ワンポイント 一定時間後に一度だけ実行するタイマー処理

タイマー処理にはsetIntervalメソッドの他にsetTimeout()というメソッドが用意されています。使い方は基本的に同じですが、こちらは、一定間隔で処理を繰り返すのではなく、一定時間後に一度だけ処理を実行します。第1引数に実行したい処理を記述した関数を、第2引数に、実行までの待ち時間（ミリ秒）を指定します。

一定時間ごとの繰り返し処理はsetIntervalメソッドで記述するのが一般的ですが、一回の処理が繰り返し時間内に終わり切らなかった場合、前の処理が終了する前に次の処理を開始してしまう危険性があります。そんなときには、setTimeoutメソッドを使用して繰り返し処理を記述することもできます。

setIntervalメソッドと違って必ず「処理が終了してから」間隔をあけて次の処理を実行するので、実行時間がわからないときは、こちらの書き方が便利です。

▶ setTimeoutメソッド

```
var timer = setTimeout(timerfunc, 1000);
```

タイマー識別用の変数　実行したい関数　待ち時間

▶ clearTimeOutメソッド

```
clearTimeOut(timer);
```

タイマー識別用の変数

▶ setTimeoutメソッドによる繰り返し処理の例

```
function foo(){
  //setTimeoutメソッドで1秒後に関数fooを呼び出す
  setTimeout(foo, 1000);
  console.log('繰り返し');
}
foo();
```


Chapter

8

データをまとめて扱おう

この章では、データをまとめて扱うことのできる「配列」という仕組みや、これまで利用してきた「オブジェクト」を自分で作る方法について学びます。

Lesson 44 [イベントの概要]

データをまとめて扱いやすくしましょう

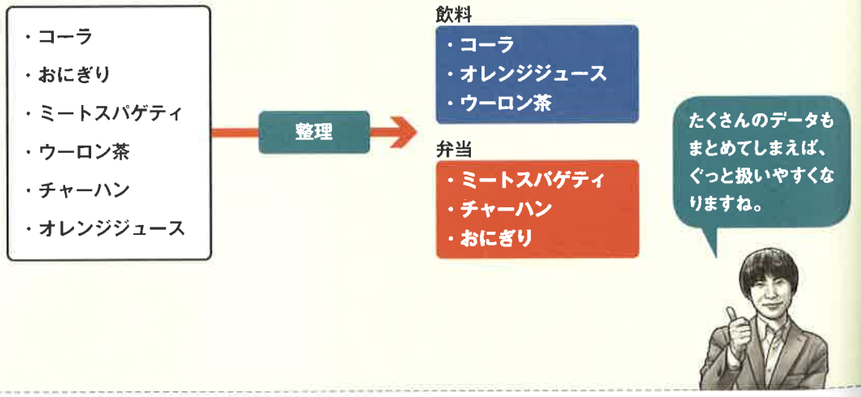
このレッスンのポイント

Lesson 15で、データを扱う際は「変数」に代入して、記憶する必要があることを学びましたね。でも、データごとにたくさんの変数を作ると、管理が大変になってしまいます。このレッスンでは、複数のデータをまとめて扱う方法について学んでいきます。

膨大なデータでもまとめると扱いやすい

私たちは普段から膨大なデータを扱っています。例えば、コンビニには1店舗あたり約3,000点の商品が並んでいるそうです。コンビニの小さな店舗にそれだけの商品があることも驚きですが、3,000点もの商品の中から目的の商品を見つけられる仕組みは素晴らしいですよね。コンビニでは、お客さんが商品を見つけやすいよう「雑誌」「お弁当」「飲料」など、種類ごとにまとめて陳列する工夫をしています。以下の図では、商品名がバラバラな状態と、種類ごとにまとめて整理された状態を比較して掲載しています。整理された後のほうが、ずっと選びやすいですよね。コンビニの商品にかぎらず、多くのデータを扱う場合は、データをまとめることで、より扱いやすくすることができます。

▶ データを分類してわかりやすくする



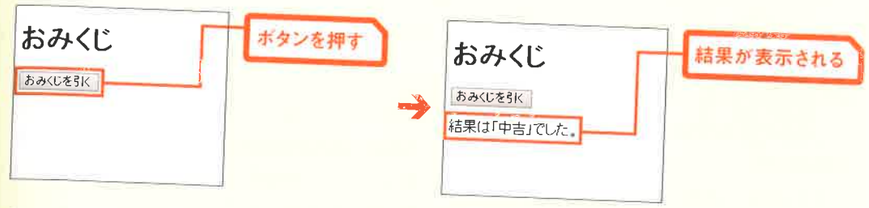
配列とオブジェクトを理解しよう

この章では、データをまとめて扱う方法を学びます。具体的には、**同種のデータを手軽にまとめることのできる「配列」という仕組みと、データや関数をまとめて扱うことのできる「オブジェクト」の作り方について学びます。**

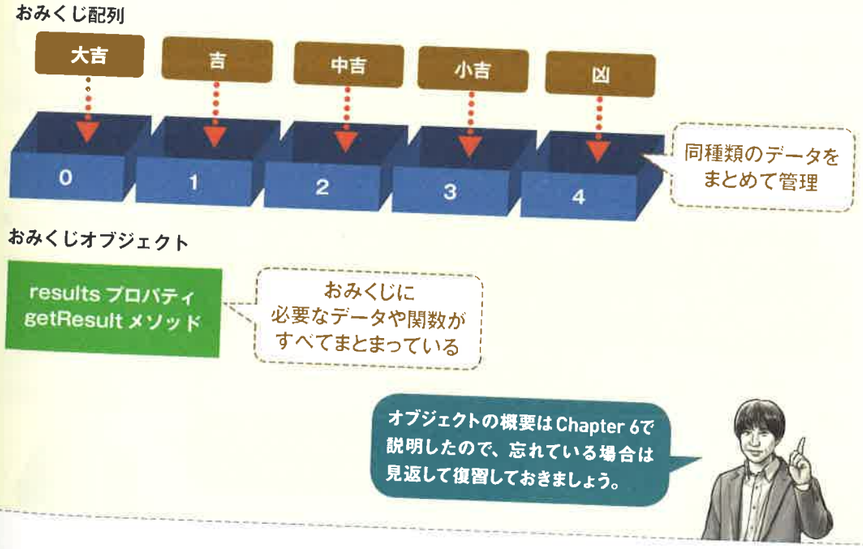
また章の最後では、学んだことを実践して「おみくじ」のプログラムを作成します。おみくじのプログラムでは、「大吉」「中吉」……といったおみくじの結果となるデータを「配列」で管理して、おみくじの結果をランダムに決定する機能を「オブジェクト」として提供できるようにします。

新しい言葉がたくさん出てきて、わからないことがあっても大丈夫です。この章を通じて1つずつ学んでいきましょう。

▶ おみくじプログラムのサンプルイメージ



▶ おみくじ配列とおみくじオブジェクト



オブジェクトの概要はChapter 6で説明したので、忘れている場合は見返して復習しておきましょう。

Lesson 45 [配列]

配列でデータをまとめましょう

このレッスンのポイント

データをまとめることで、扱いやすくなることは理解できましたか？このレッスンではさっそく、データをまとめて表現できる「配列」という仕組みについて学んでいきます。配列はfor文と組み合わせて利用することが多いので、その具体例も確認していきましょう。

配列でデータをまとめる

コンビニの例では、商品を種類ごとに分けて、まとめて陳列していることを学びました。プログラムにも、データをまとめて表現するための「配列」という仕組みが用意されています。

コンビニの商品をデータとすると、配列はちょうど「棚」のようなイメージで、配列の中にデータを並べて収め、まとめて管理することができます。例えば、飲料名のデータを納めた配列「drink」を作るには、以下のように記述します。

▶ 飲料名をまとめた配列「drink」を作成

```
var drink = ['コーラ', 'オレンジジュース', 'ウーロン茶'];
```

変数名　データ1　データ2　データ3

配列 drink

配列は英語で「Array」といい、「ずらりと並んだもの」という意味があります。

配列のデータにアクセスする

配列に所属するデータは、前から順に0,1,2,3……というインデックス（管理番号）が振られます。

配列に所属するデータにアクセスするには、このインデックスの番号を配列名の後の[ ]の中に指定してあげればOKです。

▶ インデックスを指定して配列内のデータにアクセスする

```
drink[0];
```

配列名　インデックス

配列のインデックスは「1」ではなく「0」からスタートするので注意しましょう！

▶ 配列内のデータを利用する

```
// 飲料名をまとめた配列「drink」を作成
var drink = ['コーラ', 'オレンジジュース', 'ウーロン茶'];

// インデックスが「0」のデータにアクセス
console.log(drink[0]); ……コンソールに「コーラ」と表示される
```

配列に所属するデータの数を調べる

配列は「オブジェクト型」のデータで、所属するデータの数を表す「length」プロパティを持っています。

配列に所属するデータの数を調べるには、「配列名.length」という形式で、「length」プロパティの値を参照します。

▶ lengthプロパティの利用例

```
// 飲料名をまとめた配列「drink」を作成
var drink = ['コーラ', 'オレンジジュース', 'ウーロン茶'];

// 配列に所属するデータの数を表示
console.log(drink.length);……コンソールに「3」と表示される
```

lengthプロパティで調べた要素数は、for文の繰り返し条件などに使われます。

おみくじの結果を「配列」で扱ってみる

1 配列を作成する

08/array/practice/js/app.js

今回は、この章の最後に作成する「おみくじ」の準備体操として、おみくじの結果となるデータを配列で扱ってみましょう。このレッスンのapp.jsファイルをBracketsで開いて、以下のコードを記述してください。

まずは、おみくじの結果である「大吉」「吉」「中吉」……といった文字列データを持つ配列「results」を作成します❶。

```
001 //_おみくじの結果データを作成
002 results_=_['大吉','吉','中吉','小吉','凶'];
```

1 配列を作成

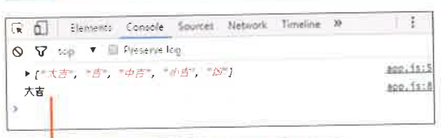
2 配列の内容を確認する

次に、配列ができていることを確認するため、配列そのものと、配列の最初の要素（インデックスが「0」）をコンソールに表示する処理を記述してみましょう❶。プログラムが完成したら、内容を上書き保存して、index.htmlをブラウザで開いてコンソールを確認しましょう。

コンソールで配列やオブジェクトを表示すると、中身が折りたたまれて、名前と概要だけが表示されます。「▶」マークをクリックすると、さらに詳細な情報が表示され、インデックスの振られたデータの内容や、lengthの値も確認できます。

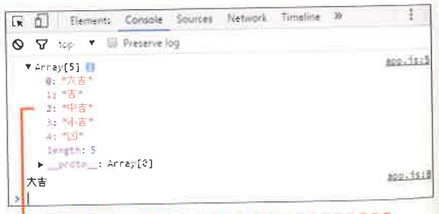
```
003
004 //_配列「results」をコンソールに表示
005 console.log(results);
006
007 //_インデックスが「0」の要素をコンソールに表示
008 console.log(results[0]);
```

1 配列を表示



resultsの内容と最初のデータが表示された

ここでArray [5] と表示された場合はリロードする



「▶」をクリックすると、resultsの内容が表示される

3 配列のデータをすべて表示する

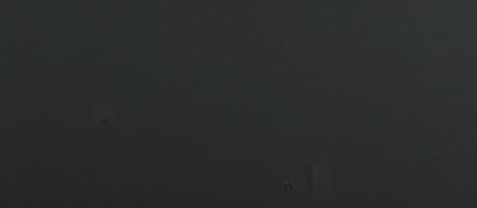
続いて、配列に所属するすべてのデータを表示してみましょう。配列のデータを表示するには「配列[インデックス]」の形式で記述します。

インデックスの値は、「0」から順に増えるので、for文（Lesson 31参照）を使って0～最大値までの繰り返しを記述します❶。

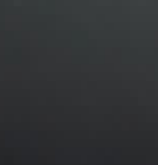
配列のインデックスの最大値は、配列のデータ数より1つ小さくなるので、for文の条件式は、カウント用の変数の値が「results.length」未満になるように設定すればOKです。

```
010 //_配列に所属するデータをfor文ですべて表示
011 for_(var_i_=_0;_i_<_results.length;_i++)_{
012 __console.log(results[i]);
013 }
```

1 for文を追加



配列のデータすべてが表示された



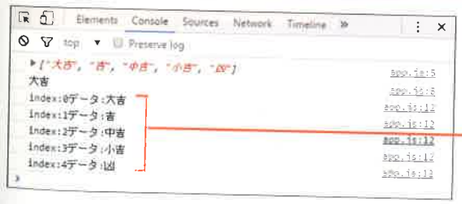
配列のデータを繰り返し処理で取り出すテクニックは非常によく使用するので、慣れておくとよいでしょう。

4 データを見やすくする

先ほどの結果をより見やすくするために、データのインデックスもあわせて表示するように変更してみましょう❶。プログラムが完成したら、内容を上書き保存して、index.htmlをブラウザで再読み込みして動作を確認しましょう❷。

```
010 //_配列に所属するデータをfor文ですべて表示
011 for_(var_i_=_0;_i_<_results.length;_i++)_{
012 __console.log('index:'_+_i_+_'データ:'_+_results[i]);
013 }
```

1 インデックスも表示



インデックスとデータすべてが表示された

Lesson 46 [オブジェクトの作成]

オブジェクトでデータをまとめましょう

このレッスンのポイント

配列の仕組みは理解できましたか？ これまでに配列以外にも、データを扱いやすくまとめる「オブジェクト」の概念をChapter 6で学びました。ここでは、オブジェクトと配列を比べながら理解を深め、オブジェクトを自分で作成する方法を学びます。

オブジェクトのおさらい

Chapter 6の冒頭で説明したように、オブジェクトは、テーマに沿って変数や関数などのデータをまとめ、データを扱いやすくしたものです。

配列とオブジェクトの違いは、配列が「インデックスを使って所属するデータを参照する」のに対して、オブジェクトは「名前（プロパティ）を使って所属するデータを参照する」という点です。

コンビニの商品の例のように、横並びのデータを扱う場合は配列が便利ですが、「自動車の高さ」「自動車の幅」という具合に、異なる性質のデータを1つのテーマでまとめる場合はオブジェクトが適しています。

オブジェクトを自分で定義するには、{ }で囲んだ範囲に、プロパティ名と、対応するデータを記述していきます。

▶ オブジェクトの定義

```
var human = {
  name: '山田一郎',
  age: 31,
  ……
}
```

変数名　オブジェクトの始まり
プロパティ名1　データ1
プロパティ名2　データ2
オブジェクトの終わり

▶ 配列とオブジェクト

配列 [大吉 , 吉 , 中吉 , 小吉 , 凶]

オブジェクト
{
　幅 ： 1695mm ,
　長さ： 3895mm
}

データを[ ]で囲むと配列、{ }で囲むとオブジェクトが作られます。

独自のメソッドを作る

Chapter 6の冒頭で、メソッドは、プロパティの一種で、オブジェクトの操作や振る舞いを記述した関数であると学びましたね。

このメソッドを独自に定義したい場合は、オブジェクトの定義の中で「メソッド名:」の後に無名関数（P.108参照）を記述します。メソッドもプロパティの一種なので、基本的な書き方は変わりません。

プロパティ名の部分をメソッド名に、データの部分を無名関数に置き換えて記述すればOKです。

下の例では「おみくじ」オブジェクトの定義の中で「くじを引く」メソッドを定義しています。これで「おみくじ.くじを引く()」という呼び出し方でくじを引くことができます。

▶ メソッドの定義

```
var 変数名 = {
  メソッド名1: function(引数1, 引数2…) {
    //実行したい処理
  },
  …
}
```

▶ メソッドのイメージ

```
var おみくじ = {                          「おみくじ」オブジェクトを定義
  くじを引く: function() {                「くじを引く」メソッドを定義
    //くじを引くメソッドの定義として、結果を返す処理を書く
  }
}

//くじを引くメソッドの結果をコンソールに表示
console.log(おみくじ.くじを引く());      「くじを引く」メソッドを呼び出す
```

オブジェクトの内容を思い出せましたか？　実践パートでは、実際に「おみくじ」オブジェクトを作ってみましょう。

NEXT PAGE

オブジェクトを使って「おみくじプログラム」を作る

1 HTMLファイルを編集する 08/fortune/practice/index.html

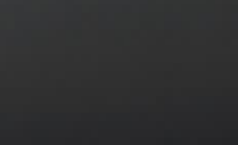
この章の集大成として、オブジェクトと配列を使った「おみくじプログラム」を作りましょう。まずは、おみくじの結果を表示する画面と、おみくじを引くためのボタンを作ります。

このレッスンのindex.htmlファイルをBracketsで開いて、以下のコードを記述し上書き保存してください。

「おみくじ」という見出しをh1要素で用意し①、「おみくじを引く」ボタンをinput要素で用意します②。「おみくじを引く」ボタンの要素には、後でJavaScriptでイベントを登録するためにid属性を付与しています。最後に、おみくじの結果を表示するための要素として、id属性を付与したdiv要素を設置しています③。

```
008 <body>
009 ··<h1>おみくじ</h1>
010 ··<p><input·type="button"·id="getResult"·value="おみくじを引く"></p>
011 ··<div·id="result"></div>
012 ··<script·src="js/app.js"></script>
013 </body>
```

1 見出しを追加
2 ボタンを追加
3 結果を表示するdiv要素を追加



おみくじの画面ができた

2 JavaScriptファイルを編集する 08/fortune/practice/js/app.js

続いて、おみくじに関するデータをまとめた「おみくじオブジェクト」を作成していきます。このレッスンのapp.jsファイルをBracketsで開いて、以下のコードを記述し上書き保存してください。

まずは、omikujiオブジェクトを定義して、{}内にプロパティを記述していきます④。最初のプロパティとして、おみくじの結果をまとめた配列「results」を定義します⑤。

```
001 //·おみくじオブジェクトの定義
002 var·omikuji·=·{
003 ··results:·["大吉","吉","中吉","小吉","凶"]
004 }
```

1 オブジェクトを定義
2 プロパティを定義

3 くじの結果を返すメソッドを作る

続いて、おみくじの結果を表示するためのメソッド「getResult」を定義していきましょう⑥⑦。おみくじの結果のデータは同じオブジェクトの「results」プロパティにまとめられています。

オブジェクトを定義する際に、自分自身のプロパティにアクセスする際は、「this.プロパティ名」でアクセスできるので、「var results = this.results」として、結果のデータを取得しています⑧。後は、結果をまとめた配列の中から、ランダムに1つのデータを選んで返す処理を記述します⑨。

```
001 //·おみくじオブジェクトの定義
002 var·omikuji·=·{
003 ··results:·["大吉","吉","中吉","小吉","凶"],
004 ··getResult:·function()·{
005 ····var·results·=·this.results;
006 ····return·results[Math.floor(Math.random()·*·results.length)];
007 ··}
008 }
```

1 カンマを追加
2 メソッドを定義
3 プロパティにアクセス
4 データを返す

プロパティを追加した際は、カンマ「,」の追加を忘れないようにしましょう。

Point 配列からランダムに1つのデータを取得する

配列からランダムに1つのデータを取得するには、Math.randomメソッド（P.88参照）を利用してインデックスを求めます。Math.randomメソッドは0以上1未満の値をランダムに作り出すので、それに配列のデータ数を掛ければ、「0以上～配列のデータ数未満」の値となります。さらにMath.floorメソッドで整数値にすれば、配列のインデックスとして利用できます。

```
配列名[Math.floor(Math.random() * 配列名.length)];
```

NEXT PAGE

4 おみくじオブジェクトを動作確認する

ここまでのプログラムが完成したら、作成したメソッドが問題なく利用できるか、結果をコンソールに表示して動作確認を行ってみましょう。最後の行に、コンソールに結果を表示する以下のプログラムを記述し❶、内容を上書き保存して、index.htmlをブラウザで開きます。ブラウザを読み込むたびに、ランダムなおみくじ結果がコンソールに表示されればバッチリです。もし問題が見つかった場合は、いままでのプログラムを見直してみましょう。
動作確認が終わったら、今回追記した確認用のプログラムは削除しておきましょう。

```
008 }
009
010 console.log(omikuji.getResult());
```

1 コンソールに表示

ブラウザを更新すると、ランダムに結果が表示される

5 ボタンが押されたときにくじを引く

「おみくじを引く」ボタンが押されたときに、おみくじの結果が表示されるようにしましょう。
まずは、「おみくじを引く」ボタンの要素をid属性の値「getResult」を使用して取得します。結果を表示するための要素をid属性の値「result」を使用して取得します❶。
次に、取得した「おみくじを引く」ボタンの要素、「getResult」にイベントリスナーを登録して、クリックイベントが発生したときに処理を実行できるようにします❷。処理の中では、先に作成した「omikuji.getResult()」メソッドを使っておみくじの結果を取得し、「result」の「innerHTML」プロパティを上書きして、結果を表示しています❸。

```
001 //_要素オブジェクトの取得
002 var_getResult_=_document.getElementById('getResult');
003 var_result_=_document.getElementById('result');
004
005 //_イベントの登録
006 getResult.addEventListener('click',_function(){
007 __result.innerHTML_=_'結果は「'_+_omikuji.getResult()_+_'」でした。';
008 });
009
010 //_おみくじオブジェクトの定義
```

1 要素を取得
2 イベントリスナーを登録
3 結果を表示

6 プログラムが完成した

プログラムが完成したら、内容を上書き保存して、index.htmlをブラウザで再読み込みして動作を確認しましょう❶。

NEXT PAGE

ワンポイント thisの意味は利用する場面で変わる

P.173で使用した「this」は状況によって指し示すものが変わる特殊なキーワードです。大きく分けると、関数の中で使うthisと、関数の外で使うthisで表すものが違います。

関数定義の外で使ったthisは、windowオブジェクトを指します。以下のように、比較演算子で厳密な比較を行っても真（true）になります。

メソッドの定義内で使ったthisは、メソッドが所属するオブジェクトを示しています。Lesson 46では、omikujiオブジェクトのgetResult()メソッドから、同じomikujiオブジェクトのresultsプロパティを利用するためにthisを利用しました。このとき、thisはomikujiオブジェクトを指しています。

上級者向けの内容になるので本書では扱いませんが、関数内のthisは、関数の呼び出し方によってもthisの値が指すものが変化します。メソッド定義内で使用した場合以外にもいくつかパターンがあるのです。さらに詳しくthisについて知りたい場合は、Chapter 13で紹介しているMDNのリファレンスで調べてみるといいでしょう。

▶ 関数の外部で使うthisはwindowオブジェクトを指す

```
console.log(this === window);
```
厳密に等しいので結果はtrue

▶ メソッド定義の中で使うthisは所属するオブジェクトを指す

```
// おみくじオブジェクトの定義
var omikuji = {
  results: ["大吉","吉","中吉","小吉","凶"],
  getResult: function() {
    var results = this.results;
    return results[Math.floor(Math.random() * results.length)];
  }
}
```
このthisはomikujiオブジェクトを指す

Chapter

9

フォトギャラリーを作成しよう

この章では、これまでの学習の集大成として、フォトギャラリーを作成します。実践を通じて学んだ内容を復習しながら、確かな力にしていきましょう。