math.PI )
 context1.rotate( 45 / 180 * Math.PI ); // ( ﾗｼﾞｱﾝ単位の回転角 )
 context1.fillRect( 0, 10, 50, 50 );
```

```
        // 拡大縮小 ( 縮小する場合は、倍率を小数で指定する。)
        context1.scale( 2, 0.5 ); // ( 横幅の拡縮倍率,縦幅の拡縮倍率 )
        context1.fillRect( 20, 20, 50, 50 );

        // 行列変形 ( 立体的に傾ける ) http://www.htmq.com/canvas/setTransform.shtml
        context1.setTransform( 1, 0.1, 0, 1, 50, 50 );
        // ( 伸縮x, 傾斜y, 傾斜x, 伸縮y, 移動x, 移動y )
        // 変形しない場合は、( 1, 0, 0, 1, 0, 0 )
        context1.fillRect( 20, 20, 50, 50 );

        // --------------
      }

      // ----------------------------------------------------

    </script>

  </head>

  <body onload="load();"> <!-- bodyﾀﾞｸﾞの属性で、Loadﾊﾝﾄﾞﾗを指定しておく。 -->

    <!-- ｷｬﾝﾊﾞｽを配置する。 ( idは、js側でｷｬﾝﾊﾞｽを取得する際に指定する。) -->
    <canvas id="mycanvas1" style="border: 1px solid;"
          width="640" height="480" >このﾌﾞﾗｳｻﾞでは表示できません。</canvas>

  </body>

</html>
```

ｷｬﾝﾊﾞｽの再描画処理を書いてみよう

```
<!doctype html> <!-- このﾃｷｽﾄﾌｧｲﾙが、html文書であることを明示する。 -->

<html>

 <head>

  <title> ｷｬﾝﾊﾞｽに描画する </title> <!-- webﾍﾟｰｼﾞのﾀｲﾄﾙ -->

  </head>

  <body> <!-- bodyﾀﾞｸﾞ -->

   <!-- ｷｬﾝﾊﾞｽを配置する。 ( idは、js側でｷｬﾝﾊﾞｽを取得する際に指定する。) -->
   <canvas id="canvas1" style="border: 1px solid;"
        width="640" height="480" >このﾌﾞﾗｳｻﾞでは表示できません。</canvas>

   <script>

    // ------------------------------------------------------
    // window の 読み込みが終わった直後の処理。

    window.onload = function ()
    {

      // 描画ｲﾍﾞﾝﾄ処理を設定する。
      canvas1.onpaint = canvas1_OnPaint();

      // 描画ｲﾍﾞﾝﾄ処理を設定する。
      canvas1.addEventListener( "mousemove",
                                canvas1_OnMouseMove, false );

    }; // end window.onload = function ()
```

```
// ------------------------------------------------------
// ｷｬﾝﾊﾞｽへの再描画処理。

function canvas1_OnPaint()
{

    // --------------

    var context1 = null;
    context1 = canvas1.getContext( "2d" ); // その2Dｺﾝﾃｷｽﾄを取得する。
    if ( context1 == null ) return; // 未対応なら終了。

    // 矩形を黒色で塗りつぶす。
    context1.clearRect( 0, 0, 20, 20 ); // (x,y,w,h)

    // 矩形を描画する。
    context1.fillStyle = "rgb(255,255,0)"; // 塗り色を指定する。 (r,g,b)
    context1.fillRect( px, py, 20, 20 );   // 矩形を塗りつぶす。 (x,y,w,h)

     // 矩形の輪郭を描画する。
     context1.strokeStyle = "rgb(255,0,0)"; // 線の色を指定する。     (r,g,b)
     context1.strokeRect( px, py, 20, 20 ); // 矩形の輪郭を描画する。(x,y,w,h)

      // --------------

  } // end window.onload = function ()

  // ------------------------------------------------------
  // ｷｬﾝﾊﾞｽがｸﾘｯｸされたら、

  // ｷｬﾝﾊﾞｽがｸﾗｲｱﾝﾄ領域全体であれば、clientX と clientY をそのまま
  // 描画に使うことができますが、大抵の場合は、余白があいています。
  // その差分を引くことによって、ｷｬﾝﾊﾞｽへの描画で使う座標が得られます。

  var px = 0; // ﾏｳｽﾎﾟｲﾝﾀの位置x。( ｷｬﾝﾊﾞｽの座標系 )
  var py = 0; // ﾏｳｽﾎﾟｲﾝﾀの位置y。( ｷｬﾝﾊﾞｽの座標系 )
```

```
    function canvas1_OnMouseMove( event )
    {

      // ｷｬﾝﾊﾞｽの表示範囲を取得する。
      var rect = canvas1.getBoundingClientRect();

      px = event.clientX - rect.left; // ｷｬﾝﾊﾞｽの左上位置までの余白分を引く。
      py = event.clientY - rect.top;  // ｷｬﾝﾊﾞｽの左上位置までの余白分を引く。

      canvas1_OnPaint(); // ｷｬﾝﾊﾞｽを再描画する。

    } // end function canvas1_OnMouseMove( event )

    // -----------------------------------------------------

  </script>

 </body>

</html>
```

Web Storage

・**web**ｻｰﾋﾞｽでは、ﾕｰｻﾞｰ名やﾊﾟｽﾜｰﾄﾞなど、
ﾕｰｻﾞｰごとの個人ﾃﾞｰﾀは、
ﾌﾞﾗｳｻﾞの **Cookie** に記録されていますが、
最大で**4KB**までしか保存することができません。

・「**Web Storage**」 は、ﾕｰｻﾞｰ側のﾛｰｶﾙ環境上に
5MBまでのﾃﾞｰﾀを持つことができます。

・使い方は、ﾊｯｼｭﾃｰﾌﾞﾙと似ていて、とても簡単です。↓

```
<!doctype html> <!-- このﾃｷｽﾄﾌｧｲﾙが、html文書であることを明示する。 -->

<html>

 <head>

  <title>ｾｯｼｮﾝｽﾄﾚｰｼﾞのｻﾝﾌﾟﾙｺｰﾄﾞ</title> <!-- webﾍﾟｰｼﾞのﾀｲﾄﾙ -->

  <!-- javascript -->
  <script>

   // ｾｯｼｮﾝｽﾄﾚｰｼﾞを取得する。
   var ss = sessionStorage;

   // ｷｰﾍﾟｱを追加する。
   ss.setItem( 'name', '太郎' );

   var count = ss.length; // ｷｰﾍﾟｱの総数を取得する。

   // ｷｰﾍﾟｱの値を上書きする。
   ss.setItem( 'name', '花子' );
```

```
  // ｷｰﾍﾟｱの値を取得する。
  var v1 = ss.getItem( 'name' );

  // ｷｰﾍﾟｱを削除する。
  ss.removeItem( 'name' );

  // すべてのｷｰﾍﾟｱを削除する。
  ss.clear();

 </script>

 </head>

 <body>
 </body>

</html>
```

・「**session**Storage」のところを、「**local**Storage」にすると、
ﾍﾟｰｼﾞやﾌﾞﾗｳｻﾞを閉じた後も、ﾃﾞｰﾀを残すことができます。

・ｾｯｼｮﾝｽﾄﾚｰｼﾞは、**1**ﾍﾟｰｼﾞごとに割り当てられる保存領域で、
Cookie のように、他のﾍﾟｰｼﾞとﾃﾞｰﾀを共有することはできません。

・ﾛｰｶﾙｽﾄﾚｰｼﾞは、**1**ｵﾘｼﾞﾝごとに割り当てられる保存領域で、
他のﾍﾟｰｼﾞとﾃﾞｰﾀを共有することができます。

・「ｵﾘｼﾞﾝ」というのは、「ﾌﾟﾛﾄｺﾙ**:**//ﾄﾞﾒｲﾝ**:**ﾎﾟｰﾄ番号」のことです。

IISでwebｻｲﾄを作ってみよう

・自宅のﾊﾟｿｺﾝ上に、**web**ｻｲﾄを作ってみましょう。

・通常、webｻｲﾄを自作する場合は、
VPSなどのﾚﾝﾀﾙｻｰﾋﾞｽを使うのが一般的ですが、
今回は、ｻｰﾊﾞｰとﾌﾞﾗｳｻﾞとの通信を確認するだけですから、
最もお手軽な、**IIS**を使った方法で作ってみます。

・**IIS**は、Windowsに最初から含まれているもので、
少し設定を変更するだけで、ｲﾝｽﾄｰﾙされます。↓

https://creativeweb.jp/personal-site/iis/

・**IIS**の管理画面を開くには、**[** ｺﾝﾄﾛｰﾙﾊﾟﾈﾙ **]** → **[** 管理ﾂｰﾙ **]**
→ **[** ｲﾝﾀｰﾈｯﾄｲﾝﾌｫﾒｰｼｮﾝｻｰﾋﾞｽ **(IIS)** ﾏﾈｰｼﾞｬｰ **]** の順にｸﾘｯｸします。

(※ ちなみに、この項目は、先だっての設定画面
「**Windows** の機能の有効化または無効化」で、
管理ﾂｰﾙのところにﾁｪｯｸを入れていないと、追加されません。)

・「ｻｲﾄ」というﾌｫﾙﾀﾞをｸﾘｯｸすると、
「**Default Web Site**」というｱｲｺﾝが出てきます。

・これを右ｸﾘｯｸして、**[** **web**ｻｲﾄの管理 **]** ▸ **[** 詳細設定 **]** をｸﾘｯｸします。

・ﾀﾞｲｱﾛｸﾞ画面が表示されたら、「物理ﾊﾟｽ」のところに、
webｻｲﾄのﾙｰﾄﾌｫﾙﾀﾞとして使いたいﾌｫﾙﾀﾞへのﾊﾟｽを入力してください。

・このﾌｫﾙﾀﾞの中に、たとえば、「**index.html**」というﾌｧｲﾙを置いて、
webﾌﾞﾗｳｻﾞを起動し、**URL**ｱﾄﾞﾚｽの入力ﾎﾞｯｸｽに、
「**http://localhost/index.html**」と入力してみてください。

・Enterｷｰを押すと、**index.html** が表示されたはずです。

・ちなみに、この**web**ｻｲﾄを、別のﾊﾟｿｺﾝから閲覧する場合は、
この「**localhost**」というﾄﾞﾒｲﾝでは、つながりません。

・理由はｶﾝﾀﾝで、その別のﾊﾟｿｺﾝのﾛｰｶﾙ環境上には、
このwebｻｲﾄが存在していないからです。

・ひとまず、この「localhost」という言葉の意味は、
自分のﾊﾟｿｺﾝのことだと考えておけばいいでしょう。

IISを再起動させる手順

・[ｺﾝﾄﾛｰﾙﾊﾟﾈﾙ]→[管理ﾂｰﾙ]→**[**ｲﾝﾀｰﾈｯﾄｲﾝﾌｫﾒｰｼｮﾝｻｰﾋﾞｽ **(IIS)** ﾏﾈｰｼﾞｬｰ**]** をｸﾘｯｸすると、
IISの管理画面が開きます。

・この管理画面の左側のﾂﾘｰﾋﾞｭｰにある、

「ｻｲﾄ」
　|
　　+- **[Default Web Site]**

　　で右ｸﾘｯｸして、「**Web**ｻｲﾄの管理」をｸﾘｯｸすると、
　　「再起動」をｸﾘｯｸすると、IISが再起動します。

「**HTTP** ｴﾗｰ**404.3 - Not Found**」の対処法

・指定したURLに、ﾌｧｲﾙがない場合は、
「HTTP ｴﾗｰ 404」というｴﾗｰﾒｯｾｰｼﾞが表示されますが、
「HTTP ｴﾗｰ404.3 - Not Found」と表示された場合は、
以下の手順で、設定を変更して下さい。↓

・[ｺﾝﾄﾛｰﾙﾊﾟﾈﾙ]→[ﾌﾟﾛｸﾞﾗﾑと機能]の画面で、
[Windowsの機能の有効化または無効化]をｸﾘｯｸしてﾀﾞｲｱﾛｸﾞを開く。

・このﾀﾞｲｱﾛｸﾞ画面で、

[ｲﾝﾀｰﾈｯﾄｲﾝﾌｫﾒｰｼｮﾝｻｰﾋﾞｽ」
　|

+-「World Wide Webｻｰﾋﾞｽ]
|
+-「ｱﾌﾟﾘｹｰｼｮﾝ開発機能」
| |
| +-「ASP.NET」 ← ﾁｪｯｸを入れる。
|
+-「ｾｷｭﾘﾃｨｰ」
|
+-「Windows認証」 ← ﾁｪｯｸを入れる。

・それから、IIS管理画面の機能ﾋﾞｭｰで、
「ハンドラー マッピング」をｸﾘｯｸし、
画面右端の「ﾏﾈｰｼﾞﾊﾝﾄﾞﾗの追加」をｸﾘｯｸします。

・「要求ﾊﾟｽ」には、「*.fbx」など、ﾌｧｲﾙ拡張子を入力します。

・次に、機能ﾋﾞｭｰの「**MIME**の種類」をｸﾘｯｸし、
画面右端の「追加」をｸﾘｯｸします。

・「ﾌｧｲﾙ名の拡張子」には、「**.fbx**」などﾌｧｲﾙ拡張子を入力し、
「**MIME**の種類」には、「**application/octet-stream**」を入力します。
(これはbinﾌｧｲﾙなどで使う汎用的なものです。)

・それでもﾀﾞﾒな場合は、ﾌｧｲﾙ拡張子を「xml」など、一般的なものに変えてみて下さい。

Web Worker

・下記の2つのﾌｧｲﾙを、前回設定した「**Default Website**」の
物理ﾊﾟｽとして指定したﾌｫﾙﾀﾞに置いてください。

・そして、**web**ﾌﾞﾗｳｻﾞを開いて、**URL**の入力欄に、
「**http://localhost/web_worker.html** 」と入力して、
Enterｷｰを押してください。

```
<!-- web_worker.html -->
<!DOCTYPE html>
 <html>
  <head>
   <title> Web Worker API Test </title>
  </head>
  <body>
   <script>

  // --------------------
  // 外部jsを読み込んで、ﾜｰｶｰを生成する。

  var worker = new Worker( "web_worker_sub.js" );

  // --------------------
  // ﾜｰｶｰが message を送信した際の処理。( ﾒｯｾｰｼﾞを表示する。)

  worker.onmessage = function( e ) // ﾜｰｶｰに、 messageｲﾍﾞﾝﾄのﾊﾝﾄﾞﾗを追加する。
  {
     console.log( "ﾒｲﾝｽﾚｯﾄﾞ側のmessageﾊﾝﾄﾞﾗが呼ばれました。" );
     console.log( "渡された値は、" + e.data + "です。" );
  };

  // --------------------
  // ﾜｰｶｰでｴﾗｰが発生した際の処理。

  worker.onerror = function( e )
```

```
  {
    console.log( "ｴﾗｰ: " + e.message + "\n" ); // ｴﾗｰﾒｯｾｰｼﾞを出力する。

    throw e; // 例外をｽﾛｰする。
  };

  // --------------------

  console.log( "ﾒｲﾝｽﾚｯﾄﾞ: ﾒｯｾｰｼﾞ送信の直前" );

  worker.postMessage( [1,2] ); // ﾜｰｶｰに、ﾒｯｾｰｼﾞ「1,2」を送信してみる。

  console.log( "ﾒｲﾝｽﾚｯﾄﾞ: ﾒｯｾｰｼﾞ送信の直後" );

  // --------------------

 </script>

 </body>
</html>
```

```
// web_worker_sub.js

onmessage = function( e )
{
 console.log( "ﾜｰｶｰ側のmessageﾊﾝﾄﾞﾗが呼ばれました。" );

 console.log( "渡された値は、" + e.data[0] + " と " + e.data[1] + "です。" );

 postMessage( e.data[0] + e.data[1] ); // 渡された値を、計算して、ﾒｲﾝｽﾚｯﾄﾞ側へ返信する。

}
```

・ご利用のﾌﾞﾗｳｻﾞがChromeの場合は、ﾍﾟｰｼﾞ上を右ｸﾘｯｸして、

「検証」をｸﾘｯｸすると、画面右側に、開発ﾂｰﾙが表示されます。

・ｺﾝｿｰﾙ上に表示される出力結果は、下記の通りです。↓

ﾒｯｾｰｼﾞ送信の直前
ﾒｯｾｰｼﾞ送信の直後
Navigated to http://localhost/web_worker.html
web_worker_sub.js:3 ﾜｰｶｰ側のmessageﾊﾝﾄﾞﾗが呼ばれました。
web_worker_sub.js:5 渡された値は、1 と 2です。
web_worker.html:21 ﾒｲﾝｽﾚｯﾄﾞ側のmessageﾊﾝﾄﾞﾗです。
web_worker.html:22 渡された値は、3です。

Web Audio

・効果音を再生してみましょう。

・音声ﾌｧｲﾙは、下記のｻｲﾄなどからﾀﾞｳﾝﾛｰﾄﾞするなどして、
お好みのものを用意してください。
http://www.hmix.net/music_gallery/music_top.htm

・下記のﾍﾟｰｼﾞを開くと、「**sample1.mp3**」がﾙｰﾌﾟ再生されます。
・また、ﾍﾟｰｼﾞ上でｷｰを押すと、「**picon.wav**」が再生されます。

```
<!doctype html> <!-- このﾃｷｽﾄﾌｧｲﾙが、html文書であることを明示する。-->

<html>
  <head>
    <title> Web Audio API Test </title> <!-- webﾍﾟｰｼﾞのﾀｲﾄﾙ -->

      <script>

// ------------------------------------------------------
// ｵｰﾃﾞｨｵ・ｺﾝﾃｷｽﾄを生成する。( すべてのｻｳﾝﾄﾞを管理するｵﾌﾞｼﾞｪｸﾄ。)

// WebKit系ﾌﾞﾗｳｻﾞに対応させる。
window.AudioContext = window.AudioContext || window.webkitAudioContext;

var audio_context; // ｵｰﾃﾞｨｵ・ｺﾝﾃｷｽﾄ


    try
    {
        audio_context = new AudioContext(); // ｺﾝﾃｷｽﾄを生成する。
    }
    catch( ex )
    {
        alert( "このﾌﾞﾗｳｻﾞは、Web Audio API に対応していません。" );
    }

// ------------------------------------------------------
// ｻｳﾝﾄﾞﾊﾞｯﾌｧを読み込み、ｻｳﾝﾄﾞｿｰｽを作成する。

function playSound( file_name_ , is_loop_ )
{
  // ----------------------
```

```
  // HTTPﾘｸｴｽﾄを利用して、ｻｳﾝﾄﾞﾊﾞｯﾌｧを取得する。

  var req = new XMLHttpRequest(); // HTTPﾘｸｴｽﾄを生成する。

  req.open( 'GET' , file_name_ , true ); // ﾘｸｴｽﾄを開く。

  req.responseType = 'arraybuffer';  // ﾚｽﾎﾟﾝｽﾀｲﾌﾟには、配列ﾊﾞｯﾌｧを指定しておく。

  // ----------------------
  // ﾘｸｴｽﾄの onload 処理

  req.onload = function()
  {
      // ｵｰﾃﾞｨｵﾃﾞｰﾀをﾃﾞｺｰﾄﾞする。
      audio_context.decodeAudioData(
                                          req.response, // ｵｰﾃﾞｨｵﾃﾞｰﾀ
                                         function( buffer )
                                         {
                                           // ｺﾝﾃｷｽﾄからｻｳﾝﾄﾞｿｰｽを生成する。
                                           var source =
                                           audio_context.createBufferSource();

                                           // ｻｳﾝﾄﾞｿｰｽに、ｻｳﾝﾄﾞﾊﾞｯﾌｧを設定する。

                                           source.buffer = buffer;

                                           // ｺﾝﾃｷｽﾄに接続する。
                                           source.connect( audio_context.destination );

                                           source.loop = is_loop_; // ﾙｰﾌﾟﾌﾗｸﾞ。

                                           source.start( 0 ); // 再生する。
                                         }
                                         , null ); // 任意のｴﾗｰ処理。
  };

  // ----------------------

  req.send( null ); // ﾘｸｴｽﾄを送信する。

  // ----------------------
};

// -------------------------------------------------
// Load ﾊﾝﾄﾞﾗ
```

```
window.onload = function()
{
    playSound( "sample1.mp3", true ); // ｻｳﾝﾄﾞを再生する。
};

// ------------------------------------------------
// KeyPress ﾊﾝﾄﾞﾗ

window.onkeypress = function()
{
     playSound( "picon.wav", false ); // ｻｳﾝﾄﾞを再生する。
};

// ------------------------------------------------

     </script>

   </head>

   <body>
   </body>
</html>
```

・さて、少し読みにくいのが、この
HTTPﾘｸｴｽﾄ の **onload** ｲﾍﾞﾝﾄの処理のところで、
decodeAudioData ﾒｿｯﾄﾞを呼んでいるんですが、
これの**引数1**には、ｵｰﾃﾞｨｵﾃﾞｰﾀを渡します。

・これは、**HTTP**ﾘｸｴｽﾄ(要求)を送信した結果として
返信されるﾚｽﾎﾟﾝｽ(反応)です。

・次に、そのとなりの**引数2**は、
ｵｰﾃﾞｨｵﾃﾞｰﾀをﾃﾞｺｰﾄﾞした直後に実行される処理で、
引数1は、ﾃﾞｺｰﾄﾞ済みのｵｰﾃﾞｨｵﾊﾞｯﾌｧです。

```
<!doctype html> <!-- このﾃｷｽﾄﾌｧｲﾙが、html文書であることを明示する。 -->

<html>
  <head>

    <title> ｸﾛｽﾄﾞｷｭﾒﾝﾄﾒｯｾｰｼﾞﾝｸﾞ (送信側のﾍﾟｰｼﾞ) </title> <!-- webﾍﾟｰｼﾞのﾀｲﾄﾙ -->

     <!-- javascript -->
     <script> <!-- 直接書き込む。 -->

     // ----------------------------------------
     // 送信ﾎﾞﾀﾝがｸﾘｯｸされた時のｲﾍﾞﾝﾄ処理。( 別のﾍﾟｰｼﾞへ、ﾒｯｾｰｼﾞを送信する。)

      function button1_OnClick( event )
      {
       // このﾍﾟｰｼﾞ上に配置された iframe ﾀｸﾞ (ﾌﾚｰﾑｳｨﾝﾄﾞｳ) を取得する。
       var frame1 = document.getElementById( "frame1");

       // iframe ﾀｸﾞに表示されているﾍﾟｰｼﾞの window を取得する。
       var another_window1 = frame1.contentWindow;

       // 送信するﾒｯｾｰｼﾞを入力するﾃｷｽﾄﾎﾞｯｸｽを取得する。( input ﾀｸﾞ )
       var textbox1 = document.getElementById( "textbox1" );
       var msg_text = textbox1.value; // 入力された値を取り出す。

       // 別のwindowへ、ﾒｯｾｰｼﾞを送信する。( 引数2では、ｵﾘｼﾞﾝを指定している。)
       another_window1.postMessage( msg_text,
                              "http://localhost/cross_doc_msg2.html" );

       textbox1.value = ""; // 入力値をｸﾘｱする。
       textbox1.select();  // ﾃｷｽﾄﾎﾞｯｸｽを選択する。( 入力ﾌｫｰｶｽを戻す )
      }
```

```
    // ----------------------------------------

     </script>

    </head>

    <!-- body ﾀｸﾞ が読み込まれる時に、 URL表示ﾃｷｽﾄﾎﾞｯｸｽに、
         このﾍﾟｰｼﾞ のURLを入力しておく。 -->
    <body onload="form1.textbox2.value=location.href;">

     <!-- ﾌｫｰﾑ -->
     <form name="form1" id="form1">

        <!-- 別ﾍﾟｰｼﾞ へ送信するﾒｯｾｰｼﾞ を入力するﾃｷｽﾄﾎﾞｯｸｽ -->
        メッセージ：<input type="text" id="textbox1"
                           size="60" value="こんにちは">

        <!-- 送信ﾎﾞﾀﾝ -->
        <input type="button" id="button1" value="送信"
                  onclick="button1_OnClick();"><br>
              <!-- ｸﾘｯｸ時のｲﾍﾞﾝﾄ処理を設定しておく。 -->

        <!-- 別のﾍﾟｰｼﾞ を表示するﾌﾚｰﾑｳｨﾝﾄﾞｳ -->
        <iframe id="frame1"
                    src="http://localhost/cross_doc_msg2.html"
                    width="600" height="200">
          </iframe><br>

        <!-- このﾍﾟｰｼﾞ のURLを表示するﾃｷｽﾄﾎﾞｯｸｽ。 -->
        URL:<input type="url" id="textbox2" size="45" readonly="true">
    </form>

  </body>
</html>
```

```
<!doctype html> <!-- このﾃｷｽﾄﾌｧｲﾙが、html文書であることを明示する。 -->

<html>
  <head>

    <title> ｸﾛｽﾄﾞｷｭﾒﾝﾄﾒｯｾｰｼﾞﾝｸﾞ (受信側のﾍﾟｰｼﾞ) </title> <!-- webﾍﾟｰｼﾞのﾀｲﾄﾙ -->

      <!-- javascript -->
      <script>

      // ------------------------------------------------
      // ｸﾞﾛｰﾊﾞﾙ空間

      // このﾍﾟｰｼﾞの window が、ﾒｯｾｰｼﾞを受信した時のｲﾍﾞﾝﾄ処理を設定しておく。
      window.addEventListener( "message",
                               window_OnRecieveMessage, true );

      // ------------------------------------------------
      // ﾒｯｾｰｼﾞを受信した時のｲﾍﾞﾝﾄ処理。

      function window_OnRecieveMessage( event )
       {
        // ﾃｷｽﾄｴﾘｱを取得する。( input要素 )
        var text_area1 = document.getElementById( "text_area1" );

        // event には、受信したﾒｯｾｰｼﾞを含むﾃﾞｰﾀが格納されている。

        // event.origin (送信元のｵﾘｼﾞﾝ) が、許可されているｵﾘｼﾞﾝなら、
        if ( event.origin == "http://localhost" )
        {
          // event.data (受信ﾒｯｾｰｼﾞ) を、ﾃｷｽﾄｴﾘｱに表示する。
          text_area1.value += event.data + "\r\n";
```

```
    // ﾃｷｽﾄｴﾘｱを、最後の行まで、ｽｸﾛｰﾙさせる。
    text_area1.scrollTop = text_area1.scrollHeight;

    // さらに、source ﾌﾟﾛﾊﾟﾃｨ ( 送信元windowへの参照 ) で、
    // 受信した事を、送信元のﾍﾟｰｼﾞ側に、返信する事もできる。
    event.source.textbox1.value = "受信しましたよ！";

  } // end if
  else
  {
    // e.data (受信ﾒｯｾｰｼﾞ ) を、ﾃｷｽﾄｴﾘｱに表示する。
    text_area1.value = "失敗\r\n";

  } // end else

 } // end function window_OnRecieveMessage

 // ------------------------------------------------

 </script>

</head>

 <!-- body ﾀｸﾞ が読み込まれる時に、URL表示ﾃｷｽﾄﾎﾞｯｸｽに、
      このﾍﾟｰｼﾞのURLを入力しておく。 -->
<body onload="form1.textbox2.value=location.href;">

 <!-- ﾌｫｰﾑ -->
 <form name="form1">

  <!--受信したﾒｯｾｰｼﾞを表示するﾃｷｽﾄｴﾘｱ-->
  <textarea name="text_area1" id="text_area1" rows="5" cols="70">
  </textarea><br>

  <!-- このﾍﾟｰｼﾞのURLを表示するﾃｷｽﾄﾎﾞｯｸｽ。 -->
  URL:<input type="url" id="textbox2" size="45" readonly="true">
```

```
    </form>

  </body>
</html>
```

ﾍﾟｰｼﾞにｽﾀｲﾙｼｰﾄを適用する

・さて、HTML5からは、ｽﾀｲﾙ系の要素ﾀｸﾞが、ほぼすべて廃止され、
ﾍﾟｰｼﾞの外観は、ｽﾀｲﾙｼｰﾄで定義するようになりました。

・これはやはり、ﾃﾞｰﾀと外観とは、別々に書いた方が、
あとで修正しやすい、という事情からのようです。

・ｽﾀｲﾙｼｰﾄの書き方については、次回に譲るとして、
今回は、ｽﾀｲﾙｼｰﾄをﾍﾟｰｼﾞに適用する方法を見ていきます。↓

```
<!doctype html> <!-- このﾃｷｽﾄﾌｧｲﾙが、html文書であることを明示する。-->

<html>

 <head>

  <title> ﾍﾟｰｼﾞにｽﾀｲﾙｼｰﾄを適用する </title> <!-- webﾍﾟｰｼﾞのﾀｲﾄﾙ -->


  <!-- このﾍﾟｰｼﾞで使用しているｽﾀｲﾙｼｰﾄが、CSS形式であることを明示する。
     ( このﾀｸﾞは省略してもよく、形式は、ﾌﾞﾗｳｻﾞによって自動的に識別される。) -->
  <meta name="Content-Style-Type" content="text/css">


  <!-- 外部のcssﾌｧｲﾙをﾘﾝｸする。↓
      これにより、ﾌｧｲﾙ内で定義されたｽﾀｲﾙが、このﾍﾟｰｼﾞ上で有効になる。-->
  <link rel="stylesheet" type="text/css" href="sample1.css">


  <!-- このﾍﾟｰｼﾞ上で、ｽﾀｲﾙを定義することもできる。↓ -->
  <style type="text/css">

     *{background-color: rgb(255,128,0);} /* ( red green blue ) */
```

```
  </style>


</head>

<body>


  <!-- 要素ﾀｸﾞごとに、style属性でｽﾀｲﾙを設定する場合は、次のように書く。↓ -->


  <!-- ｲﾝﾗｲﾝ -->
  <span style="color: #00FFFF">あいうえお</span>


  <!-- ﾌﾞﾛｯｸ(改行される) -->
  <div style="color: #FF00FF">
    かきくけこ
    さしすせそ
  </div>


</body>

</html>
```

ｽﾀｲﾙｼｰﾄまとめ

・**CSS**の文法は、至って簡単です。↓

```
ｾﾚｸﾀ {ﾌﾟﾛﾊﾟﾃｨ:値;}              /* ﾌﾟﾛﾊﾟﾃｨを一つだけ設定する。 */
ｾﾚｸﾀ {ﾌﾟﾛﾊﾟﾃｨ:値;ﾌﾟﾛﾊﾟﾃｨ:値;} /* 複数のﾌﾟﾛﾊﾟﾃｨを設定する。   */
```

・「セレクタ」　... ｽﾀｲﾙを適用する対象。(指定した属性値を持つ要素ﾀｸﾞのみ、など)
・「プロパティ」 ... 適用するｽﾀｲﾙの種類。(文字のｻｲｽﾞや、背景色など)

・よく使うｾﾚｸﾀは、

```
*{}                    ... すべての要素ﾀｸﾞ。
要素ﾀｸﾞ名{}             ... 指定された名前の要素ﾀｸﾞすべて。
要素ﾀｸﾞ名[属性名]{}      ... 指定された属性を持つ指定の要素ﾀｸﾞすべて。
要素ﾀｸﾞ名[属性名=値]{}   ... 指定された属性に、指定した値を持つ指定の要素ﾀｸﾞすべて。
要素ﾀｸﾞ名[属性名*=値]{} ... 指定した属性に、指定した値を含む指定の要素ﾀｸﾞすべて。
.ｸﾗｽ名{}               ... class属性の値が、指定されたｸﾗｽ名の要素ﾀｸﾞすべて。
#id名{}                ... id属性の値が、指定されたid名の要素ﾀｸﾞすべて。
```

・よく使うﾌﾟﾛﾊﾟﾃｨは、

カラー系

```
color: #FF0000                      ... 文字色
background-color: #FF00FF           ... 背景色
background-image: url("ahiru.png")  ... 背景画像
```

フォント系

font-style: normal ... ﾌｫﾝﾄｽﾀｲﾙを標準のものにする。(ﾃﾞﾌｫﾙﾄ)
font-style: italic ... ﾌｫﾝﾄｽﾀｲﾙをｲﾀﾘｯｸ体にする。 (斜体は、**oblique**)
font-weight: bold ... ﾌｫﾝﾄを太字にする。 (標準は、**normal**)
font-size: 12px; } ... ﾌｫﾝﾄｻｲｽﾞを12ﾋﾟｸｾﾙにする。
font-family: "ＭＳ ゴシック"; ... ﾌｫﾝﾄを指定する。

テキスト系

line-height: 20px ... 行の高さが20ﾋﾟｸｾﾙになる。
(行間の余白の高さは、これからﾌｫﾝﾄの高さを引いたもの。)
text-align: left ... 水平位置揃えを左寄せにする。(均等割付なら、**justify**)
vertical-align: top ... 垂直位置揃えを植え寄せにする。(**top middle botom** その他)
text-decoration: underline ... 下線がつく。(解除は **none**。**line-through** で打ち消し線。)
letter-spacing: 10px ... 文字の間隔を10ﾋﾟｸｾﾙにする。(ﾃﾞﾌｫﾙﾄは、**normal**)

サイズ系 (ﾃｰﾌﾞﾙ、画像、ｺﾝﾄﾛｰﾙなど)

width: 640px ... 横幅を640ﾋﾟｸｾﾙにする。
height: 480px ... 高さを480ﾋﾟｸｾﾙにする。

マージン系 (上下左右の余白)

padding: 10px 20px 30px 40px; ... 上、右、下、左の余白幅。

ボーダー系 (表の中枠の線)

border-style: solid; ... ﾎﾞｰﾀﾞｰのｽﾀｲﾙを、一本線にする。
(非表示なら、**none**。破線なら、**dashed**。点線なら、**dotted**。)
border-color: red; ... ﾎﾞｰﾀﾞｰの色を、赤色にする。
border-width: 1px; ... ﾎﾞｰﾀﾞｰの太さを、1ﾋﾟｸｾﾙにする。

アウトライン系 (表の外枠の線)

outline-style: solid ... ｱｳﾄﾗｲﾝのｽﾀｲﾙを、一本線にする。

(非表示なら、**none**。破線なら、**dashed**。点線なら、**dotted**。)

outline-color: **red**;　... ｱｳﾄﾗｲﾝの色を、赤色にする。

outline-width: 1px;　... ｱｳﾄﾗｲﾝの太さを、1ﾋﾟｸｾﾙにする。

テーブル系

table-layout: **fixed**　　... ﾃｰﾌﾞﾙのﾚｲｱｳﾄを、固定にする。(自動調整なら、**auto**)

border-collapse: **separate**　... ｾﾙの輪郭を空ける。　　(通常は、**collapse**)

border-spacing: 2px　　... ｾﾙの間隔幅を、2ﾋﾟｸｾﾙにする。(値を2つ指定すると、左右　上下)
