Appendix

JavaScriptのサンプル集

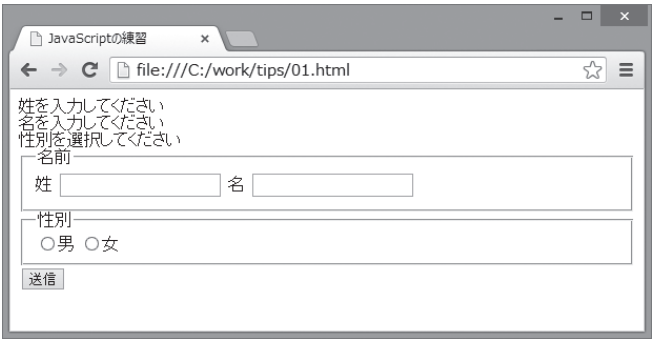
すぐに使える便利なサンプル集です

Tips 01 実用的な入力チェックを行なう

難易度★☆☆

フォームの入力チェック方法は、第9章のリスト9.5（222ページ）で紹介しました。しかし、エラーメッセージをダイアログボックスで表示する方法はあまり実用的とは言えませんので、ここではエラーメッセージをページ上に表示する方法を紹介します。また、スクリプトはjQueryを利用して記述します。

■実行結果



ソースコード

HTML 01.html

```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title>JavaScriptの練習</title>
  <script src="http://ajax.googleapis.com/ajax/libs/
jquery/1.9.1/jquery.min.js"></script>
  <script src="main01.js"></script>
</head>
 <body>
  <!-- エラー表示エリア -->
  <div id="errorArea"></div>
```

```
  <form name="frm1" action="xxx">
   <fieldset>
    <legend>名前</legend>
    <label>姓
     <input type="text" name="lastname">
    </label>
    <label>名
     <input type="text" name="firstname">
    </label>
   </fieldset>
   <fieldset>
    <legend>性別</legend>
    <label>
     <input type="radio" name="gender" value="m">男
    </label>
    <label>
     <input type="radio" name="gender" value="f">女
    </label>
   </fieldset>
   <input type="button" id="sendBtn" value="送信">
  </form>
 </body>
</html>
```

JavaScript　main01.js

```
01: $(function(){
02:   $("#sendBtn").on("click", inputCheck);
03: });
04: function inputCheck(){
05:   var errors = [];
06:
07:   if(!$("input[name='lastname']").val()) {
08:     errors.push("姓を入力してください");
09:   }
10:   if (!$("input[name='firstname']").val()) {
11:     errors.push("名を入力してください");
12:   }
13:   if (!$("input[name='gender']:checked").val()) {
14:     errors.push("性別を選択してください");
15:   }
16:
17:   if(errors.length > 0) {
18:     $("#errorArea").html(errors.join("<br>"));
19:   } else {
20:     $("form[name='frm1']").submit();
21:   }
22: }
```

解説

［送信］ボタンがクリックされると、inputCheck関数が実行されます。まず、5行目でerrorsに空の配列を代入しています。errorsはエラーメッセージを保持する配列です。発生したエラーの件数分、配列の要素が追加されます。

7行目から9行目にかけて、姓が入力されているかどうかをチェックしています。テキストボックスが未入力の場合、valメソッドは空文字（""）を返します。JavaScriptでは、空文字はfalseとして判定されます。条件式の先頭に!演算子が付いているので、姓が未入力の場合にifブロック内が実行され、配列errorsにエラーメッセージが追加されます。

▼姓が未入力の場合

```
if(!$("input[name='lastname']").val()) {
```

↓ テキストボックスに入力された文字列を取得

```
if(!"") {
```

↓ 空文字はfalseとして解釈される

```
if(!false) {
```

↓ 論理値を反転

```
if(true) {
```

10行目から12行目も、前述の処理と同様のチェックを行なっています。

13行目では、オプションボタン（ラジオボタン）がチェックされているかどうかを調べています。チェックされているオプションボタンが存在しない場合、セレクタに該当する要素は存在しません。よってvalメソッドを実行しても「undefined」が返却されます。undefinedも空文字と同様、falseとして扱われます。

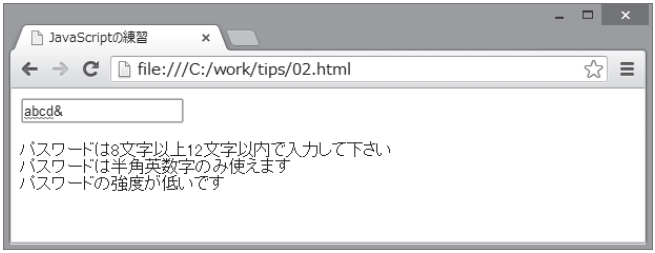
17行目から21行目では、3つのチェック項目でエラーが発生したかどうかを調べています。エラーが発生しているときは、配列errorsに1件以上のエラーメッセージが追加されています。その場合、複数のエラーメッセージ群を
で区切って1つの文字列として連結し、エラー表示エリアに挿入しています。

Tips 02 パスワードの入力チェックを即時実行する

難易度★☆☆

先のサンプルのようにボタンを押したタイミングではなく、テキストボックスに文字列を入力すると即座にチェックする方法を紹介します。例として、パスワードの文字列長、文字種、強度をチェックします。

■実行結果



ソースコード

HTML　02.html

```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title>JavaScriptの練習</title>
  <script src="http://ajax.googleapis.com/ajax/libs/
jquery/1.9.1/jquery.min.js"></script>
  <script src="main02.js"></script>
 </head>
 <body>
  <input type="text" id="password">
  <p id="message"></p>
 </body>
</html>
```

JavaScript main02.js

```
01: $(function() {
02:   $("#password").on("keyup", inputCheck);
03: });
04: function inputCheck() {
05:   var errors = [];
06:   var password = $("#password").val();
07:
08:   if(password.length < 8 || password.length > 12) {
09:     errors.push("パスワードは8文字以上12文字以内で入力して下さい");
10:   }
11:   if(password.match(/[^0-9a-zA-Z]/)) {
12:     errors.push("パスワードは半角英数字のみ使えます");
13:   }
14:   if(!(password.match(/[0-9]/) && password.
    match(/[a-z]/) && password.match(/[A-Z]/))) {
15:     errors.push("パスワードの強度が低いです");
16:   }
17:
18:   $("#message").html(errors.join("<br>"));
19: }
```

解説

テキストボックス上で文字列が入力されたタイミングで処理を実行するには、keyupイベントを使います。keyupは、キーボードから1文字分の入力が行なわれた際に実行されるイベントです。

8行目では、文字列長をチェックしています。lengthプロパティは、配列の場合は要素数でしたが、文字列の場合は文字列の長さを表わします。

11行目では、文字種をチェックしています。以下の正規表現は、半角英数字以外の文字にマッチするかどうかを調べています。[ ] の先頭に ^ を付けると、以降のパターンと一致しない文字にマッチします。

```
password.match(/[^0-9a-zA-Z]/)
```

0-9、a-z、A-Z以外の文字が含まれているかどうか

14行目では、パスワードの強度をチェックしています。本来、強度チェックでは「1234」などの単純な文字列もはじかなければなりませんが、今回は簡易

版として大文字、小文字、数字の3種類がすべて含まれていることのみを条件としています。以下のif文は、3種の文字すべてが含まれている、という条件を否定しています。

```
if(!(password.match(/[0-9]/) && password.match(/[a-z]/) &&
password.match(/[A-Z]/))) {
```

数字が含まれる

小文字が含まれる

大文字が含まれる

どれか1つでも満たさなければfalse

↓

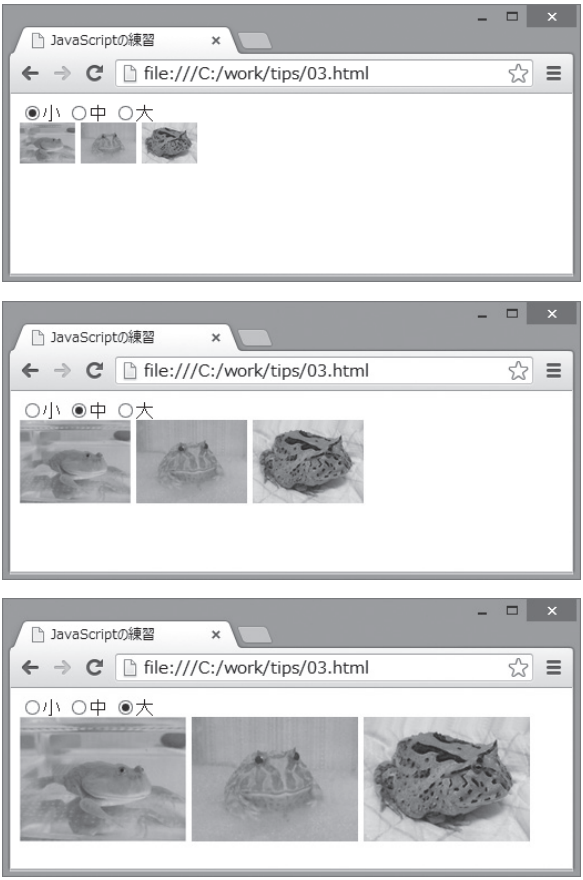
全体が!演算子で囲まれているので、論理値が反転されtrueとなる

Tips 03 サムネイル画像のサイズを切り替える

難易度★☆☆

オプションボタンで選択されたサイズに応じて、画像の拡大率を変更します。

■実行結果



ソースコード

HTML　03.html

```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title>JavaScriptの練習</title>
  <script src="http://ajax.googleapis.com/ajax/libs/
jquery/1.9.1/jquery.min.js"></script>
  <script src="main03.js"></script>
  <link rel="stylesheet" type="text/css" href="style03.css">
 </head>
 <body>
  <div>
   <input type="radio" name="size" value="small" checked>小
   <input type="radio" name="size" value="medium">中
   <input type="radio" name="size" value="large">大
  </div>
  <div id="photo">
   <img src="images/frog1.jpg" class="small">
   <img src="images/frog2.jpg" class="small">
   <img src="images/frog3.jpg" class="small">
  </div>
 </body>
</html>
```

CSS　style03.css

```
.small {
    width: 50px;
}
.medium {
    width: 100px;
}
.large {
    width: 150px;
}
```

JavaScript　main03.js

```
$(function() {
   $("input[name='size']").on("change", changeSize);
});
function changeSize() {
   $("#photo > img").removeClass();
   $("#photo > img").addClass($("input[name='size']:
    checked").val());
}
```

解説

オプションボタンの選択状態が変更されると、changeSize関数が実行されます。まず、5行目で全画像から現在設定されているCSSクラスを削除し、初期化しています。

次に6行目で3つあるオプションボタンのうち現在選択中のオプションボタンのvalue属性値を取得し、全画像に対し同名のCSSクラスを適用しています。

▼例 ラジオボタンで「大」を選択している場合

❶「大」のvalue属性値である"large"が取得される

```
$("#photo > img").addClass($("input[name='size']:checked").val());
```

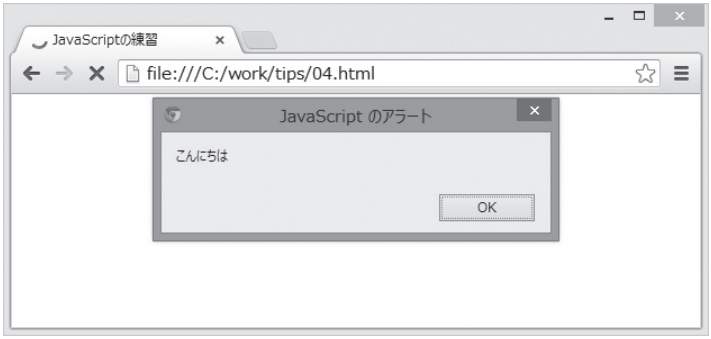
"large"

❷画像に対してCSSクラス"large"が適用される

```
$("#photo > img").addClass("large");
```

JavaScriptはブラウザの設定で無効にすることが可能です。必ずJavaScriptを動作させたいページでは、ユーザーにスクリプトの有効化を促すメッセージを表示すると良いでしょう。

■実行結果



JavaScriptが有効な場合

JavaScriptが無効な場合

ソースコード

HTML 04.html

```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title>JavaScriptの練習</title>
  <script>
   alert("こんにちは");
  </script>
 </head>
 <body>
  <noscript>JavaScriptを有効にして下さい。</noscript>
 </body>
</html>
```

解説

<noscript>タグは、スクリプトが動作しない場合に表示する内容を定義するタグです。スクリプトが有効になっている場合、<noscript>タグは無視されます。

Tips 05 テーブルの特定行を強調する

難易度★☆☆

テーブルの各行のうち、特定の文字列を含む行を強調表示します。例として、会員名簿の中でゴールド会員の行を強調します。

■実行結果

ソースコード

HTML 05.html

```
<!DOCTYPE html>
<html lang="ja">
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title>JavaScriptの練習</title>
  <script src="http://ajax.googleapis.com/ajax/libs/
jquery/1.9.1/jquery.min.js"></script>
  <script src="main05.js"></script>
  <link rel="stylesheet" type="text/css" href="style05.css">
 </head>
 <body>
  <h1>会員名簿</h1>
  <table id="userlist">
```

```
    <tr>
     <th>No</th>
     <th>名前</th>
     <th>ランク</th>
    </tr>
    <tr>
     <td>1</td>
     <td>田中</td>
     <td>ゴールド</td>
    </tr>
    <tr>
     <td>2</td>
     <td>鈴木</td>
     <td>シルバー</td>
    </tr>
    <tr>
     <td>3</td>
     <td>渡辺</td>
     <td>ゴールド</td>
    </tr>
    <tr>
     <td>4</td>
     <td>佐藤</td>
     <td>ゴールド</td>
    </tr>
    <tr>
     <td>5</td>
     <td>高橋</td>
     <td>プラチナ</td>
    </tr>
   </table>
  </body>
</html>
```

CSS style05.css

```
table, th, td {
    padding: 0px 10px;
    border-collapse: collapse;      /* セル間の間隔を開けない */
    border: solid 1px black;        /* 実線、1px、黒の線を引く */
}
.highlight {
    background-color: blue;
    color: white;
}
```

JavaScript　main05.js

```
01: $(function(){
02:   $("#userlist tr:contains('ゴールド')").addClass("highlight");
03: });
```

解説

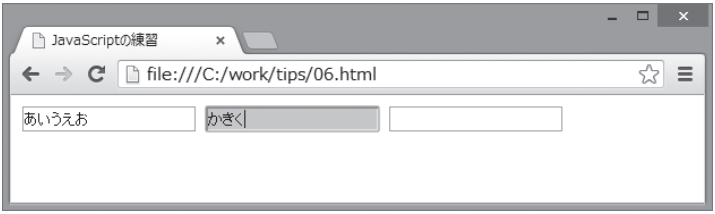
テーブルの各行のうち、"ゴールド"という文字列を含む行のみ抽出し、強調表示用のCSSクラス"highlight"を適用しています。

Tips 06 入力中のテキストボックスの背景色を変える

難易度★☆☆

現在カーソルが当たっている要素がどれであるかわかりやすく見せるために、入力中のテキストボックスの背景色を変えて強調表示します。

■実行結果



ソースコード

HTML 06.html

```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title>JavaScriptの練習</title>
  <script src="http://ajax.googleapis.com/ajax/libs/
jquery/1.9.1/jquery.min.js"></script>
  <script src="main06.js"></script>
 </head>
 <body>
  <input type="text" name="text1">
  <input type="text" name="text2">
  <input type="text" name="text3">
 </body>
</html>
```

JavaScript　main06.js

```
01: $(function() {
02:   $("input[type='text']").on("focus", function() {
03:     $(this).css("background-color", "pink");
04:   });
05:   $("input[type='text']").on("blur", function() {
06:     $(this).css("background-color", "white");
07:   });
08: });
```

解説

各テキストボックスにカーソルが当たったタイミングで実行されるfocusイベントで背景色にピンクを設定し、カーソルが外れたタイミングで実行されるblurイベントで背景色を白に戻しています。$(this)で取得した要素に対してスタイルを適用しているので、イベントの発生元であるテキストボックスにのみ背景色が設定されます。

Tips 07 ローディング画面を表示する

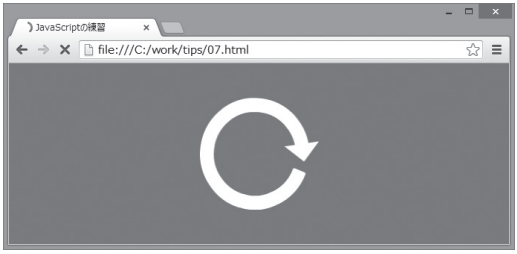
難易度★☆☆

Ajaxによるデータの取得など、ページの表示までに時間がかかる処理を実行する場合に、読込中であることを表わすローディング画面を表示します。第13章で紹介したリスト13.3（331ページ）のツイッター APIを利用するサンプルに、データの取得中はローディング画面を表示する処理を追加します。

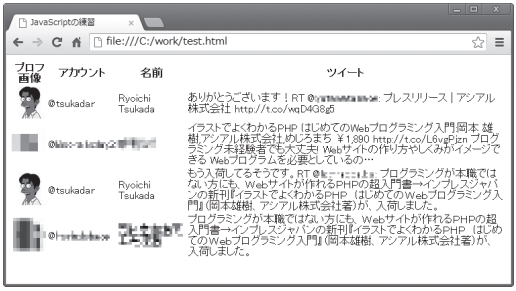
■使用する画像

ローディング画像（loading.png）

■実行結果



データ取得中



データ取得完了後

ソースコード

HTML　07.html

```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title>JavaScriptの練習</title>
  <script src="http://ajax.googleapis.com/ajax/libs/
jquery/1.9.1/jquery.min.js"></script>
  <script src="main07.js"></script>
  <link rel="stylesheet" type="text/css" href="style07.css">
 </head>
 <body>
  <!-- ローディング画面 -->
  <div id="wall"><img src="images/loading.png"
class="loading"></div>
  <table>
   <tr>
    <th>プロフ画像</th>
    <th>アカウント</th>
    <th>名前</th>
    <th>ツイート</th>
   </tr>
  </table>
 </body>
</html>
```

CSS　style07.css

```
* {
    margin: 0;
    padding: 0;
}
/* ローディング画面 */
#wall {
    /* スクロールしてもウィンドウ全体に固定表示する設定 */
    position: fixed;
    width: 100%;
    height: 100%;
    background-color: gray;
}
/* ローディング画像 */
.loading {
    /* 画面中央に表示する設定 */
    position: absolute;
    width: 160px;
```

```
    height: 150px;
    top: 50%;
    left: 50%;
    margin-top: -75px;
    margin-left: -80px;
}
```

JavaScript main07.js

```
01: $(function() {
02:   $("#wall").show();
03:   var q = encodeURIComponent("アシアル株式会社");
04:   $.getJSON("http://search.twitter.com/search.
    json?callback=?&q=" + q,
05:     function(data) {
06:       for(var i=0; i<data.results.length; i++) {
07:         $("table").append(
08:           "<tr>" +
09:           "<td><img src='" + data.results[i].
    profile_image_url + "'></td>" +
10:           "<td>@" + data.results[i].from_user + "</td>" +
11:           "<td>" + data.results[i].from_user_name + "</td>" +
12:           "<td>" + data.results[i].text + "</td>" +
13:           "</tr>"
14:         );
15:       }  // for文の閉じ括弧
16:       $("#wall").hide();
17:     }  // コールバック関数の閉じ括弧
18:   );
19:
20: });
```

解説

Ajax通信の実行前にローディング画面を表示し、データの取得が完了したらローディング画面を非表示にする処理を行なっています。取得完了後のタイミングで非表示処理を行なうには、getJSON関数の引数であるコールバック関数の中に処理を記述します。

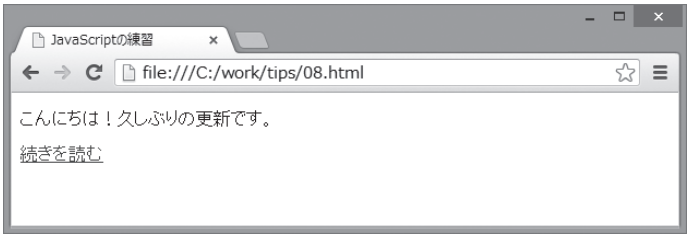
ローディング画像が回転したりするアニメーションを付けたい場合は、画像をGIFアニメーション（複数のGIFファイルをつなぎ合わせた形式）にしておきます。

Tips 08 長い文章を省略し、「続きを読む」リンクを表示する

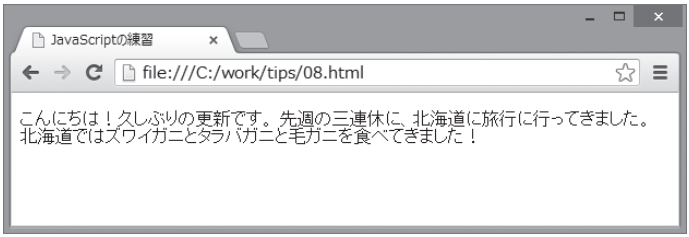
難易度★☆☆

最初は記事の一部のみを表示しておき、リンクをクリックすることで全文が表示される、ブログなどでよく見られる機能を作成します。

■実行結果



初期表示



「続きを読む」をクリック後

ソースコード

HTML 08.html

```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title>JavaScriptの練習</title>
  <script src="http://ajax.googleapis.com/ajax/libs/
jquery/1.9.1/jquery.min.js"></script>
```

```
  <script src="main08.js"></script>
 </head>
 <body>
  <p id="content">
   こんにちは！久しぶりの更新です。
   先週の三連休に、北海道に旅行に行ってきました。
   北海道ではズワイガニとタラバガニと毛ガニを食べてきました！
  </p>
  <!-- 省略記事の表示エリア -->
  <p id="omitText"></p>
  <a href="javascript:void(0)" rel="all">続きを読む</a>
 </body>
</html>
```

JavaScript main08.js

```
01: $(function() {
02:    // 記事全文を隠す
03:    $("#content").hide();
04:    // 先頭から20文字分抽出
05:    var omit = $("#content").text().slice(0, 20);
06:    // 省略記事表示エリアに抽出した文字列をセット
07:    $("#omitText").text(omit);
08:    $("a[rel='all']").on("click", showAll);
09: });
10: function showAll() {
11:    // リンクを隠す
12:    $(this).hide();
13:    // 省略記事を隠す
14:    $("#omitText").hide();
15:    // 記事全文を表示
16:    $("#content").show();
17: }
```

解説

HTML文書にはあらかじめ記事全文を記述しておき、初期表示時に全文の非表示化と、省略記事の抽出／表示処理を行ないます。5行目で記事の先頭から20文字分の文字列を抽出しています。textメソッドはhtmlメソッドと同様に要素の内容を取得しますが、内容に含まれるタグは除外されます。

「続きを読む」リンククリック時の処理は、showAll関数で行なっています。リンクと省略記事を隠し、記事全文を表示して、初期表示時と逆の表示状態にしています。

なお、「続きを読む」リンクのHTMLは、以下のように定義されています。

```
<a href="javascript:void(0)" rel="all">続きを読む</a>
```

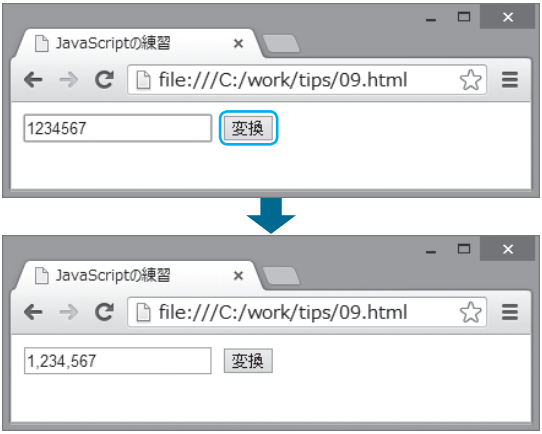
<a>タグのhref属性に"javascript:void(0)"を指定すると、<a>タグのリンク機能が無効化されます。リンククリック時にリンク先ページの表示ではなくJavaScriptによる処理を実行したい場合は、href属性にこの記述を入れておく必要があります。

Tips 09 金額をカンマ区切りで表示する

難易度★★☆

数字のみで入力された金額を、3桁ごとにカンマで区切って表示します。カンマ表記への変換は正規表現を使って行ないます。

■実行結果



ソースコード

HTML 09.html

```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title>JavaScriptの練習</title>
  <script src="http://ajax.googleapis.com/ajax/libs/
jquery/1.9.1/jquery.min.js"></script>
  <script src="main09.js"></script>
 </head>
 <body>
  <input type="text" id="price">
  <input type="button" id="btn" value="変換">
 </body>
</html>
```

JavaScript　main09.js

```
01: $(function(){
02:   $("#btn").on("click", convert);
03: });
04: function convert(){
05:   var price = $("#price").val();
06:   while (price.match(/([0-9]+)([0-9]{3})/)) {
07:     price = price.replace(/([0-9]+)([0-9]{3})/, "$1,$2");
08:   }
09:   $("#price").val(price);
10: }
```

解説

［変換］ボタンがクリックされると、convert関数が実行されます。5行目で入力された金額を取得し、変数priceに代入しています。テキストボックスの入力値は、必ず文字列型として取得されます。

6行目から8行目が、カンマ表記への変換を行なっている箇所です。6行目のwhile文の継続条件式は、priceの値が正規表現にマッチしている間、繰り返しを続けるという意味です。この正規表現のパターンは、「“1文字以上の数字列”と“3文字の数字列”が隣接している」ということを表わしています。

```
price.match(/[0-9]+[0-9]{3}/)
```

1文字以上の数字列　3文字の数字列

続いて7行目では、金額にカンマを挿入しています。replaceメソッドでは、正規表現の「後方参照」という機能を利用できます。後方参照とは、パターン文字列のうち、丸かっこ()で囲んだ部分を置換文字列として利用できる機能です。

```
price.replace(/([0-9]+)([0-9]{3})/, "$1,$2")
```

「1234567」という数字列が入力された場合を例とすると、最初の繰り返しでは「1234と567」がパターンにマッチし、「1234,567」に置き換えられます。

```
price.replace(/([0-9]+)([0-9]{3})/, "$1,$2")
```

1234,567
全体がマッチ

1234
567
1234,567

2回目の繰り返しでは、priceの値は「1234,567」になっているので、パターンにマッチするのは「1と234」で、マッチした箇所が「1,234」に置き換えられます。

```
price.replace(/([0-9]+)([0-9]{3})/, "$1,$2")
```

1234,567
「1234」がマッチ

1
234
1,234

これを繰り返していくことで最終的にパターンにマッチする箇所がなくなり、3桁おきにカンマが挿入された文字列が完成する仕組みとなっています。

Tips 10 開閉式のメニューを作る

難易度★★☆

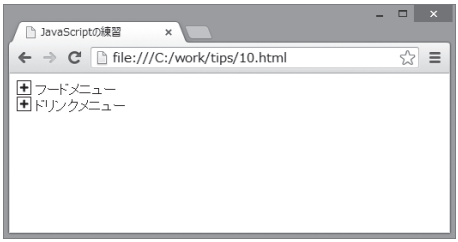
開閉ボタンをクリックするたびに、メニューを展開／折り畳みします。

■使用する画像

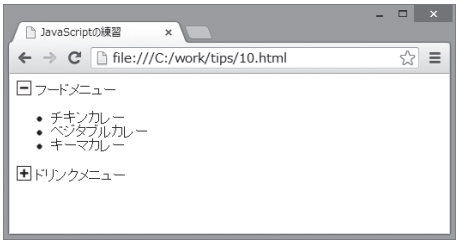
開くボタン（open.png）

閉じるボタン（close.png）

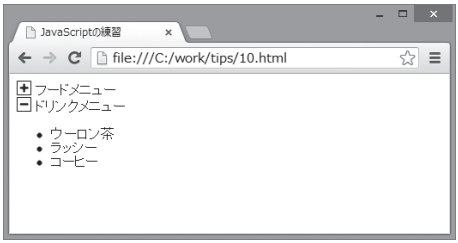
■実行結果



初期表示



フードメニューを開く



ドリンクメニューを開く

ソースコード

HTML 10.html

```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title>JavaScriptの練習</title>
  <script src="http://ajax.googleapis.com/ajax/libs/
jquery/1.9.1/jquery.min.js"></script>
  <script src="main10.js"></script>
 </head>
 <body>
  <div class="menu">
   <img src="images/open.png" name="open">
   <img src="images/close.png" name="close">
   フードメニュー
   <ul>
    <li>チキンカレー</li>
    <li>ベジタブルカレー</li>
    <li>キーマカレー</li>
   </ul>
  </div>
  <div class="menu">
   <img src="images/open.png" name="open">
   <img src="images/close.png" name="close">
   ドリンクメニュー
   <ul>
    <li>ウーロン茶</li>
    <li>ラッシー</li>
    <li>コーヒー</li>
   </ul>
  </div>
 </body>
</html>
```

JavaScript main10.js

```
01: $(function() {
02:   // メニューをすべて隠す
03:   $(".menu ul").hide();
04:   // 閉じるボタンをすべて隠す
05:   $(".menu img[name='close']").hide();
06:   $(".menu img").on("click", visibleChange);
07: });
08: function visibleChange() {
09:   // クリックされたメニューを展開／折り畳みする
10:   $(this).parent().children().toggle();
11: }
```

解説

まず、初期表示時にメニューと閉じるボタンを隠します。フードとドリンクの2つのメニューがあるので、両方の要素をセレクタで取得し、一括で非表示化を行なう処理を2～5行目で行なっています。

ボタンがクリックされたときは、visibleChange関数が実行されます。10行目の処理は、メニュー一覧および開閉ボタンの表示状態を切り替える処理です。parent()は、対象要素の親要素を取得します。また、children()は、要素が持つ子要素をすべて取得します。

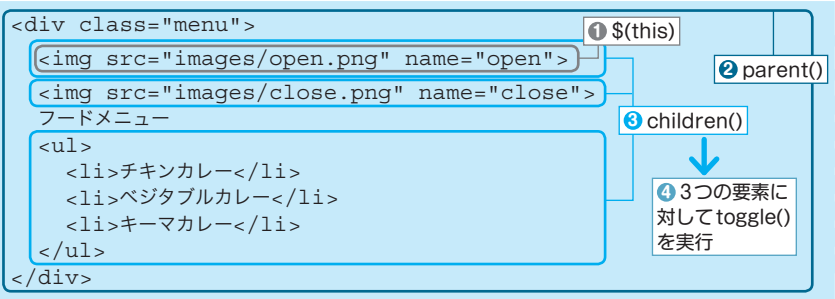
```
10:   $(this).parent().children().toggle();
```

イベントの発生元（開閉ボタン）	開閉ボタンの親要素（ ）	の子要素（<img>×2、 ）	表示／非表示の切り替え

▼初期表示時の状態（濃い網掛けの要素は非表示）

```
<div class="menu">
  <img src="images/open.png" name="open">
  <img src="images/close.png" name="close">
  フードメニュー
  <ul>
    <li>チキンカレー</li>
    <li>ベジタブルカレー</li>
    <li>キーマカレー</li>
  </ul>
</div>
<div class="menu">
  <img src="images/open.png" name="open">
  <img src="images/close.png" name="close">
  ドリンクメニュー
  <ul>
    <li>ウーロン茶</li>
    <li>ラッシー</li>
    <li>コーヒー</li>
  </ul>
</div>
```

▼「フードメニュー」の開くボタンがクリックされた場合の実行順序



▼「フードメニュー」の開くボタンクリック後の状態（濃い網掛けの要素は非表示）

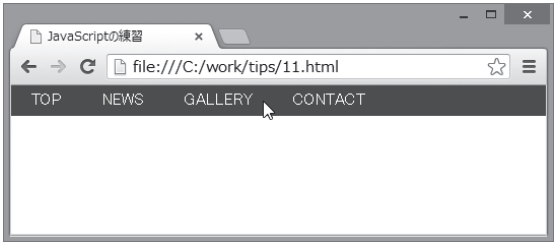
```
<div class="menu">
  <img src="images/open.png" name="open">
  <img src="images/close.png" name="close">
  フードメニュー
  <ul>
    <li>チキンカレー</li>
    <li>ベジタブルカレー</li>
    <li>キーマカレー</li>
  </ul>
</div>
<div class="menu">
  <img src="images/open.png" name="open">
  <img src="images/close.png" name="close">
  ドリンクメニュー
  <ul>
    <li>ウーロン茶</li>
    <li>ラッシー</li>
    <li>コーヒー</li>
  </ul>
</div>
```

Tips 11 自動的に隠れるナビゲーションメニューを作る

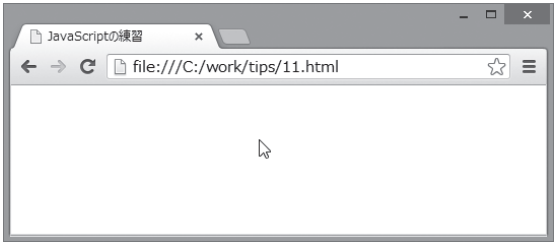
難易度★★☆

マウスポインタの位置に応じて、ウィンドウ上部に表示されるナビゲーションメニューの表示状態を切り替えます。

■実行結果



ウィンドウ上部から30px未満にマウスポインタがある場合



ウィンドウ上部から30px以降にマウスポインタがある場合

ソースコード

HTML　11.html

```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title>JavaScriptの練習</title>
  <script src="http://ajax.googleapis.com/ajax/libs/
jquery/1.9.1/jquery.min.js"></script>
```

```
  <script src="main11.js"></script>
  <link rel="stylesheet" type="text/css" href="style11.css">
 </head>
 <body>
  <div id="nav">
   <ul>
    <li>TOP</li>
    <li>NEWS</li>
    <li>GALLERY</li>
    <li>CONTACT</li>
   </ul>
  </div>
 </body>
</html>
```

CSS style11.css

```
* {
    margin: 0;
    padding: 0;
}
#nav {
    position: fixed;     /* ウィンドウ上部に固定表示する */
    width: 100%;
    height: 30px;
    background-color: blue;
    color: white;
}
#nav ul{
    list-style-type: none;     /* リストの行頭記号を表示しない */
}
#nav li {
    float: left;               /* リストの要素を横並びにする */
    padding: 5px 20px;
}
```

JavaScript main11.js

```
01: $(function() {
02:   $("#nav").hide();
03:   $(window).on("mousemove", showNaviArea);
04: });
05: function showNaviArea(e) {
06:   if($("#nav").is(':hidden') && e.clientY < 30) {
07:     $("#nav").slideDown(500);
08:   } else if($("#nav").is(':visible') && e.clientY >= 30){
09:     $("#nav").slideUp(500);
10:   }
11: }
```

解説

マウスが動いたときに処理を行なうには、mousemoveイベントを利用します。3行目で、ウィンドウ上でマウスポインタが動かされたときのイベントを設定しています。

```
$(window).on("mousemove", showNaviArea);
```

ウィンドウ全体を表わす

イベントが発生すると、showNaviArea関数が実行されます。mousemoveイベント発生時の処理は、引数としてマウスポインタの座標情報を受け取ります。横方向の座標はclientXプロパティ、縦方向の座標はclientYプロパティに入っています。

6～7行目で、ナビゲーションメニューをアニメーション付きで表示する処理を行なっています。表示条件は、ナビゲーションメニューが非表示の状態でかつ、縦方向の座標が30px未満の場合です。isメソッドは、対象の要素がセレクタ書式で記述した条件に一致する場合にtrueを返します。

```
6:    if($("#nav").is(':hidden') && e.clientY < 30) {
```

ナビゲーションメニューが非表示状態かどうか

マウスポインタの縦位置が上から30px未満かどうか

8～9行目は、ナビゲーションメニューが表示状態でかつ、縦方向の座標が30px以上の場合にナビゲーションメニューを隠す処理です。

```
8:    } else if($("#nav").is(':visible') && e.clientY >= 30){
```

ナビゲーションメニューが表示状態かどうか

マウスポインタの縦位置が上から30px以降かどうか

Tips 12 ページトップへ戻るボタンを表示する

難易度★★☆

ウィンドウを下にスクロールしたときに、ページの先頭へ戻るための画像ボタンを表示します。スクロールしていない状態ではボタンは表示しません。

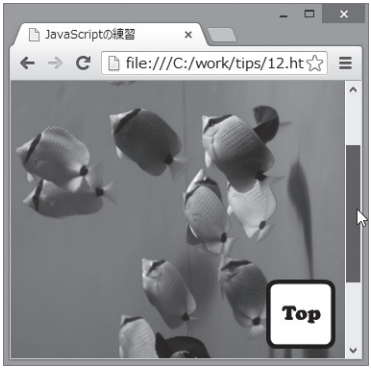
■使用する画像

ページの背景画像（fish.jpg）

戻るボタン（top.png）

■実行結果

ページトップ



スクロールした状態

ソースコード

HTML　12.html

```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title>JavaScriptの練習</title>
  <script src="http://ajax.googleapis.com/ajax/libs/
jquery/1.9.1/jquery.min.js"></script>
  <script src="main12.js"></script>
  <link rel="stylesheet" type="text/css" href="style12.css">
 </head>
 <body>
  <img src="images/top.png" id="topbtn">
 </body>
</html>
```

CSS　style12.css

```
body {
    height: 500px;
    background-image: url("images/fish.jpg"); /* 背景画像の設定 */
}
#topbtn {
    /* スクロールしても画像をウィンドウ右下に固定表示する設定 */
    position: fixed;
    right: 10px;
    bottom: 10px;
}
```

JavaScript　main12.js

```
$(function(){
   $("#topbtn").hide();
   $("#topbtn").on("click", gotoTop);
   $(window).on("scroll", showTopbtn);
});
function showTopbtn() {
   if($(window).scrollTop() > 0) {
     $("#topbtn").show();
   } else {
     $("#topbtn").hide();
   }
}
function gotoTop() {
   $("html,body").animate({ "scrollTop" : "0" }, 300);
   $("#topbtn").hide();
}
```

解説

まず、初期表示時に実行される処理から見てみましょう。

```
01: $(function(){
02:   $("#topbtn").hide();
03:   $("#topbtn").on("click", gotoTop);
04:   $(window).on("scroll", showTopbtn);
05: });
```

初期状態では戻るボタンは表示しないので、2行目で非表示処理を行なっています。3行目と4行目では、戻るボタンクリック時のイベントとウィンドウスクロール時のイベントを登録しています。スクロールイベントは、<window>要素に対して設定します。

続いて、ウィンドウスクロール時に実行されるshowTopbtn関数です。

```
06: function showTopbtn() {
07:   if($(window).scrollTop() > 0) {
08:     $("#topbtn").show();
09:   } else {
10:     $("#topbtn").hide();
11:   }
12: }
```

showTopbtn関数は、戻るボタンの表示状態を制御します。<window>要素のscrollTopメソッドは、現在のスクロール位置をピクセル値で表わします。ページの先頭を表示している場合、scrollTop()が返す値は0となります。この関数では、ウィンドウが下にスクロールされていれば戻るボタンを表示し、未スクロールの状態では戻るボタンを非表示にする処理を行なっています。

最後に、戻るボタンクリック時に実行されるgotoTop関数です。

```
13: function gotoTop() {
14:   $("html,body").animate({ "scrollTop" : "0" }, 300);
15:   $("#topbtn").hide();
16: }
```

gotoTop関数は、現在の位置からページの先頭までアニメーション付きで移動する処理を行ないます。animateメソッドでscrollTopの値を操作する場合は、<html>要素と<body>要素に対して処理を実行します。<window>要素に対してアニメーションを実行してもうまく動作しないので気をつけましょう。

Tips 13 スクロールで画像がめくれる動きを表現する

難易度★★☆

あらかじめ画像を2枚重ねて配置しておき、ウィンドウを下にスクロールしたときに、前面の画像がめくれて背面の画像が見えるような動作を実現します。

■使用する画像

前面の画像（frog1.jpg）

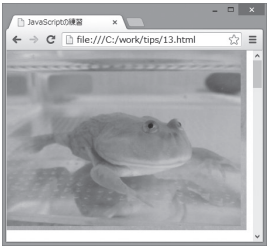
背面の画像（frog2.jpg）

高さは300px

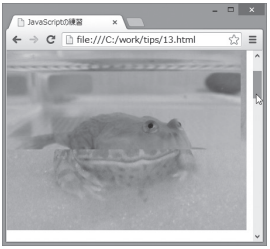
■画像の配置イメージ

X軸、Y軸ともに
同じ位置に重ねて配置される

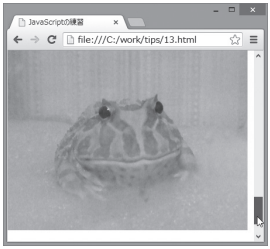
■実行結果



ページトップ



途中までスクロールした状態



下までスクロールした状態

ソースコード

HTML　13.html

```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title>JavaScriptの練習</title>
  <script src="http://ajax.googleapis.com/ajax/libs/
jquery/1.9.1/jquery.min.js"></script>
  <script src="main13.js"></script>
  <link rel="stylesheet" type="text/css" href="style13.css">
 </head>
 <body>
  <div id="over"><img src="images/frog1.jpg"></div>
  <div id="under"><img src="images/frog2.jpg"></div>
 </body>
</html>
```

CSS　style13.css

```
body {
    height: 2000px;
}
/* 画像の親要素 */
div {
    /* スクロールしても画像をウィンドウ左上に固定表示する設定 */
    position: fixed;
    left: 0px;
    top: 0px;
    height: 300px;      /* 画像と同じ高さを設定 */
}
/* frog2.jpgを囲む<div>要素 */
#under {
    z-index: 1;         /* 背面に表示 */
}
/* frog1.jpgを囲む<div>要素 */
#over {
    z-index: 2;         /* 前面に表示 */
    overflow: hidden;
}
```

JavaScript main13.js

```
01: $(function(){
02:   $(window).on("scroll", peelOff);
03: });
04: function peelOff() {
05:   var scroll = $(window).scrollTop();
06:   var height = $("#over img").height();
07:   $("#over").css("height", height-scroll);
08: }
```

解説

2つの画像はそれぞれ タグで囲まれていて、前面の画像を囲む 要素には"over"、背面の画像を囲む 要素には"under"というIDが付けられています。

要素は左上に固定表示するようにCSSで設定しているため、2つの画像はどちらも同じ座標上に配置されていることになります。

```
div {
    /* スクロールしても画像をウィンドウ左上に固定表示する設定 */
    position: fixed;
    left: 0px;
    top: 0px;
    height: 300px;      /* 画像と同じ高さを設定 */
}
```

各 要素には、それぞれz-indexを指定しています。z-indexは、重なり順を設定するプロパティです。数値が大きいほど前面に、数値が小さいほど背面に表示されます。

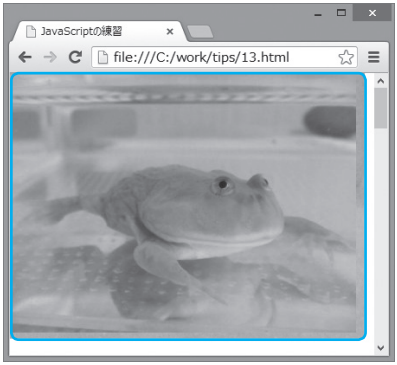
また、前面の画像を囲む 要素にはoverflow: hidden;が設定されているため、 要素の枠からはみ出た部分は非表示となります。

```
#under {
    z-index: 1;       /* 背面に表示 */
}
#over {
```

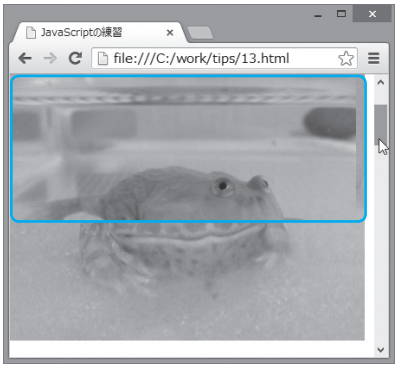
```
    z-index: 2;        /* 前面に表示 */
    overflow: hidden;
}
```

スクリプトの内容は単純です。ウィンドウがスクロールされると、peelOff関数が実行されます。peelOff関数では、前面の画像を囲む 要素の高さを変化させる処理を行なっています。 要素の高さは、「画像の高さ － スクロール量」で求めることができます。なお、6行目のheightメソッドは、要素の高さを数値型で取得するメソッドです。

```
05:    var scroll = $(window).scrollTop();
06:    var height = $("#over img").height();
07:    $("#over").css("height", height-scroll);
```



要素の高さは
画像と同じ300px



140px分スクロールした場合、
300px－140px=160pxを
要素の高さとして設定する

前面の 要素の高さが減った分、
背面の画像が見えるようになる

Tips 14 ドアが開くようなアニメーションを表現する

難易度★★☆

ある画像の前面に2枚の扉画像を重ねて配置しておき、どちらかの扉画像をクリックするとドアが左右に開いていくように見える動作を実現します。

■使用する画像

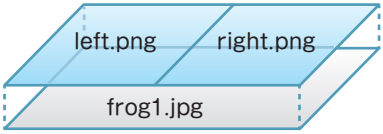
幅は400px

背面の画像（frog1.jpg）

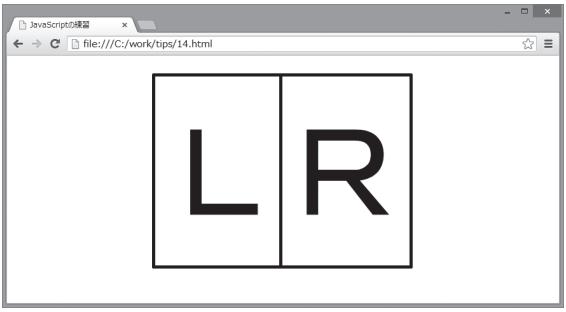
幅は200px

前面の画像（left.pngとright.png）

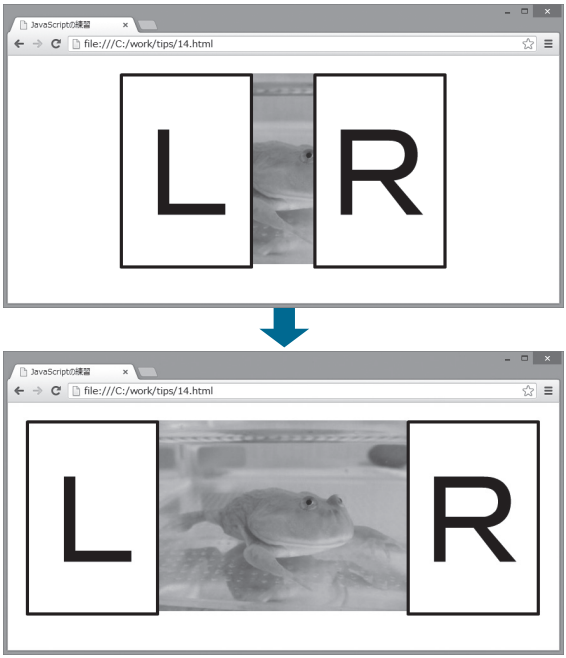
■画像の配置イメージ



■実行結果



初期表示



扉画像をクリックするとドアが開くような動作をする

ソースコード

HTML　14.html

```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title>JavaScriptの練習</title>
  <script src="http://ajax.googleapis.com/ajax/libs/
jquery/1.9.1/jquery.min.js"></script>
  <script src="main14.js"></script>
  <link rel="stylesheet" type="text/css" href="style14.css">
 </head>
 <body>
  <div class="door">
   <img src="images/frog1.jpg" id="photo">
   <img src="images/left.png" id="left">
   <img src="images/right.png" id="right">
  </div>
 </body>
</html>
```

CSS style14.css

```
body {
    padding: 20px;
}
/* 3つの画像の親要素 */
.door {
    position: relative;
    width: 400px;
    left: 200px;
}
.door img {
    /* 各画像が親の<div>要素を基準とした相対配置となる */
    position: absolute;
    top: 0px;
}
#photo {
    z-index: 1;        /* 背面に表示 */
    left: 0px;
}
#left {
    z-index: 2;        /* 前面に表示 */
    left: 0px;         /* 左に配置 */
}
#right {
    z-index: 2;        /* 前面に表示 */
    left: 200px;       /* 右に配置 */
}
```

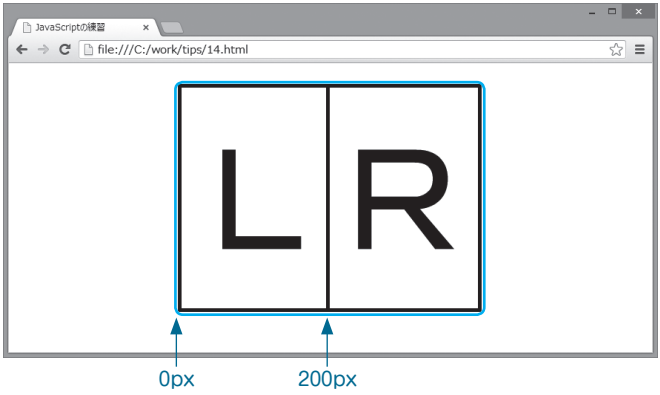
JavaScript main14.js

```
01: $(function(){
02:   $("#left,#right").on("click", openDoor);
03: });
04: function openDoor() {
05:   $("#left").animate({"left" : "-200px"}, 400);
06:   $("#right").animate({"left" : "400px"}, 400);
07: }
```

解説

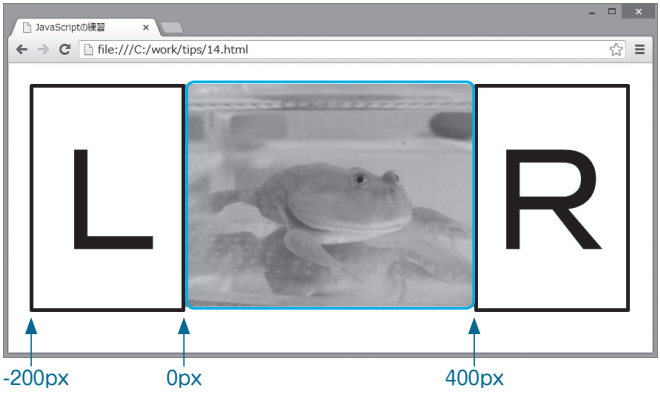
写真の前面に、左右の扉画像が2枚配置されています。扉画像はそれぞれ、親の 要素の左上を基準とした相対位置に配置されています。

▼色付き枠部分が親の 要素



扉画像のどちらかがクリックされると、左の扉画像は親の 要素の左端から-200pxの位置へ、右の扉画像は親の 要素の左端から400pxの位置へ移動します。

▼色付き枠部分が親の 要素

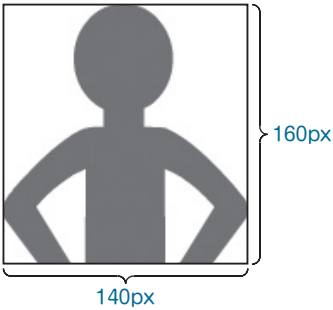


Tips 15 アバター機能を作る

難易度★★☆

アバター画像に対して、ユーザーが選択した服を着せる機能を実装します。アバター画像の前面に同じ大きさの着替え画像を重ねることで、服を着ているように見せることができます。

■使用する画像

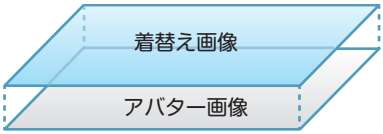


アバター画像（human.png）

着替え画像（plain.png ／ star.png ／ heart.png）

※大きさはアバター画像と同じ。服の周りの余白は透過色。
※アバター画像、着替え画像ともに実際の画像には枠は付いていません。

■画像の配置イメージ



X軸、Y軸ともに
同じ位置に重ねて配置される

■実行結果

初期表示

着替え画像をクリック

ソースコード

HTML　15.html

```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title>JavaScriptの練習</title>
  <script src="http://ajax.googleapis.com/ajax/libs/
jquery/1.9.1/jquery.min.js"></script>
  <script src="main15.js"></script>
  <link rel="stylesheet" type="text/css" href="style15.css">
 </head>
 <body>
  <!-- アバター表示エリア -->
  <div id="avatar">
   <img src="images/human.png" id="human">
  </div>
  <!-- 着替え画像一覧エリア -->
  <div id="clothes">
   <img src="images/plain.png">
   <img src="images/star.png">
   <img src="images/heart.png">
  </div>
 </body>
</html>
```

CSS style15.css

```
/* アバター表示エリア */
#avatar {
    position: relative;
    width: 140px;
    height: 160px;
    margin: 10px auto;
}
/* アバター表示エリア内の画像 */
#avatar img {
    position: absolute;
    top: 0px;
    left: 0px;
}
/* 着替え画像一覧エリア */
#clothes {
    white-space: nowrap;
}
/* 着替え画像 */
#clothes img {
    width: 100px;       /* 縮小表示 */
    border: solid 1px black;
}
```

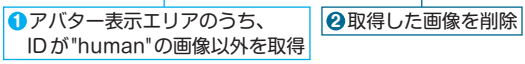
JavaScript main15.js

```
01: $(function() {
02:   $("#clothes img").on("click", changeClothes);
03: });
04: function changeClothes() {
05:   $("#avatar img:not(#human)").remove();
06:   $("#avatar").append($(this).clone());
07: }
```

解説

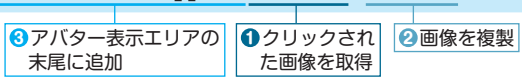
着せ替え画像一覧のうちいずれかをクリックすると、changeClothes関数が実行されます。まず、アバター画像を初期化する処理を行ないます。5行目では、アバター表示エリアから、アバター画像以外の画像をすべて削除しています。remove()は対象の要素を削除するメソッドです。これにより、すでにアバターに服が着せられている場合も裸の状態に戻ります。

```
05:    $("#avatar img:not(#human)").remove();
```



続いて、6行目で選択された画像を複製し、アバター表示エリアに追加しています。clone()は対象の要素のコピーを生成するメソッドです。

```
06:    $("#avatar").append($(this).clone());
```



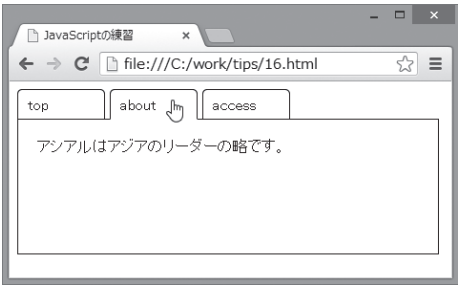
複数の要素が同じ座標に配置される場合、後ろに挿入された要素ほど前面に表示されます。よってこのサンプルではz-indexで重ね順を指定する必要はありません。

Tips 16 タブメニューを作る

難易度★★☆

タブメニューのクリックにより、対応するコンテンツを表示します。同時に現在アクティブなタブが最前面に表示されているような視覚効果を表現します。

■実行結果



ソースコード

HTML　16.html

```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title>JavaScriptの練習</title>
  <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js"></script>
  <script src="main16.js"></script>
  <link rel="stylesheet" type="text/css" href="style16.css">
 </head>
 <body>
  <div id="tabArea">
   <div id="top" class="tab active">
    <span>top</span>
   </div>
   <div id="about" class="tab">
    <span>about</span>
   </div>
   <div id="access" class="tab">
    <span>access</span>
   </div>
  </div>
  <!-- コンテンツ表示エリア -->
  <div id="content">
   <div id="content-top">アシアル株式会社のWebサイトです。</div>
   <div id="content-about" class="hidden">アシアルはアジアのリーダーの略です。</div>
   <div id="content-access" class="hidden">御茶ノ水駅から徒歩10分です。</div>
  </div>
 </body>
</html>
```

CSS　style16.css

```
#content {
    width: 400px;
    height: 100px;
    padding: 20px;
    margin-top: 1px;     /* 上部に1px分の余白を取る */
    border: solid 1px black;
    z-index: 1;          /* 背面に表示 */
}
#tabArea {
```

```
    height: 30px;
    white-space: nowrap;
    z-index: 2;          /* 前面に表示 */
}
.tab {
    width: 80px;
    height: 30px;
    padding-left: 10px;
    display: inline-block;          /* タブを横並びに表示する */
    border: solid 1px black;
    border-radius: 5px 5px 0 0;     /* タブの角を丸くする */
    background-color: white;
    cursor: pointer;                /* 手の形のカーソルにする */
}
.active {
    border-bottom: solid 1px white;  /* タブの下線を白で表示 */
}
.tab span {
    line-height: 30px;     /* 1行分の高さをタブの高さと同じにする */
}
.hidden {
    display: none;
}
```

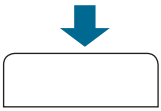
JavaScript main16.js

```
01: $(function() {
02:   $("#tabArea .tab").on("click", tabChange);
03: });
04: function tabChange() {
05:   // タブと本文の初期化
06:   $("#tabArea .tab").removeClass("active");
07:   $("#content div").hide();
08:
09:   // タブと本文の有効化
10:   $(this).addClass("active");
11:   var id = $(this).attr("id");
12:   $("#content-" + id).show();
13: }
```

解説

タブは、 要素のまわりにCSSで線を引いて作成します。

```
.tab {
    width: 80px;
    height: 30px;
    padding-left: 10px;
    display: inline-block;          /* タブを横並びに表示する */
    border: solid 1px black;
    border-radius: 5px 5px 0 0;     /* タブの角を丸くする */
    background-color: white;
    cursor: pointer;                /* 手の形のカーソルにする */
}
```



要素に"tab"クラスを適用した状態

現在選択中のアクティブなタブは、下線のみ白で描画します。

```
.active {
    border-bottom: solid 1px white;   /* タブの下線を白で表示 */
}
```

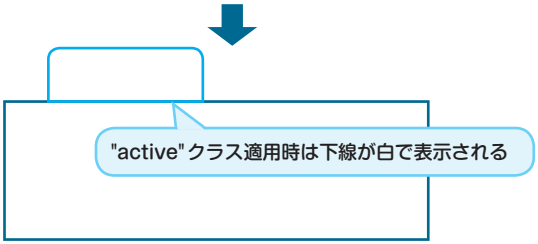
要素に"tab"クラスと"active"クラスを適用した状態

同様に、コンテンツ表示エリアも 要素のまわりに線を引いて作成します。線は要素の外側に描画されるため、 要素が隣接している場合は線が重なり合って表示されます。

この微妙なズレを調整するために、コンテンツ表示エリアの上に余白をとって、1px分下にずらします。

さらに、z-indexプロパティでタブをコンテンツ表示エリアの前面に重なるように指定します。タブに"active"クラスが指定されている場合は下線が白で表示されるため、タブとコンテンツ表示エリアを区切る線が隠れ、1枚のつながった紙のように見えます。

```
#content {
    width: 400px;
    height: 100px;
    padding: 20px;
    margin-top: 1px;     /* 上部に1px分の余白を取る */
    border: solid 1px black;
    z-index: 1;          /* 背面に表示 */
}
#tabArea {
    height: 30px;
    white-space: nowrap;
    z-index: 2;          /* 前面に表示 */
}
```



各タブがクリックされると、tabChange関数が実行されます。まず、6行目と7行目では全タブから"active"クラスを削除し、各コンテンツを非表示にする初期化処理を行なっています。

10行目では、イベントの発生元であるタブに対して"active"クラスを適用しています。

11行目、12行目ではイベント発生元のタブが持つID値を取得し、タブに対応するコンテンツを表示しています。

▼「about」タブがクリックされた場合

```
<div id="tabArea">
  <div id="top" class="tab active">
    <span>top</span>
  </div>
  <div id="about" class="tab">
    <span>about</span>
  </div>
  <div id="access" class="tab">
    <span>access</span>
  </div>
</div>
<!-- コンテンツ表示エリア -->
<div id="content">
  <div id="content-top">アシアル株式会社のWebサイトです。</div>
  <div id="content-about" class="hidden">アシアルはアジアの
リーダーの略です。</div>
  <div id="content-access" class="hidden">御茶ノ水駅から徒歩
10分です。</div>
</div>
```

イベントの発生元

"content-" + タブのID が対応するコンテンツ

Tips 17 サムネイル画像付きのスライドショーを作る

難易度★★★

第14章のリスト14.8（355ページ）で作成したスライドショーを改造して、サムネイル一覧の中から選択した画像まで一気に移動する機能を作成します。

■実行結果

ソースコード

HTML 17.html

```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title>JavaScriptの練習</title>
  <script src="http://ajax.googleapis.com/ajax/libs/
jquery/1.9.1/jquery.min.js"></script>
  <script src="main17.js"></script>
  <link rel="stylesheet" type="text/css" href="style17.css">
 </head>
```

```
 <body>
  <!-- スライドショー表示エリア -->
  <div id="photoFrame">
   <div id="imageArea"></div>
  </div>
  <!-- サムネイル画像表示エリア -->
  <div id="naviArea"></div>
 </body>
</html>
```

CSS　style17.css

```
* {
    margin: 0;
    padding: 0;
}
#photoFrame {
    position: relative;
    width: 200px;
    height: 150px;
    left: 50%;
    margin: 30px 0px 30px -100px;    /* センタリング表示する設定 */
    overflow: hidden;
}
/* スライドショー表示エリアの設定 */
#imageArea {
    position: absolute;
    left: 0px;
    width: 1000px;
    height: 150px;
    white-space: nowrap;
}
#imageArea img {
    width: 200px;
    height: 150px;
}
/* サムネイル画像表示エリアの設定 */
#naviArea {
    position: relative;
    width: 400px;
    left: 50%;
    margin-left: -200px;     /* センタリング表示する設定 */
    white-space: nowrap;
}
#naviArea img{
    width: 80px;
}
```

JavaScript main17.js

```
01: $(function() {
02:   var imageArea = $("#imageArea"); // スライドショー表示エリア
03:   var naviArea = $("#naviArea");   // サムネイル画像表示エリア
04:
05:   // スライドショー表示エリアとサムネイル画像表示エリアの両方に画像をロード
06:   for(var i=1; i<=5; i++) {
07:     imageArea.append("<img src='images/flower" + i + ".jpg'>");
08:     naviArea.append("<img src='images/flower" + i + ".jpg'>");
09:   }
10:
11:   // サムネイル画像クリック時の処理
12:   $("#naviArea img").on("click", moveImageArea);
13: });
14: function moveImageArea() {
15:   var imageArea = $("#imageArea");
16:   // 選択されたサムネイル画像のソースを取得
17:   var src = $(this).attr("src");
18:   // スライドショー表示エリア内で、選択したサムネイルと同じ画像の表示位置を取得
19:   var targetPosition = imageArea.find(
    "img[src='" + src + "']").position().left * -1;
20:   // 対象画像の表示位置までアニメーションで移動
21:   imageArea.animate({"left" : targetPosition }, 500);
22: }
```

解説

まず、6～9行目で、スライドショー表示エリアとサムネイル画像表示エリアの両方にまったく同じ画像を5件挿入しています。

サムネイル画像表示エリアのいずれかの画像がクリックされると、moveImageArea関数が実行されます。

17行目では、選択されたサムネイル画像のsrc属性を取得しています。19行目で、同じsrc属性が設定されている画像をスライドショー表示エリアの中から検索し、画像の左位置を取得しています。リスト14.8では左位置をcssメソッドで取得していましたが、今回その方法は使えません。cssメソッドは、CSSで明示的にプロパティを指定している場合にのみ有効です。スライドショー表示エリアにはleftプロパティが設定されていますが、その中の<img>要素1つ1つには設定していないので、cssメソッドでleftプロパティを取得すると返却される値は"auto"になります。このような場合、positionというメソッドを利用します。position()は、親要素に対する要素の相対位置を取得します。

プロパティとして「top」と「left」があり、位置情報を数値型で保持しています。

flower3.pngが選択された場合を例として解説します。

```
imageArea.find("img[src='" + src + "']").position().left * -1;
```

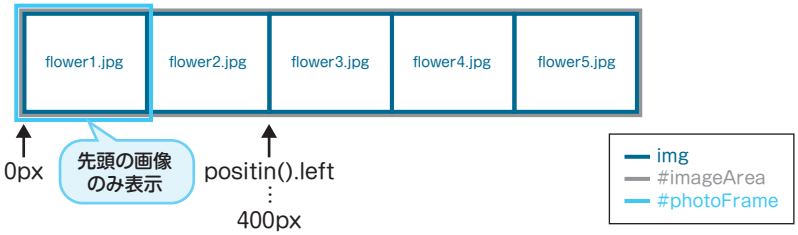
クリックされたサムネイル画像のsrc属性値

```
imageArea.find("img[src='images/flower3.png']").position().left * -1;
```

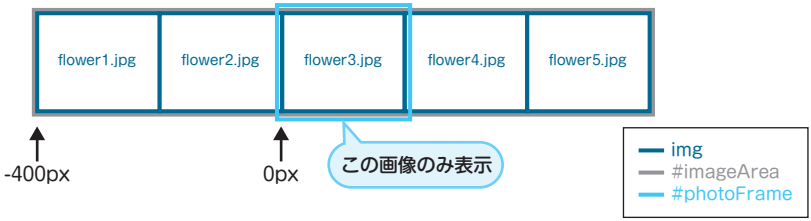
スライドショー表示エリアの中からflower3.pngを表示しているimg要素を取得

最後にposition().leftで取得した左位置に対して-1を掛けています。これはスライドショー表示エリアのleftプロパティとして設定すべき値を算出する処理です。

初期表示時、imageAreaのleftは0pxに設定されています。



flower3.pngを表示領域であるphotoFrame内に配置するには、flower3.pngのleftを0pxの位置に移動しなければなりません。そのためには、スライドショー表示エリアのleftを-400pxに設定する必要があるのです。



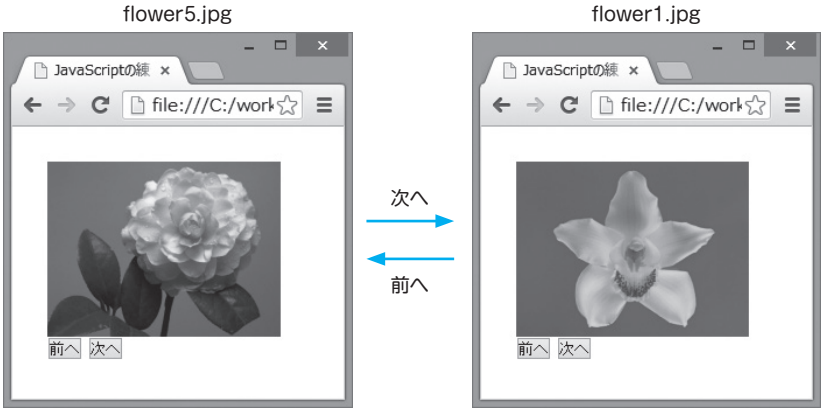
このようにして算出した位置までスライドショー表示エリアをanimateメソッドで移動させることで、指定画像まで一気にスライドさせる処理を実現しています。

Tips 18 ループするスライドショーを作る

難易度★★★

第14章のリスト14.8（355ページ）で作成したスライドショーを、画像がループして表示されるように改造します。

■実行結果



ソースコード

HTML 18.html

```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title>JavaScriptの練習</title>
  <script src="http://ajax.googleapis.com/ajax/libs/
jquery/1.9.1/jquery.min.js"></script>
  <script src="main18.js"></script>
  <link rel="stylesheet" type="text/css" href="style18.css">
 </head>
 <body>
  <div id="photoFrame">
```

```
    <div id="imageArea0" class="imageArea"></div>
    <div id="imageArea1" class="imageArea"></div>
   </div>
   <div>
    <button id="prev">前へ</button>
    <button id="next">次へ</button>
   </div>
  </body>
</html>
```

CSS　style18.css

```
* {
    margin: 0;
    padding: 0;
}
body {
    padding: 30px;
}
#photoFrame {
    position: relative;
    width: 200px;
    height: 150px;
    overflow: hidden;
}
.imageArea {
    position: absolute;
    width: 1000px;
    height: 150px;
    white-space: nowrap;
}
#imageArea0 {
    left: -1000px;
}
#imageArea1 {
    left: 0px;
}
img {
    width: 200px;
    height: 150px;
}
```

JavaScript main18.js

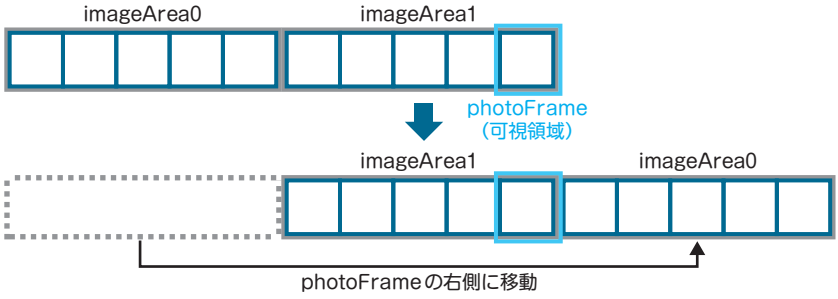
```
01: var currentIndex = 1;
02:
03: $(function() {
04:   var photoFrame = $("#photoFrame");
05:   for(var i=1; i<=5; i++) {
06:     photoFrame.find(".imageArea").append(
    "<img src='images/flower" + i + ".jpg'>");
07:   }
08:
09:   $("#next").on("click", function() {
10:     toDisabled();
11:     photoFrame.find(".imageArea").animate(
12:       {"left" : "-=200px"}, "fast", "linear", next);
13:   });
14:
15:   $("#prev").on("click", function() {
16:     toDisabled();
17:     photoFrame.find(".imageArea").animate(
18:       {"left" : "+=200px"}, "fast", "linear", prev);
19:   });
20: });
21:
22: function next() {
23:   // 隠れているほうのimageAreaのインデックスを算出
24:   var otherIndex = 1 - currentIndex;
25:   if(parseInt($("#imageArea" + currentIndex).
    css("left")) == -800) {
26:     // 末尾の画像を表示したときの処理
27:     // 隠れているほうのimageAreaのleftプロパティをphotoFrameの右側に設定
28:     $("#imageArea" + otherIndex).css("left", "200px");
29:   }
30:   else if(parseInt($("#imageArea" + currentIndex).
    css("left")) == -1000) {
31:     // imageAreaが切り替わったタイミングでカレントインデックスを変更
32:     currentIndex = otherIndex;
33:   }
34:     toEnabled();
35: }
36:
37: function prev() {
38:   // 隠れているほうのimageAreaのインデックスを算出
39:   var otherIndex = 1 - currentIndex;
40:   if(parseInt($("#imageArea" + currentIndex).
    css("left")) == 0) {
41:     // 先頭の画像を表示したときの処理
42:     // 隠れているほうのimageAreaのleftプロパティをphotoFrameの左側に設定
43:     $("#imageArea" + otherIndex).css("left", "-1000px");
44:   }
```

```
45:    else if(parseInt($("#imageArea" + currentIndex).
    css("left")) == 200) {
46:      // imageAreaが切り替わったタイミングでカレントインデックスを変更
47:      currentIndex = otherIndex;
48:    }
49:    toEnabled();
50: }
51:
52: function toDisabled() {
53:    $("#prev, #next").attr("disabled", "disabled");
54: }
55:
56: function toEnabled() {
57:    $("#prev, #next").removeAttr("disabled");
58: }
```

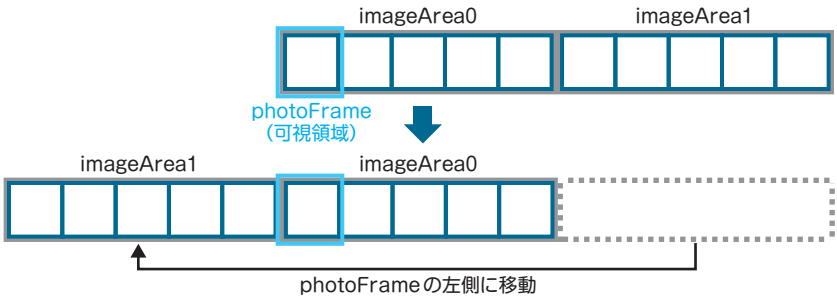
解説

スライドショーのループを実現するには、imageArea（5件分の画像を囲む 要素）を2つ用意します。末尾の画像、あるいは先頭の画像を表示したタイミングで、隠れているほうのimageAreaのleftプロパティを操作し、左右のどちらかに移動します。以下のような実行イメージになります。

▼［次へ］ボタンがクリックされて末尾の画像が表示されたとき



▼［前へ］ボタンがクリックされて先頭の画像が表示されたとき



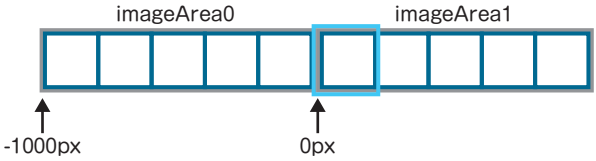
まず、あらかじめHTML文書にimageAreaを2つ用意しておきます。IDはそれぞれimageArea0とimageArea1としています。末尾の0と1は、それぞれを識別するためのインデックスとして利用します。また、両方に同じクラス名を設定しているので、2つまとめて操作することもできます。

```
<div id="photoFrame">
 <div id="imageArea0" class="imageArea"></div>
 <div id="imageArea1" class="imageArea"></div>
</div>
```

CSSでは、各imageAreaのleftプロパティに初期位置を設定しています。最初はimageArea0をphotoFrame（可視領域）の左側に隠し、imageArea1の1番目の画像が表示されるように配置します。

```
#imageArea0 {
    left: -1000px;
}
#imageArea1 {
    left: 0px;
}
```

▼各imageAreaの初期位置



以降、現在表示されているほうのimageAreaのことを「カレントimage Area」と表現します。カレントとは、「現在の」という意味です。

それでは、JavaScriptのコードを順に追っていきましょう。

まず、1行目ではグローバル変数currentIndexを宣言し、初期値としてカレントimageAreaのインデックスを代入しています。

```
01: var currentIndex = 1;
```

4～7行目でimageAreaの親要素であるphotoFrameを取得し、子要素のimageAreaに画像をロードしています。imageAreaは2つあるので、両方にまったく同じ5件分の<img>要素が挿入されます。

```
03: $(function() {
04:   var photoFrame = $("#photoFrame");
05:   for(var i=1; i<=5; i++) {
06:     photoFrame.find(".imageArea").append(
    "<img src='images/flower" + i + ".jpg'>");
07:   }
```

9～13行目は、［次へ］ボタンクリック時の処理です。アニメーション完了後にnext関数を実行します。

15～19行目は、［前へ］ボタンクリック時の処理で、こちらはアニメーション完了後にprev関数を実行します。

続いて、このサンプルの肝の部分であるimageAreaの移動処理を行なうnext関数を見ていきましょう。

```
22: function next() {
23:   // 隠れているほうのimageAreaのインデックスを算出
24:   var otherIndex = 1 - currentIndex;
```

まず、24行目で隠れているほうのimageAreaのインデックスを取得しています。インデックスは、以下の計算式によって算出することが可能です。

▼隠れているほうのimageAreaのインデックスを求める計算式

```
1 - カレントimageAreaのインデックス
```

Hint 計算式の考え方

- カレントのインデックスが1の場合、隠れているほうのインデックスは0
- カレントのインデックスが0の場合、隠れているほうのインデックスは1

↓

カレントのインデックスを反転させれば、もう一方のインデックスを得ることができる。

実際にimageAreaを移動する処理は、25～29行目のifブロック内で行なっています。

```
25:   if(parseInt($("#imageArea" + currentIndex).
    css("left")) == -800) {
26:     // 末尾の画像を表示したときの処理
27:     // 隠れているほうのimageAreaのleftプロパティをphotoFrameの右側に設定
28:     $("#imageArea" + otherIndex).css("left", "200px");
29:   }
```

if文の条件式では、カレントimageAreaの末尾の画像を表示しているかどうかを判定しています。次のように変数を使って記述することで、動的にセレクタを生成できます。

▼カレントimageAreaのインデックスが1の場合

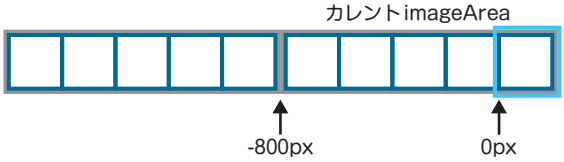
```
$("#imageArea" + currentIndex)
```

1

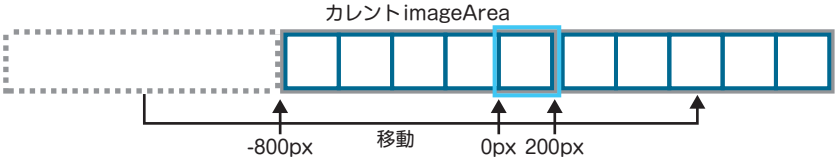
```
$("#imageArea1")
```

末尾の画像を表示している場合、カレントimageAreaのleftは-800pxです。このときに、隠れているほうのimageAreaを右側に移動します。photoFrame（可視領域）の右側に配置するためには、leftに200pxを設定します。

▼末尾の画像を表示しているときの配置



▼隠れているほうのimageAreaを移動した後の配置



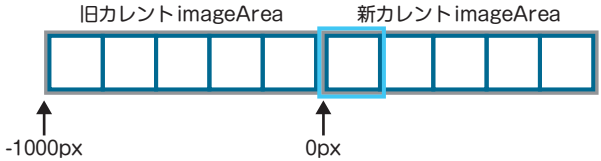
画像が末尾に達するたびにこの処理を繰り返すことにより、あたかも画像が延々とつながっているかのように見せることができるのです。

また、next関数ではcurrentIndexの値を変更する処理も行なっています。

```
30:    else if(parseInt($("#imageArea" + currentIndex).
    css("left")) == -1000) {
31:      // imageAreaが切り替わったタイミングでカレントインデックスを変更
32:      currentIndex = otherIndex;
33:    }
```

30～33行目のelse if文では、カレントimageAreaがphotoFrame（可視領域）から外れたタイミングで、今まで隠れていたほうをカレントに設定し直しています。

▼カレントimageAreaがphotoFrameから外れたときの配置



［前へ］ボタンをクリックしたときのprev関数もほぼ同じことを行なっています。相違点は、leftプロパティの設定値のみです。

```
37: function prev() {
38:   // 隠れているほうのimageAreaのインデックスを算出
39:   var otherIndex = 1 - currentIndex;
40:   if(parseInt($("#imageArea" + currentIndex).
    css("left")) == 0) {
41:     // 先頭の画像を表示したときの処理
42:     // 隠れているほうのimageAreaのleftプロパティをphotoFrameの左側に設定
43:     $("#imageArea" + otherIndex).css("left", "-1000px");
44:   }
45:   else if(parseInt($("#imageArea" + currentIndex).
    css("left")) == 200) {
46:     // imageAreaが切り替わったタイミングでカレントインデックスを変更
47:     currentIndex = otherIndex;
48:   }
49:   toEnabled();
50: }
```

Hint　より良いコードの記述方法

next関数とprev関数は、leftプロパティの設定値が異なるだけで、まったく同じ処理を行なっています。見た目にはわかりやすいのですが、実はこのような冗長なスクリプトはあまり好ましくありません。なんらかの変更を加えようとしたときに両方の関数を修正しなければならなくなるため、メンテナンス性が劣るのです。

改善例としては、まず関数を1つにまとめ、引数で「次へ」「前へ」のどちらの処理なのかを示す値を受け取ります。leftプロパティの設定値群は、あらかじめ変数や配列に保持しておき、引数に応じて適切なほうを選択します。画像の数を増やしても動作するように、画像のwidthプロパティを元にしてleftプロパティの設定値を動的に算出できれば、さらに良いスクリプトになるでしょう。

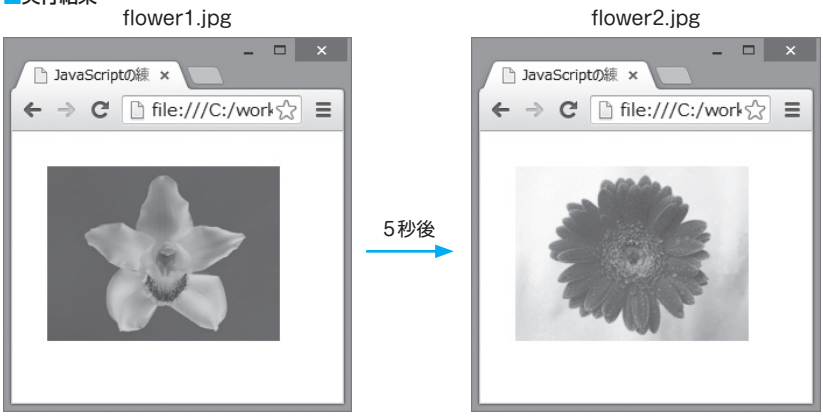
なお、animateメソッドのコールバック関数は関数名または関数式の形式でなければ記述できないため、引数を渡す場合は関数呼び出しを匿名関数で内包する必要があります。詳しくは第10章の「引数のある関数をイベントハンドラに設定する」（246ページ）を参考にしてください。

Tips 19 一定時間おきに自動で動くスライドショーを作る

難易度★★☆

Tips18のループするスライドショーのサンプルを、ボタンクリックではなく5秒おきに自動で画像が流れるように改造します。

■実行結果



ソースコード

HTML　19.html ⇒［前へ］［次へ］ボタンを削除

```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title>JavaScriptの練習</title>
  <script src="http://ajax.googleapis.com/ajax/libs/
jquery/1.9.1/jquery.min.js"></script>
  <script src="main19.js"></script>
  <link rel="stylesheet" type="text/css" href="style18.css">
 </head>
 <body>
  <div id="photoFrame">
```

```
    <div id="imageArea0" class="imageArea"></div>
    <div id="imageArea1" class="imageArea"></div>
   </div>
   <div>
    <button id="prev">前へ</button>
    <button id="next">次へ</button>
   </div>
  </body>
</html>
```

CSS style18.css ⇒変更なし

JavaScript main19.js ⇒網掛けの箇所が変更点

```
01: $(function() {
02:   var photoFrame = $("#photoFrame");
03:   for(var i=1; i<=5; i++) {
04:     photoFrame.find(".imageArea").append(
    "<img src='images/flower" + i + ".jpg'>");
05:   }
06:
07:   // 5秒置きに次の画像へ移動
08:   setInterval(slide, 5000);
09:   // 以降の処理は削除
10: });
11:
12: function slide() {
13:   $("#photoFrame").find(".imageArea").animate(
14:     {"left" : "-=200px"}, "fast", "linear", next);
15: }
16:
17: currentIndex = 1;
18:
19: function next() {
20:   // 隠れているほうのimageAreaのインデックスを算出
21:   var otherIndex = 1 - currentIndex;
22:   if(parseInt($("#imageArea" + currentIndex).
    css("left")) == -800) {
23:     // 末尾の画像を表示したときの処理
24:     // 隠れているほうのimageAreaのleftプロパティをphotoFrameの右側に設定
25:     $("#imageArea" + otherIndex).css("left", "200px");
26:   }
27:   else if(parseInt($("#imageArea" + currentIndex).
    css("left")) == -1000) {
28:     // imageAreaが切り替わったタイミングでカレントインデックスを変更
29:     currentIndex = otherIndex;
30:   }
31:   // toEnabled(); 削除
32: }
33: // その他の関数は削除
```

解説

前述のサンプルとの相違点は、「次へ」ボタンクリック時の処理をsetIntervalメソッドで定期実行している点です。これにより、ユーザーが何も操作しなくても一定間隔で画像がスライドしていきます。