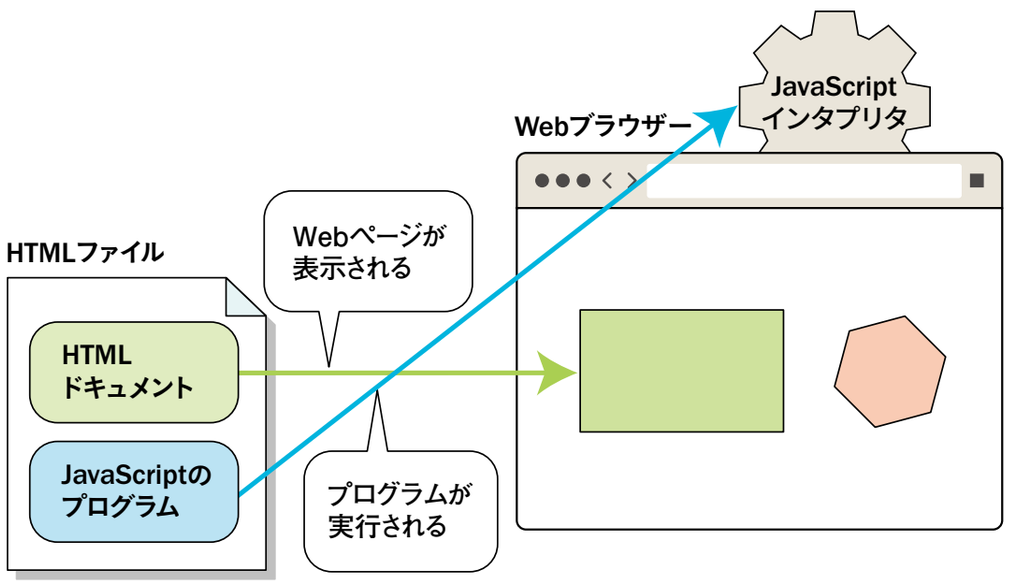
2 JavaScriptとは — Webブラウザーで実行される言語

完成ファイル | なし

予習 JavaScriptについて

本書で紹介するJavaScriptは、インタプリタ方式の高級言語に分類されるプログラミング言語です。したがって、ソースプログラムが、1行ずつマシン語に変換されながら実行されます。なお、JavaScriptのプログラムは基本的にHTMLファイル内に記述します。HTMLファイルがWebブラウザーに読み込まれると、Webブラウザーに内蔵されたインタプリタによってプログラムが解釈され、実行されます。



この節では、JavaScriptの概要について説明しましょう。

理解 JavaScriptの基本概要を理解する

JavaScriptの誕生

JavaScriptは、Mozilla Firefoxの前身であるWebブラウザー「Netscape」の開発元であるNetscape社（当時）と、Sun Microsystems社（現Oracle社）によって1990年代半ばに開発されたプログラミング言語です。当時のWebページは、単に文字情報や画像を表示するだけの静的なものがほとんどでした。そのWebページを、より動的でかつインタラクティブなものにすることを目的に、JavaScriptが開発されました。JavaScriptを活用することによって、Webブラウザーでさまざまなプログラムを実行させることが可能になります。
JavaScriptの名前の由来は、当時インターネットを中心に大きな注目を集めていたオブジェクト指向のプログラミング言語Javaです。この「Java」と「Script」を合わせて、「JavaScript」と命名されました。

「Script」（スクリプト）には演劇の台本という意味がありますが、プログラミング言語の世界では、比較的小さなプログラムを作成するのに使用される、インタプリタ方式の言語のことをスクリプト言語と呼びます。JavaScriptは、当初は、本格的なプログラミング言語であるJavaの簡易版のようなイメージで認知されていました。ただし、記述方法に似た部分があるものの、Javaとの互換性は全くないので注意しましょう。

JavaScriptはどこで実行されるのか

JavaScriptのソースプログラムは通常のテキストです。HTMLファイルの内部にscriptエレメントとして直接記述するか、あるいはプログラムだけを別のテキストファイルとして保存してHTMLファイルから読み込みます。
なお、JavaScriptの動作環境であるインタプリタは、Google ChromeやMicrosoft Edgeなどの Webブラウザーの内部に用意されています。したがって、JavaScriptに対応したWebブラウザーがあれば、WindowsやmacOSといったOSを問わずに、同じように実行できます。

3 プログラムを読みやすくする
― スペースの挿入とコメント

完成ファイル [0203]→[sample1e.html]

予習 ステートメントの記述方法とコメントについて

JavaScriptに限らず最近のプログラミング言語は、自由度の高い書き方ができます。たとえば、「+」演算子の前後には半角スペースを入れてもかまいません。そうすることによって、ステートメントが読みやすくなるでしょう。

```
alert(4+5);
```

→

```
alert(4 + 5);
```

スペース

なお、プログラムの途中には**コメント**と呼ばれる説明文を記述できます。コメントは、実行時には無視されます。

```
/*
    sample1.html 2024/5/6
*/
//alert("ようこそJavaScriptの世界へ");
alert(4 + 5);
alert(10 - 2);
```

コメントは実行されない

ステートメントは実行される

体験 スペースやコメントを使う

1 演算子の前後にスペースを入れる

前節で作成したsample1.htmlを編集して、「+」演算子と「-」演算子の前後にスペースを入れてみましょう❶。

Tips

使用できるのは**半角スペース**のみです。**全角スペース**は使用できないので注意してください。

```
<!DOCTYPE html>
<html lang="ja">
<head>
    <meta charset="utf-8">
    <title>JavaScriptサンプル</title>
</head>

<body>
    <script>
        alert("ようこそJavaScriptの世界へ");
        alert(4 + 5);
        alert(10 - 2);
    </script>
</body>
</html>
```

❶修正する

```
<script>
    alert("ようこそJavaScriptの世界へ");
    alert( 4 + 5 );
    alert( 10 - 2 );
</script>
```

2 ステートメントをコメントにする

最初のalert命令の先頭に、「//」（スラッシュ「/」を2つ並べて記述）を挿入します❶。これで、このalert命令がコメントになります。

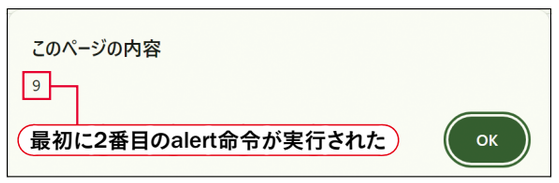
```
<!DOCTYPE html>
<html lang="ja">
<head>
    <meta charset="utf-8">
    <title>JavaScriptサンプル</title>
</head>

<body>
    <script>
        //alert("ようこそJavaScriptの世界へ");
        alert(4 + 5);
        alert(10 - 2);
    </script>
</body>
</html>
```

❶修正する

```
<script>
    //alert("ようこそJavaScriptの世界へ");
    alert( 4 + 5 );
    alert( 10 - 2 );
</script>
```

3 実行結果を確認する

ファイルを保存し、Webブラウザーで実行します。最初のalert命令が実行されないことを確認してください。



2 変数で文字列を扱う ― 文字列と数値の違い

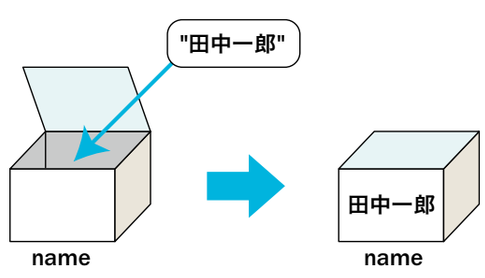
完成ファイル [0302]→[sample1e.html]

予習 変数には文字列も入れられる

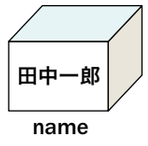
前節では、変数を宣言し、値として数値を代入しました。**変数**という名前からすると、その中には数しか格納できないようなイメージがあるかもしれません。実は、プログラミングで使う変数には、いろいろな種類の値を代入することが可能です。ここでは、変数で**文字列**を扱う方法について説明しましょう。

変数「name」に"田中一郎"を代入

```
name = "田中一郎";
```



数値の場合「+」演算子は足し算をする演算子でした。この「+」演算子を文字列に対して使用すると、「+」の左右の文字列を連結することができます。




```
"私は" + name + "です"  →  "私は田中一郎です"
```


体験 変数で文字列を操作しよう

1 変数に文字列を格納する

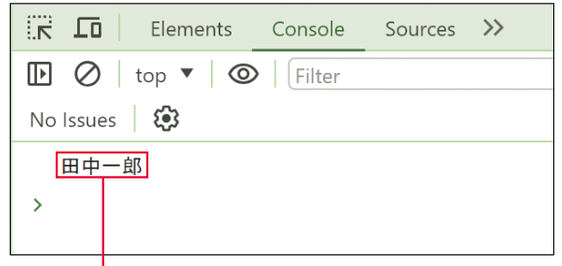
前節のsample1.htmlを編集します。まず、スクリプト最下行のconsole.logメソッドを削除します。代わりに、変数**name**を宣言して文字列を代入します❶。続いて次の行で、console.logメソッドで変数nameの値を表示します❷。

```
 2  <html lang="ja">
 3  <head>
 6  </head>
 7
 8  <body>
 9      <script>
10          let age;
11          age = 20;
12          age = age + 1;
13          const name = "田中一郎";
14          console.log(name);
15      </script>
16  </body>
17  </html>
```

```
<script>
    let age;
    age = 20;
    age = age + 1;
    const name = "田中一郎";   ❶入力する
    console.log(name);         ❷入力する
</script>
```

2 プログラムを実行する

ファイルを上書き保存し、実行してみましょう。「田中一郎」と表示されます。

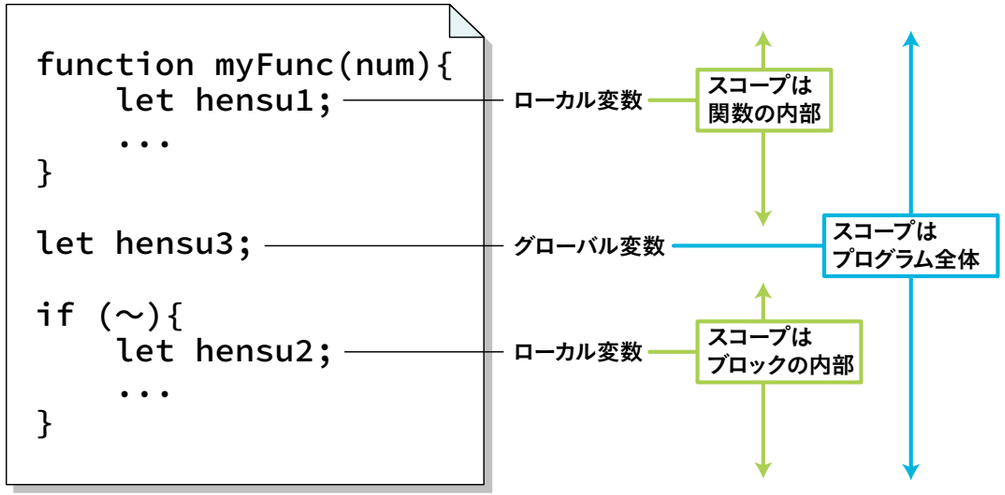


2 変数の有効範囲を知る ― ローカル変数とグローバル変数

完成ファイル [0502]→[sample2e.html]

予習 変数の有効範囲を知る

個々の変数には、その変数を利用できる範囲が決まっています。それを、変数の**スコープ**と呼びます。関数の内部など、「{」と「}」で囲まれた範囲をブロックといいますが、ブロック内でletやconstで宣言した変数のスコープは、ブロックの内部だけになります。これを**ブロックスコープ**と呼びます。また、ブロックの内部で宣言された変数を**ローカル変数**、ブロックの外部で宣言された変数を**グローバル変数**といいます。



ここでは、実際に試しながら、ローカル変数とグローバル変数の違いを確認していきましょう。

体験 スコープをテストする

1 作業用のファイルを用意してテスト用の関数を作成する

エディターで「0502」フォルダーの「template2.html」を開き、「sample2.html」といった名前で保存します。bodyエレメントに空のscriptエレメントが用意されています。テスト用の関数「**testFunc**」を定義し、ローカル変数numを宣言し、console.logメソッドで表示します❶。値を「2」に設定している点に注目してください。

```
    <meta charset="utf-8">
    <title>変数のスコープ</title>
</head>

<body>
    <script>
        function testFunc() {
            let num = 2;
            console.log("in func → " + num);
        }
    </script>
</body>
</html>
```

```
<script>
    function testFunc() {
        let num = 2;
        console.log("in func → " + num);
    }
</script>
```

❶ 入力する

Tips

ここで作成したtestFunc関数には、引数がありません。その場合でも、関数名の後には括弧「( )」が必要です。また、testFunc 関数は値を戻さないので、return文は不要です。

2 テスト用のブロックを用意する

次に、テスト用のブロックを用意します。内部で変数numを宣言し値を「3」に設定してconsole.logメソッドで表示しています❶。

```
<body>
    <script>
        function testFunc() {
            let num = 2;
            console.log("in func → " + num);
        }
        {
            let num = 3;
            console.log("in block → " + num);
        }
    </script>
</body>
```

```
<script>
    function testFunc() {
        let num = 2;
        console.log("in func → " + num);
    }
    {
        let num = 3;
        console.log("in block → " + num);
    }
</script>
```

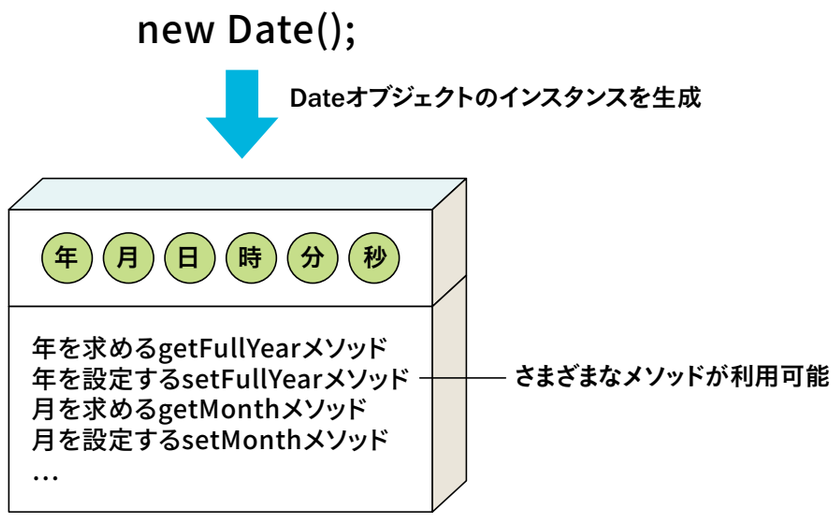
❶ 入力する

2 日付や時刻を操作する — Dateオブジェクト

完成ファイル [0602]→[sample2e.html]

予習 Dateオブジェクトのメソッド

前節では、**new演算子**と**Dateコンストラクター**を使用して、利用可能なオブジェクトである**インスタンス**を生成する方法について学びました。
いったんインスタンスを生成すると、オブジェクトに用意されているメソッドやプロパティが利用可能になります。Dateオブジェクトの個々のインスタンスは、内部に年、月、日、時、分、秒といった日付時刻に関するデータを持っています。また、それらのデータを処理するために、さまざまな日付時刻に関するメソッドが用意されています。たとえば**getFullYearメソッド**を使用することで、年を西暦4桁の数値で戻すことができます。



この節では、Dateオブジェクトのメソッドを使って、日付を「xxxx年x月x日」の形式で表示する方法と、今年の残り日数を計算する方法を説明しましょう。

体験 Dateオブジェクトのメソッドを使う

1 今日の日付を「xxxx年x月x日」の形式で表示する

エディターで「0602」フォルダーの「template2.html」を開き「sample2.html」といった名前で保存しておき、scriptエレメントにステートメントを入力していきます。変数nowとdateを宣言し❶、現在の日付時刻を表すインスタンスを生成して変数nowに格納します❷。Dateオブジェクトの年、月、日を戻すメソッドを呼び出して「+」演算子で接続し、変数dateに格納してh1エレメントとして表示します❸。

```
<!DOCTYPE html>
<html lang="ja">
<head>
    <meta charset="utf-8">
    <title>Dateオブジェクトのメソッドを使う</title>
</head>

<body>
    <script>
        let now, date;
        now = new Date();
        date = now.getFullYear() + "年" +
            (now.getMonth() + 1) + "月" +
            now.getDate() + "日";
        console.log(date);
    </script>
</body>
</html>
```

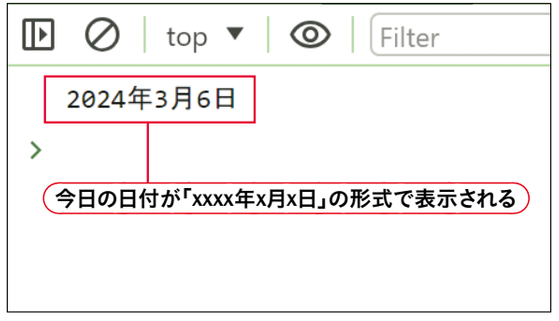
Tips
1つのステートメントが長くなる場合は、演算子の前後などに改行やインデントを入れるとわかりやすくなります。

```
<script>
    let now, date;          ❶入力する
    now = new Date();       ❷入力する
    date = now.getFullYear() + "年" +
        (now.getMonth() + 1) + "月" +
        now.getDate() + "日";
    console.log(date);
</script>
```

❸入力する

2 プログラムを実行する

ファイルを上書き保存し、プログラムを実行します。今日の日付が「xxxx年x月x日」の形式で表示されます。



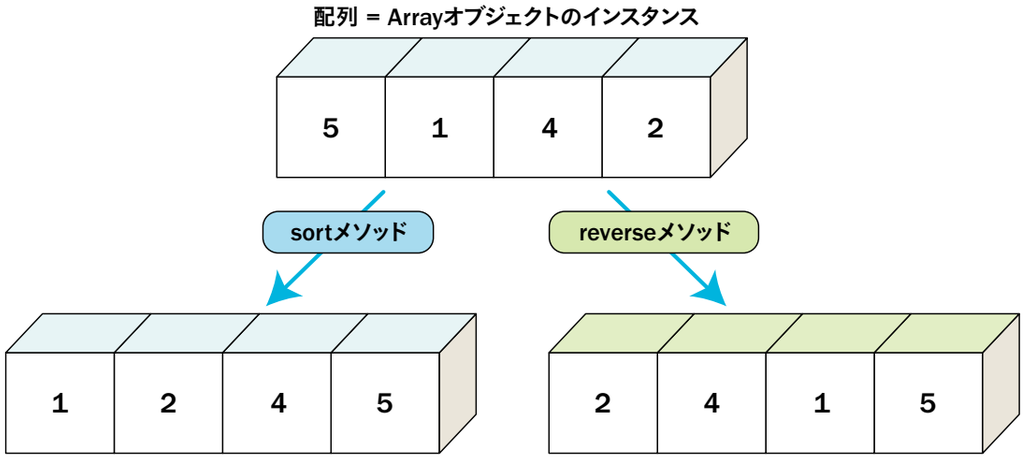
3 配列を操作する — Arrayオブジェクトのメソッド

完成ファイル [0703]→[sample3e.html]

予習 Arrayオブジェクトの活用

JavaScriptの配列は、Arrayオブジェクトのインスタンスです。配列をオブジェクトとして扱うメリットの1つは、あらかじめ用意された便利なメソッドを利用できる点にあります。

たとえば、**sort**メソッドを使うと、配列の要素を並び替えることができます。また、**reverse**メソッドを使うと、配列の要素の並びを逆順にすることが可能です。



ここでは、Arrayメソッドの使用例として、年齢が複数格納された配列agesを用意し、要素を連結した文字列を戻す**join**メソッドと、要素を並び替える**sort**メソッドを使ってみましょう。

体験 配列の要素の連結と並び替え

1 要素を連結する

エディターで「0703」フォルダーの「template3.html」を開いて「sample3.html」といった名前で保存し、bodyエレメントのscriptエレメントにステートメントを入力します。まず、複数の年齢を要素とする配列を生成し、変数agesに代入します❶。次に、**join**メソッドで要素を接続した文字列を作り、表示します❷。

```
<!DOCTYPE html>
<html lang="ja">
<head>
    <meta charset="utf-8">
    <title>Arrayオブジェクトのメソッド</title>
    <script>
    </script>
</head>

<body>
    <script>
        const ages = new Array(4, 6, 10, 24, 1, 11, 40);
        console.log(ages.join(" > "));
    </script>
</body>
</html>
```

```
<script>
    const ages = new Array(4, 6, 10, 24, 1, 11, 40);   ❶入力する
    console.log(ages.join(" > "));                     ❷入力する
</script>
```

Tips

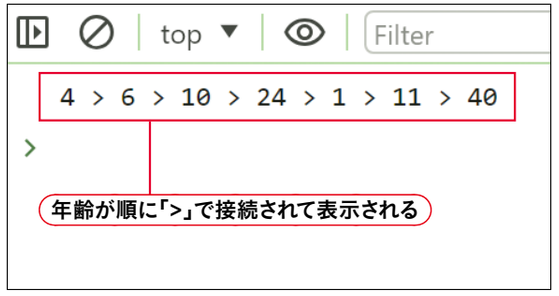
joinメソッドの引数には、要素の間にはさむ文字列を指定します。この例では「>」を指定しているため、次のような文字列になります。

要素1 > 要素2 > 要素3 > ..... > 最後の要素

引数を指定しない場合はカンマ「,」が間にはさまります。

2 プログラムを実行する

ファイルを上書き保存し、プログラムを実行します。配列agesに格納された年齢が、文字列「>」で接続されて順番に表示されます。



1 ドキュメント内のエレメントにアクセスする
— DOMの概要と基本操作

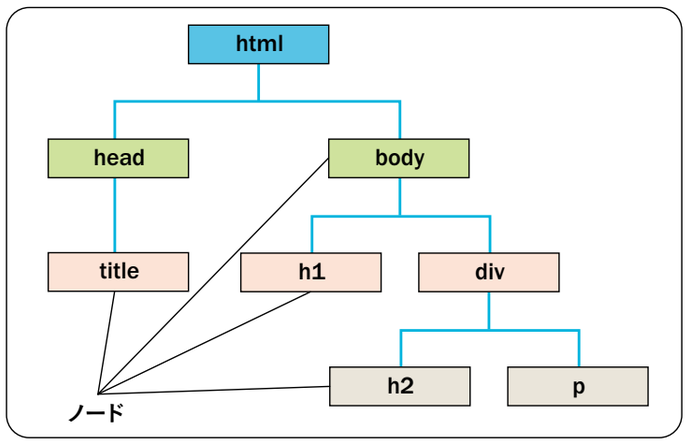
完成ファイル [0801]→[sample1e.html]

予習 DOMを理解する

HTMLドキュメントやXMLドキュメントのすべてのエレメントに、外部から階層構造でアクセスできるようにした仕組みのことを、**DOM**（Document Object Model）といいます。DOMは、W3Cという標準化団体で規定されており、JavaScriptにはHTMLドキュメントDOMを操作するためのメソッドやプロパティが用意されています。

DOMから見たHTMLの階層構造を、**DOMツリー**と呼びます。 〜 や<h1>〜</h1>など、DOMツリー内の各エレメントを**ノード**と呼びます。

DOMツリー



DOM操作メソッド

getElementById
getElementByTagName
createElement
appendChild
....

DOM操作プロパティ

innerText
innerHtml
....

この節では、DOMを操作してWebページを動的に変更する例として、おみくじプログラムを作ってみましょう。

体験 HTML版のおみくじプログラムを作る

1 作業用のファイルを用意する

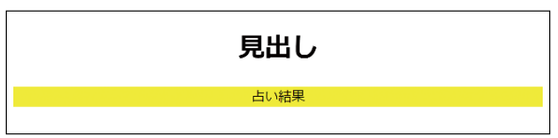
エディターで「0801」フォルダーの「template1.html」を開いて「sample1.html」といった名前で保存します。bodyにはid属性が「msg」のh1エレメント❶と、id属性が「myArea」の空のdivエレメント❷が用意されています。head部分ではbodyエレメントの「text-align」（テキスト配置）を「center」（中央揃え）に❸、id属性が「myArea」の「background」（背景色）を「yellow」（黄色）に❹するスタイルシートを設定しています。

```
<html lang="ja">
<head>
    <meta charset="utf-8">
    <title>おみくじプログラム</title>
    <style>
        body {
            text-align: center;
        }
        #myArea {
            background: yellow;
        }
    </style>
</head>

<body>
    <h1 id="msg">見出し</h1>
    <div id="myArea">占い結果</div>
    <script>
    </script>
</body>
</html>
```

```
<head>
      <meta charset="utf-8">
      <title>おみくじプログラム</title>
      <style>
            body {
                  text-align: center;
            }                                  ← ❸テキストを中央揃えに
            #myArea {
                  background: yellow;
            }                                  ← ❹背景を黄色に
      </style>
</head>

<body>
      <h1 id="msg">見出し</h1>                  ← ❶h1エレメント
      <div id="myArea">占い結果</div>           ← ❷divエレメント
      <script>
      </script>
</body>
```

2 Webブラウザーで表示する

この状態で、Webブラウザーで読み込むと、単にHTMLの内容が表示されます。



理解 Web Animations APIについて

animateメソッド

Web Animations API にはアニメーションのためのメソッドがいくつか用意されていますが、ここではシンプルな animate メソッドを使用する方法について説明します。
animateメソッドは、DOMの getElementById メソッドなどで取得したエレメントに対して次のように実行します。

▼書式

```
element.animate(キーフレームの設定, タイミングの設定);
```

＜体験＞の手順３では、まず写真のdivエレメントと「フェードイン」ボタンをgetElementByIdメソッドで取得しています。

```
const photo = document.getElementById("photo");   写真のdivエレメント
const fiBtn = document.getElementById("FiBtn");   「フェードイン」ボタン
```

animateメソッドの２つの引数は、キーと値のペアの連想配列形式で設定します。アニメーションの変化点をキーフレームといいますが、手順５ではキーフレームの設定を次のように変数keyframesに代入しています。

```
const keyframes = {
    opacity: [0, 1],          透明度のキーフレーム
    translate: [0, "150px 150px"]   座標のキーフレーム
};
```

opacity は、エレメントの非透明度を0〜1の値として管理するプロパティです。「0」で透明、「1」で非透明になります。上記のようにキーフレームの値を配列形式で「[0, 1]」と指定すると、0(透明）から1(非透明）へと変化します。
translate は、エレメントの位置や大きさを変化させるプロパティです。ここでは原点「0」から、X=150ピクセル、Y=150ピクセルの位置へと移動しています。座標を指定する場合、全体をダブルクォーテーション「"」で囲んだ文字列とし、値の末尾に単位「px」を追加します。また、値はスペースで区切ります。
タイミングの設定は、次のように変数timingOptsに代入しています。

```
const timingOpts = {
    duration: 1000,      実行時間(ミリ秒)
    fill: "forwards"     最後の位置で停止する
}
```

duration はアニメーションの時間で、ミリ秒単位で指定します。また、fill を「forwards」にすると、アニメーションの終了後の状態を保持します。これを指定しない場合、初期状態（ここでは非透明度が0）の状態に戻ります。
「フェードイン」ボタンのonclickイベントハンドラーでは、keyframesとtimingOptsの２つの変数を animate メソッドの引数にすることで、ボタンをクリックするとアニメーションを開始しています。

```
fiBtn.onclick = () => {
    photo.animate(keyframes, timingOpts);   アニメーションを開始する
}
```

「フェードアウト」ボタンのイベントハンドラー

アニメーションのためキーフレームやタイミングオプションが少ない場合には、animateコマンドの引数に直接設定しても構いません。「フェードアウト」ボタンのonclickイベントハンドラーでは、引数に直接設定しています。

```
foBtn.onclick = () => {
    photo.animate({ opacity: [1, 0] }, {
        duration: 1000,
        fill: "forwards"
    })
}
```

1 オリジナルのオブジェクトを定義する — クラスの基本

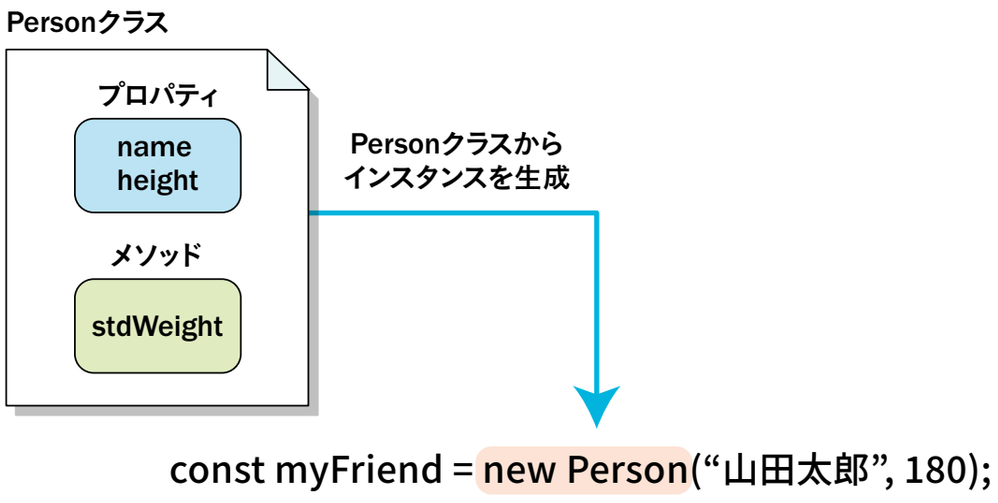
完成ファイル [1001]→[sample1e.html]

予習 クラスについて

ES2015（ES6）以降のJavaScriptでは、JavaやPythonなどと同様に、**「クラス」（class）**という機能を使用してオリジナルのオブジェクトを作成することができるようになりました。

クラスとは、**オブジェクトの設計図**のようなものです。クラスからオブジェクトを生成するには、Dateオブジェクトなど JavaScriptの組み込みオブジェクトと同じように**new演算子**を使用します。

ここでは、シンプルなクラスの例として、名前と身長をそれぞれ「name」「height」という名前のプロパティで管理する、Personクラスを作成してみましょう。メソッドとしては、標準体重を求めるstdWeightメソッドを用意します。



体験 Personクラスを定義する

1 Personクラスを定義する

エディターで「1001」フォルダーの「template1.html」を開いて「sample1.html」といった名前で保存しておきます。あらかじめ用意されている空のscriptエレメントに、Personクラスを定義します❶

```
8  <body>
9      <script>
10         class Person {
11             constructor(name, height) {
12                 this.name = name;
13                 this.height = height;
14             }
15         }
16     </script>
17 </body>
```

```
<script>
    class Person {
        constructor(name, height) {
            this.name = name;
            this.height = height;
        }
    }
</script>
```

❶ 入力する

2 Personクラスのインスタンスを生成する

scriptエレメントに、要素数が3の配列friendsを用意します❶。Personオブジェクトのインスタンスを3つ生成し、配列の要素に順に格納します❷。

```
8  <body>
9      <script>
10         class Person {
11             constructor(name, height) {
12                 this.name = name;
13                 this.height = height;
14             }
15         }
16         const friends = new Array(3);
17         friends[0] = new Person("山田太郎", 160);
18         friends[1] = new Person("田中花子", 165);
19         friends[2] = new Person("猫山一郎", 180);
20     </script>
21 </body>
```

Tips

Personコンストラクターの最初の引数には名前を、2番目の引数には身長を入力します。

```
<script>
    class Person {
        constructor(name, height) {
            this.name = name;
            this.height = height;
        }
    }
    const friends = new Array(3);
    friends[0] = new Person("山田太郎", 160);
    friends[1] = new Person("田中花子", 165);
    friends[2] = new Person("猫山一郎", 180);
</script>
```

❶ 入力する

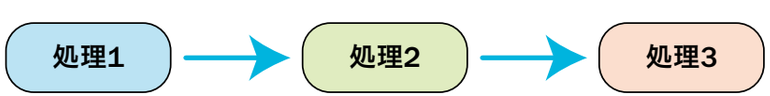
❷ 入力する

1 非同期処理とは — Promiseの基本

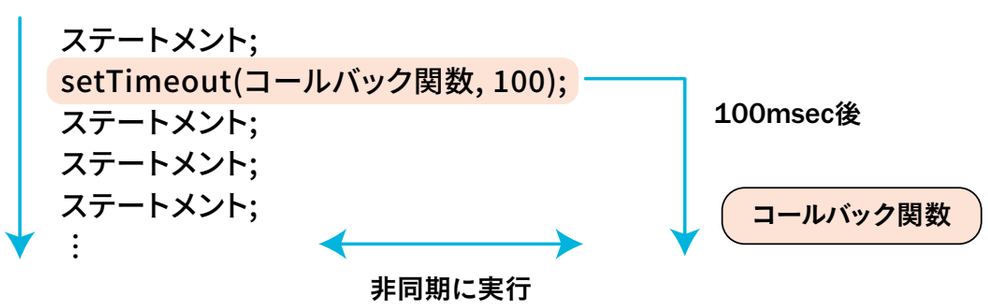
完成ファイル [1101]→[sample1e.html]

予習 非同期処理について

プログラムにおける処理の流れは、「**同期処理**」と「**非同期処理**」に大別できます。同期処理はプログラムの処理を1つずつ順に実行していく方式です、現在のステートメントの処理が完了するまで次のステートメントには進めません。



それに対して非同期処理は、**現在の処理の完了を待たずに次の処理に進む**方式です。JavaScriptでもっともシンプルな非同期処理の機能が「**タイマー**」です（9-2参照）。タイマーを使用して指定した時間後に関数を実行するメソッドにsetTimeoutがありますが、setTimeoutメソッドを実行した時点では何もせずに次のステートメントに処理が進み、**指定した時間後にコールバック関数の処理が実行**されます。



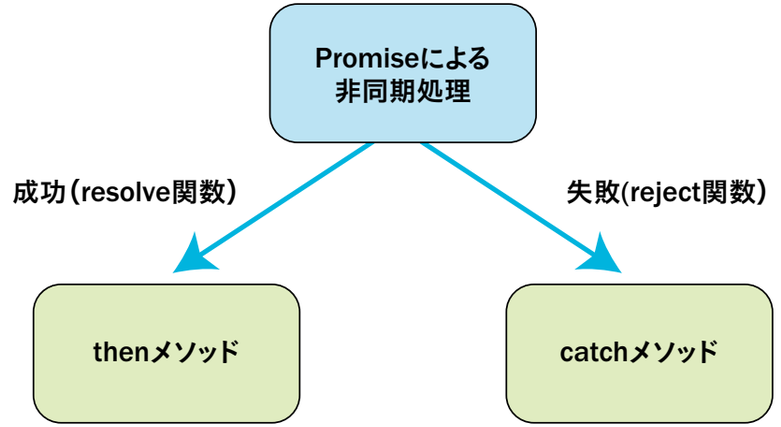
プログラムにおける処理の流れの単位を**スレッド**と呼びますが、JavaScriptは基本的に同時に1つの処理しか行えない**シングルスレッド**の言語です。そのため、実際はメインの処理とコールバック関数が同時に実行されているわけではありません。上記のコールバック関数が処理を行っている間、メインの処理は待ち状態になっています。

非同期処理を扱いやすくするPromise

非同期処理は、次節で説明するJavaScriptとWebサーバーとの間でデータのやりとりを行う**非同期通信の処理を行う際**によく利用されます。

非同期処理の完了を通知する手法の1つに、**コールバック関数**があります。処理が完了した時に実行する関数をコールバック関数として登録して、処理完了後にそれを呼び出して後処理を行うというものです。ただし、コールバックによる処理を多用すると、プログラムのネスト（階層）が深くなり、**コールバック地獄**（313ページのコラム「コールバック地獄」参照）などと呼ばれる問題が発生してしまいます。

そこで、ES2015（ES6）以降のJavaScriptでは、非同期処理を扱いやすくするために**Promise**という機能が用意されました。Promiseを使用すると、非同期処理が成功した場合は**thenメソッド**を、失敗した場合は**catchメソッド**を呼び出すことで、後処理を簡潔に記述できます。



Webサーバーと非同期通信でやりとりする方法については次の節で説明することにして、この節ではタイマーを使用した非同期処理のサンプルを通して、Promiseの基本的な使い方を説明します。