東京ITスクール
TOKYO IT SCHOOL

JavaScript

目次

1. JavaScript とは

1 概要

JavaScript は、Web ページを動的に制御するためのオブジェクト指向型のプログラミング言語です。

1995 年に登場以来、様々な Web ページで利用されています。当初、JavaScript は手軽に動的な Web ページを作成できることから人気を博しましたが、Web ブラウザ間の互換性問題やセキュリティ問題などにより、次第に人気が伸び悩むようになりました。しかし、近年 Google などの各種サービスで JavaScript が利用されたことから再び脚光を浴び、現在の Web アプリケーション開発において欠かせない技術となっています。

2 動作環境

JavaScript は、Internet Explorer、Chrome、Firefox、Safari など、様々な Web ブラウザ上で動作します。JavaScript を使用することで、たとえば「リンク上にマウスポインタを移動させると画像が変化する」などの動的な Web ページが作成できます。

JavaScript は、初心者でもとっつきやすいプログラミング言語ですが、様々なことができる本格的なプログラミング言語でもあります。主に Web ブラウザで利用されていますが、その用途は Web ブラウザだけではありません。最近では、Windows 上で動作するガジェットや Adobe AIR でのデスクトップアプリケーションを開発する際に使用するなど、様々なところで利用されています。

JavaScript と Java の違い

JavaScript と似たプログラミング言語で「Java」があります。JavaScript と Java は、基本文法が類似した高級言語である、オブジェクト指向型であるなど、共通点がいくつかありますが、実際は全くの別物です。

これらの言語の大きな違いは動作環境です。Web アプリケーションの動作環境において、Java は「サーバ側」で動作しますが、JavaScript は Web ブラウザ、つまり「クライアント側」で動作します。この違いに気を付けてください。

3 開発環境

JavaScript のプログラムは、Web ブラウザとテキストエディタがあれば、作成、実行することができます。しかし、メモ帳だと機能が少なく不便であるため、多機能なテキストエディタや統合開発環境を使用して実装することが一般的です。

なお、本研修では以下のアプリケーションを使用できます。

- サクラエディタ
- Eclipse
  ※静的 Web プロジェクト、または動的 Web プロジェクトの「WebContent」フォルダ以下にソースコードを記述したファイルを保存してください。

2. 基本構文

1　記述方法

JavaScript のソースコードを記述する方法は以下の 3 通りがあります。

1. **HTML 文書中の<script>タグ内に記述する。**
2. **HTML 文書中のイベントハンドラに記述する。**
3. **外部ファイル（js ファイル）に記述する。**

HTML 文書中の<script>タグ内に記述する方法

HTML 文書内にソースコードを記述するには、まず<script>タグを記述します。
<script>タグとは、HTML 文書内に別規格のプログラムなどを記述する際に使用するタグです。<script>タグの開始タグと終了タグの間にソースコードを記述できます。

```
<script type=”text/javascript”>
  ソースコード
</script>
```

<script>タグ内には他のプログラミング言語も記述できます。そのため、どのプログラミング言語を利用するか区別するために、type 属性を設定します。JavaScript の場合、type 属性の値に「text/javascript」と記述します。

sample02_01.html（下線部は JavaScript のソースコード）

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>サンプル</title>
  </head>
  <body>
    <script>
      document.write('<h1>');
      document.write('Hello world!');
      document.write('</h1>');
    </script>
  </body>
</html>
```

上記のサンプルコード中の「document.write(……)」という文は、()内の文字列を Web ブラウザ上に出力します。このサンプルコードでは、シングルクォ―テーション(’)で囲まれた文字列を出力しています。つまり、下記のような HTML 文書と同じ出力結果となります。

JavaScript 未使用の場合

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>サンプル</title>
  </head>
  <body>
    <h1>Hello world!</h1>
  </body>
</html>
```

【補足】type 属性の省略

HTML のバージョンが 5 の場合、<script>タグの type 属性は省略できます。ただし、使用するプログラミング言語が JavaScript の場合のみ省略可能であるため、他のプログラミング言語を使用したい場合は type 属性の記述は必須です。

また、HTML4.01 の場合、type 属性の記述は必須です。

【補足】複数の<script>タグ

<script>タグは、1 つの HTML 文書内に複数記述でき、記述された順に実行されます。

後ろの<script>タグに記述された処理は、前に記述された<scrpt>タグ内の情報を参照できますが、逆の場合は参照できません。そのため、最初に実行したい処理は<head>～</head>タグ間に記述することが一般的です。

HTML 文書中のイベントハンドラに記述する方法

JavaScript の処理は、「ボタンがクリックされた」、「テキストボックスが選択された」など、何らかのイベントが発生したタイミングで実行させられます。イベント発生時に処理を実行する特殊なコードのことを「イベントハンドラ」と呼びます。

イベントハンドラは、HTML タグに属性の形式で記述します。そして、イベントハンドラの値には、実行させたい JavaScript のソースコードを記述します。

```
<要素名 イベントハンドラ名 = "JavaScript のソースコード">〇〇</要素名>
```

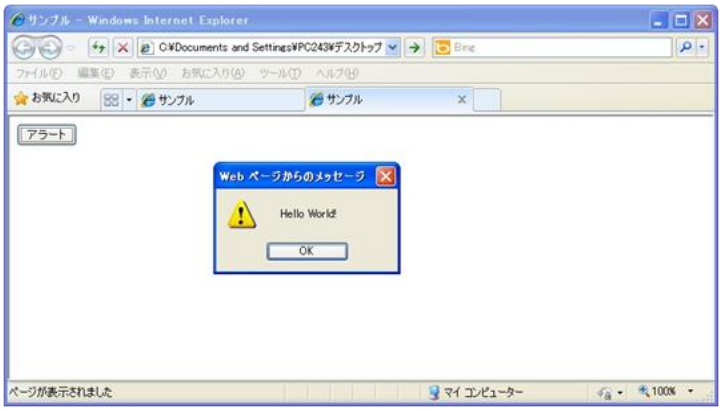
イベントハンドラは多数用意されています（詳細は別途紹介）。

たとえば、フォーム内のボタンなどをクリックしたときに発生するイベントに対応するイベントハンドラ名は「onclick」です。下記のサンプルコードでは、ボタンをクリックしたときに警告ダイアログを表示する処理が実行されます。

sample02_02.html（下線部はイベントハンドラ）

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>サンプル</title>
  </head>
  <body>
    <form>
      <input type="button" value="アラート" onclick="alert('Hello World!')">
    </form>
  </body>
</html>
```

実行結果



外部ファイル（js ファイル）に記述する

JavaScript のソースコードは、HTML ファイルとは別のファイル（外部ファイル）に記述することもできます。
JavaScript のソースコードを外部ファイル化することは、以下のようなメリットがあります。

- **HTML と JavaScript のソースコードを分離できるため、可読性が上がる。**
- **1 つの JavaScript のソースコードを複数の HTML ファイルが利用できるようになるため、保守性が上がる。**

そのため、JavaScript のソースコードは外部ファイル化することが望ましいです。
なお、外部ファイルを使用する場合は、以下の手順で実装します。

1. 外部ファイルを作成する。
2. HTML ファイルに、外部ファイルを読み込む処理を記述する。

外部ファイルの作成

外部ファイルには、拡張子「.js」を付け、ファイル内に実行させたい処理を直書きします

sample02_03.js

```
document.write('<hr>');
document.write('Hello world!');
document.write('</hr>');
```

HTML ファイルへの外部ファイルを読み込む処理の追記

HTML 文書中で外部ファイルを読み込ませたい位置に、<script>タグを記述します。そして、<script>タグの src 属性に外部ファイルの URL を記述します。

<script>タグの記述例

```
<script type="text/javascript" src="ファイルの URL" charset="文字コード"></script>
```

sample02_03.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>サンプル</title>
<script src="../js/sample02_03.js"></script>
</head>
<body>
</body>
</html>
```

【補足】文字コードの指定

HTML ファイルと外部ファイルの文字コードが異なる場合、外部ファイルの文字コードと同じものを<meta>タグの charset 属性に指定します。

sample02_03.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>サンプル</title>
<script src="../js/sample02_03.js"></script>
</head>
<body>
</body>
</html>
```

sample02_03.js

```
document.write('<h1>');
document.write('Hello world!');
document.write('</h1>');
```

【補足】JavaScript が無効化された場合の対策

Web ブラウザの標準設定では、JavaScript は実行可能となっていますが、設定を変更することで実行を無効化できます。

この場合、JavaScript による処理が実行される場所では何も処理されないため、「何も表示(出力)されない」といった状態となります。このような状況に備え、代替メッセージなどを用意しておくことがユーザへの配慮として望ましい。

代替メッセージなど表示する際には<noscript>タグを使用します。<script>タグの後ろに、次のように<noscript>～</noscript>タグを記述します。そして、タグ内に JavaScript が利用できない場合に出力させる HTML 文書を記述しておきます。

JavaScript を無効にした場合の出力例

```
<noscript>
  この Web サイトの全ての機能を利用するためには JavaScript を有効にする必要があります。
  <a href="http://www.enable-javascript.com/ja/">
    あなたの Web ブラウザで JavaScript を有効にする方法
  </a>
  を参照してください。
</noscript>
```

2　記述ルール

JavaScript のソースコードを記述する際には、以下の記述ルールを守りましょう。

空白スペース

JavaScript では、字句の間の空白スペース、タブ、改行コードを無視します。たとえば、次のような記述も可能です。

```
document.write(
  'Hello world');
```

セミコロン

JavaScript では文の終わりにセミコロン(;)を記述します。また、次の例のように 1 行以上を波括弧({　})で囲む場合もあります。波括弧で囲まれた部分は「文ブロック」と呼び、複数文をまとめて 1 つの文として扱います。なお、この文ブロックの終わりにはセミコロンを記述しません。

```
for (var i = 0 ;  i < 10; i++) {
  document.write('Hello world!');
  document.write('<br>');
}
```

大文字と小文字

JavaScript では、大文字と小文字が区別されます。
たとえば、前述の文字列「Hello world!」を出力する JavaScript を組み込んだ HTML にある「document.write('Hello world!');」の先頭文字「d」を大文字で記述すると実行時にエラーになります。

【補足】エラー情報の確認方法

JavaScript 実行時のエラー情報の表示方法は、Web ブラウザによって異なります。主要 Web ブラウザのエラー表示の表示方法は以下の通りです。

Internet Explorer の場合

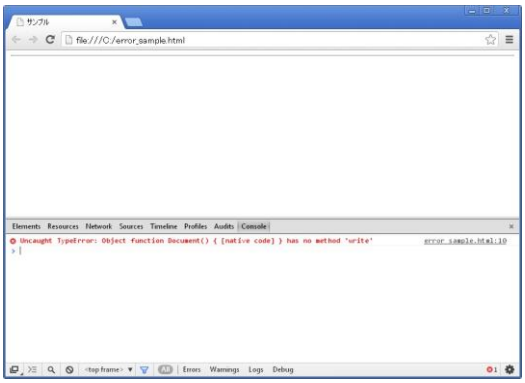
Internet Explorer では、F12 開発者ツールのコンソールにエラー情報が表示されます。

なお、JavaScript 実行時には自動的にエラー情報を表示させる設定があります。表示させたい場合は次の手順で設定します。

1. メニューバーから［ツール］→［インターネットオプション］を選択します。
2. 「インターネットオプション」ダイアログが表示されるので、「詳細設定」タブを選択します。
3. 「詳細設定」タブ内で「設定」の「ブラウズ」→「スクリプトエラーごとに通知を表示する」項目を変更することで、JavaScript 実行時のエラー情報表示のオン／オフを設定します。

Chrome の場合

Chrome では、JavaScript コンソールを表示させることで、エラー情報を表示できます。メニューバーの設定ボタンから［ツール］→［JavaScript コンソール］を選択すると、コンソールが表示されます。なお、コンソールの表示には、Ctrl＋Shift＋J キーがショートカットキーとして割り当てられています。コンソールが表示された状態で JavaScript が実行すると、エラー情報が表示されます。



また、エラーコンソールで通知されたエラー箇所をクリックすると、該当箇所のコードが表示されます。

3　コメント

「コメント」とは、プログラムの内容を自分自身や他のプログラマに対して伝えるために、プログラム内に記述する注釈のことです。プログラム実行時、コメントはすべて無視され、実行されません。

JavaScript のコメントの記述方法は以下の 2 種類があります。

単一行コメント

「//」を記述すると、その記号の後ろの文字はコメントとして解釈されます。

記述例

```
//コメント行です。
var price = 100; //  ここもコメント行です。
```

複数行コメント

「/*」と「*/」で囲まれた文字はコメント行として解釈されます。

記述例

```
/*  コメント行です*/
/*  複数行に渡って
コメントを記述できます。*/
```

4　予約語

「予約語」とは、JavaScript の文法であらかじめ定義されている特別なキーワードのことです。

代表的な JavaScript の予約語

break	case	catch	countinue	do
else	debugger	finally	for	function
if	new	with	return	switch
while	try	typeof	this	void

■JavaScriptとは
・基本的にWebブラウザ等のクライアント側で動作するクライアントサイドプログラム。
※Javaはサーバーサイドプログラミング言語。
・JavaScriptもjavaと同様、オブジェクト指向型言語。

■JSを書ける場所
・htmlタグ中のイベントハンドラの値
・拡張子.jsの外部ファイル
・scriptタグ内

3. 変数

1　利用方法

JavaScript で変数を利用する場合、以下の手順で処理を実装します。

1. 変数を宣言する。
2. 変数に値を代入する。

変数を宣言する場合、次のように「var」キーワードを記述します。

```
var　変数名；
```

宣言した変数に値を代入する際には、代入演算子「=」を使用します。JavaScript では、「=」は「等しい」ではなく、「代入」を意味します。

```
変数名 = 値;
```

変数宣言時に値を代入することもできます。

```
var　変数名　=　値；
```

次のソースコードでは、変数「age」が宣言され、その変数に 20 が代入されます。

```
var　age　=　20；
```

また、カンマ（,）を利用することで、複数の変数をまとめて宣言できます。

```
var　name, age；
```

【補足】var キーワード以外の宣言方法

このテキストでは以降の章でも「var」キーワードを使用しますが、その他に「let」キーワード、「const」キーワードでも同様の宣言が可能です。それぞれ「再度同じ変数名で宣言出来るか（再宣言）」、「再度同じ変数名に対して代入できるか（再代入）」「スコープは何か」の制約が発生するキーワードになっています。これらキーワードが存在する理由は、意図しない再宣言や再代入、変数の参照によるバグを未然に防ぐためです。それぞれのキーワードに対する制約は以下の表の通りです。

キーワード	再宣言	再代入	スコープ
var	○	○	グローバルスコープ・関数スコープ
let	×	○	ブロックスコープ
const	×	×	ブロックスコープ

2　命名規則

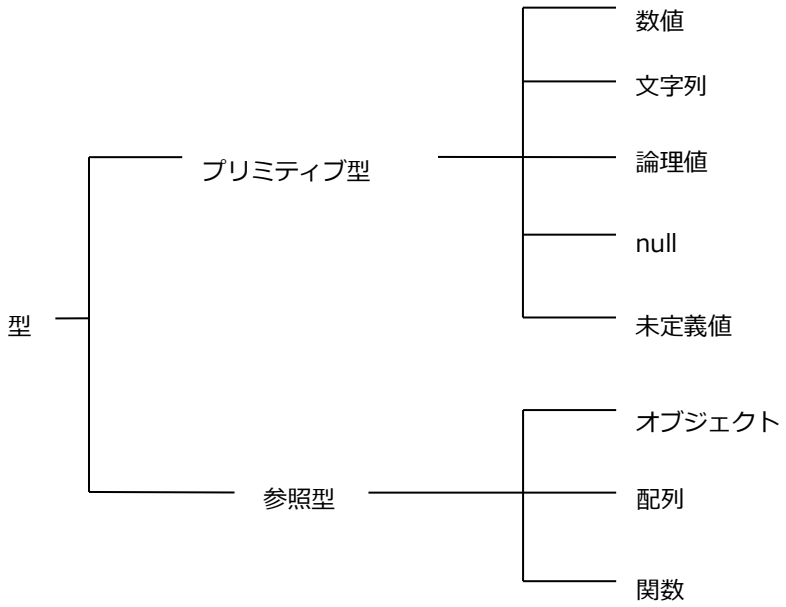
変数や関数などに対してユーザが定義する名称を「識別子」と呼びます。識別子は以下の命名規則に従う必要があります。

- **使用できる文字は、英字、アンダースコア(_)、ドル記号($)のみ。**
- **数字は先頭文字としては利用できない。**
- **予約語は利用できない。ただし、予約語を変数名の一部に含めることはできる。**

たとえば、「1st」や「2nd」は 1 文字目に数字が利用されているため、識別子として利用できません。
また、予約語「for」は識別子として利用できませんが、「force」は利用できます。

4. 型

JavaScript で利用可能な型はデータ値の格納方法の違いにより、大きく「プリミティブ型」と「参照型」の 2 つに分けられます。



プリミティブ型には「数値」「文字列」「論理値」に加え、特殊な型である「null」「未定義値」が含まれます。

参照型に含まれる「オブジェクト」とは、複数のプリミティブ型や参照型のデータの集合体です。また、特殊なオブジェクトとして、データにインデックスを割り当ててまとめて取り扱う「配列」や特定の処理を取りまとめた「関数」があります。

以上の通り、JavaScript には様々な型がありますが、JavaScript は型をあまり意識しない言語仕様となっています。たとえば、数値を格納した変数に対して文字列を代入できます。逆に、文字列を格納した変数に対して数値を格納できます。ただし、まったく型を意識しなくてもよいというわけではありません。ここではまず JavaScript で扱う型について説明し、その後「プリミティブ型」と「参照型」の違いを説明します。

1　プリミティブ型

数値

任意の数値データを取り扱う型です。JavaScript では、整数と浮動小数点数は区別されず、すべての数値を浮動小数点数で取り扱います。

ただし、表記上は整数と浮動小数点数を書き分けることができます。

文字列

任意の文字列を取り扱う型です。文字列はシングルクォーテーション(')、またはダブルクォーテーション(")で囲んで定義できます。

文字列の例を次に示します。

```
'Shoeisya'
"JavaScript"
'100'
""
"I´m a developer."
```

4 行目の例のように、文字列には 0 文字で構成される文字（空文字）も含まれます。

【補足】エスケープシーケンス

「 I´m a developer.　」のように、文字列中に「'」を含んでいる場合、「'」で文字列を囲もうとすると「 'I´m a developer.' 」となり、正しく文字列として扱われません。この場合、上記の例のように「"」で囲むか、エスケープシーケンスを利用して対処します。

代表的なエスケープシーケンス

エスケープシーケンス	説明
¥0	NULL 文字
¥b	バックスペース
¥t	タブ
¥n	改行(LF : Line Feed)
¥v	垂直タブ
¥f	フォームフィード
¥r	復帰(CR : Carriage Return)
¥“	ダブルクォーテーション「 “ 」
¥‘	シングルクォーテーション「 ‘ 」
¥¥	バックスラッシュ「¥」

【補足】シングルクォーテーションとダブルクォーテーションの使い分け

JavaScript のソースコードを HTML 文書中に記述する場合、シングルクォーテーションとダブルクォーテーションの囲み方によっては、プログラムが動作しなくなってしまいます。

そのため、たとえば「HTML 部分はダブルクォーテーションを利用する」、「JavaScript 部分はシングルクォーテーションを利用する」のように使い分けましょう。

論理値

「論理値」とは、真(true)、または偽(false)の 2 種類の値のみを取り扱う型です。何かが真であることを true、何かが偽であることを false で定義します。論理値は制御文の条件式で 2 つの値を比較する際によく利用されます。たとえば、次の式は変数 x が 0 であるかどうかを評価する式です。

```
x == 0
```

変数 x が 0 に等しい場合、論理値は true となり、等しくない場合は false となります。

null

null は値がないことを示す特殊な値で、「null」という表記のみ利用できます。オブジェクト型においてオブジェクトがないことを表す特殊な値として扱われます。

未定義値

「未定義値」とは、変数宣言しただけで値が代入されていないことを示す特殊な値です。値が代入されていない変数や存在しないオブジェクトのフィールドを利用すると、未定義値として「undefined」が返却されます。

未定義値を出力した例

```
var　x ;
alert(x) ;
```

実行結果

2　参照型

オブジェクト

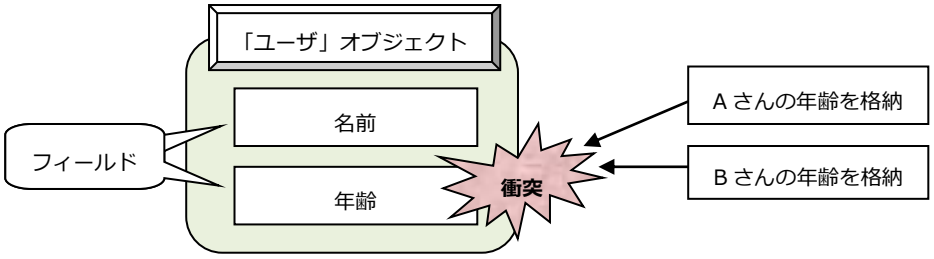
順序のない様々なデータの集合を「オブジェクト」と呼びます。

オブジェクトは、オブジェクト自身の状態や特性を表す値「フィールド（プロパティ）」と、オブジェクトに関連付いた処理の集まり「メソッド」を持っており、それらを利用して様々な機能を提供します。

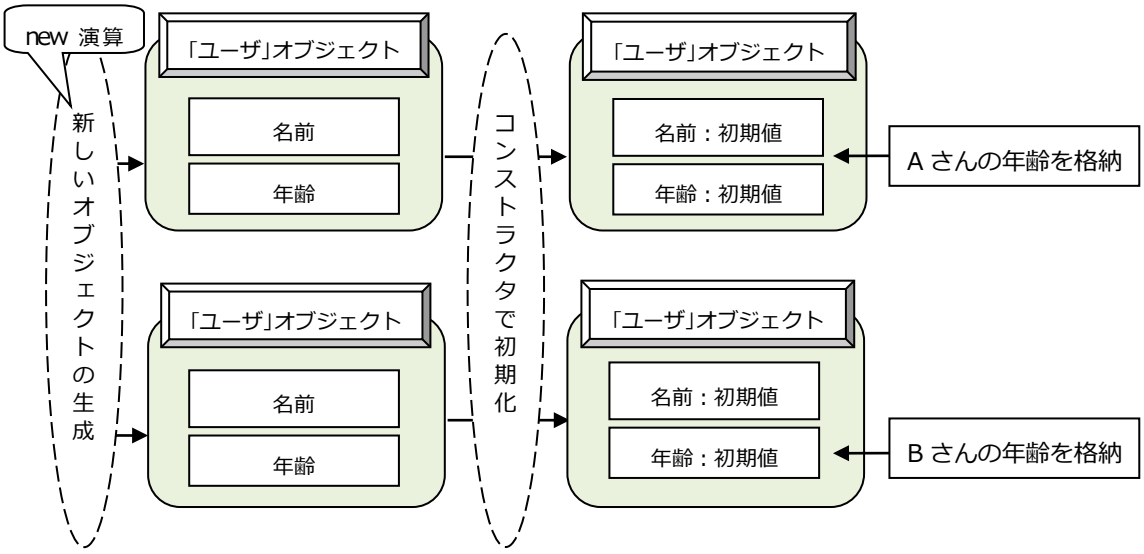
オブジェクトの生成

オブジェクトの基本的な利用方法を紹介します。

たとえば、名前、年齢などのフィールドに持つ「ユーザ」という名前のオブジェクトが定義されていたとします。この 1 つのオブジェクトに、A さんと B さん 2 人分のユーザ情報をまとめて格納して扱うことはできません。



そのため、オブジェクトを利用する場合、まず new 演算子を利用してフィールドが定義されていない新しいオブジェクトを生成します。そして、生成したオブジェクトに対して、「コンストラクタ」と呼ばれるオブジェクトを初期化する処理を実行し、フィールドに値を代入します。このように、新しいオブジェクトを作成し、保存領域を確保してからオブジェクトを定義することで、データを独立させて扱えます。



オブジェクトは、new 演算子と該当するコンストラクタを利用することで生成されます。

```
var 変数名 = new コンストラクタ名(引数, 引数, …)
```

なお、オブジェクトの生成時、コンストラクタに引き渡すデータを「引数」と呼びます。渡すべきデータがない場合、引数は省略します。

オブジェクトの利用

オブジェクトのフィールドやメソッドを利用する場合はドット（.）を利用します。また、メソッドを呼び出す際、渡すべきデータがない場合は、引数を省略します。

```
変数名.フィールド名
変数名.メソッド名(引数,　引数,　…)
```

sample04_01.js

```
//コンストラクタの定義
function user(name, age, gender) {
    this.name = name;
    this.age = age;
    this.gender = gender;
    this.showDetail = showDetail;
}

//メソッドの定義
function showDetail() {
    var detail = this.name + "¥n" + this.age + " years old¥n" + this.gender;
    alert(detail)
}
```

実行結果（sample04_01.html を実行）

配列

順序付けられたデータの集合を扱うオブジェクトを「配列」と呼びます。
データをまとめて取り扱う点ではオブジェクトと同様ですが、配列では各データに0、1、2と「インデックス（添字）」と呼ばれる連番が割り当てられ、順序付けされた状態でデータが扱われます。
また、配列内の各データのことを「配列要素（以下、要素）」と呼びます。

配列の生成

配列の生成にはnew演算子とArrayコンストラクタを使用します。

```
var 変数名 = new Array(引数, 引数, …) ;
```

オブジェクトの生成時にコンストラクタに引き渡すデータを「引数」を呼びます。渡すべきデータがない場合は省略します。
なお、new演算子を使用した配列の生成では、引数の型と数によってその引数の意味が変わります。

数値1個の場合

引数は配列の長さ（要素数）をあらわす。
ただし、数値が浮動小数点数の場合、エラーとなる。

数値以外の値1個、または複数個の場合

引数は要素の値をあらわす。

また、配列は下記のように記述することもできます。

```
var 変数名 = [要素, 要素, 要素, …] ;
```

各要素には型の制約がないため、異なる型の値を同じ配列に格納できます。また、下記のように、要素に値を格納しない場合、その要素の値は未定義値（undefined）となります。

```
var 変数名 = [1, , , 4]
```

配列の利用

配列内のデータを利用する場合、次のように変数名の後に角括弧「[ ]」を利用して要素の「インデックス番号」を指定します。

```
変数名[インデックス番号]
```

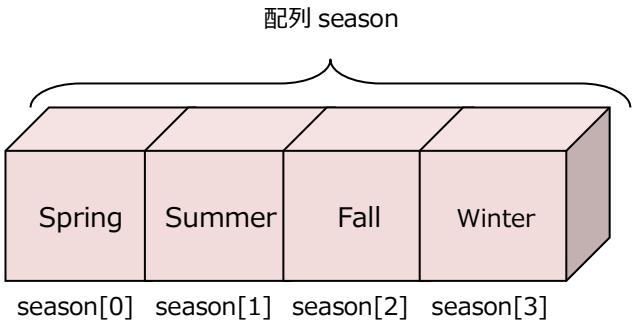
たとえば、下記の処理では、季節を表した 4 種類の文字列を配列に格納して、インデックスが「1」(2 番目)の値「Summer」を取得し、出力します。

sample04_02.js

```
var season = ['Spring', 'Summer', 'Fall' ,'Winter'];
alert(season[1]);
```

実行結果（sample04_02.html を実行）

配列のイメージ



【補足】配列の長さ

配列の長さを求めるには、配列が持つ「length」という情報を利用します。
配列を参照している変数に続けて「.length」と記述すると、その処理結果として配列の長さを取得できます。

記述例：配列の長さを求める処理

```
var names = new Array('太郎', '二郎', '花子') ;
var count = names.length;
```

関数

複数の処理をまとめたものを「関数」と呼びます。JavaScript では、関数を「function」キーワードを使って定義します。

```
function　関数名(引数, 引数, …) {
  複数の文
  return　戻り値 ;
}
```

関数に渡す値を「引数」と呼びます。引数に渡すべきが値がない場合は省略します。また、関数の呼び出しによって何らかの処理結果を呼び出し元に返却する場合、「return」の後にその値（戻り値）を記述します。

関数の定義例として、摂氏（C）から華氏（F）への変換処理を関数にしたものを以下に示します。なお、摂氏から華氏への変換式は「F = 9 / 5 × C + 32」です。

sample04_03.js

```
function convertCtoF(c) {
  var f = 9 / 5 * c + 32;
  return f;
}

alert('摂氏 20 度は、華氏' + convertCtoF(20) + '度に該当します。');
alert('摂氏 30 度は、華氏' + convertCtoF(30) + '度に該当します。');
```

実行結果（sample04_03.html を実行）

convertCtoF()の後ろでは、convertCtoF()を呼び出し、摂氏 20 度と 30 度を華氏に変換した結果を順番にアラート画面に表示しています。

5. 演算子

1　算術演算子

算術演算子は、オペランドの数値で算術演算を行う演算子です。

JavaScript で利用可能な算術演算子

演算子	説明	利用例	処理内容
+	加算の二項演算子	x = y + 2;	変数 y と 2 を加算した結果を変数 x に代入する
−	減算の二項演算子	x = y − 1;	変数 y から 1 を減算した結果を変数 x に代入する
*	乗算の二項演算子	x = y * 10;	変数 y と 10 を乗算した結果を変数 x に代入する
/	除算の二項演算子	x = y / 2;	変数 y を 2 で除算した結果を変数 x に代入する
%	剰余算の二項演算子	x = y % 10;	変数 y を 10 で除算した余りを変数 x に代入する。
+(単項)	単項演算子	x = +y;	変数 y の値をそのまま変数 x に代入する
−(単項)	符号反転の単項演算子	x = −y;	変数 y の値を符号反転して変数 x に代入
++	インクリメント演算子	x++	変数 x の 1 値を 1 増やす
−−	デクリメント演算子	x−−	変数の値を 1 減らす

2　文字列結合演算子

オペランドに文字列やオブジェクトの含まれる場合、「+」演算子は文字列結合演算子として扱われます。文字列結合演算子はオペランドを文字列として処理するため、オペランドが文字列以外の場合、文字列に変換されてから各文字列が連結されます。

sample05_01.js

```
var result1 = 'Hello' + 'World';
var result2 = '1' + '2';
var result3 = '1' + 2;
var result4 = 1 + 2;
alert(result1 + '¥n' + result2 + '¥n' + result3 + '¥n' + result4);
```

実行結果（sample05_01.html を実行）

3　比較演算子

比較演算子は、2 つのオペランドの値を比較し、その比較結果に応じて論理値 true もしくは false を返却する演算子です。

JavaScript で利用可能な比較演算子

演算子	説明	利用例	処理内容
==	値が等しいかを評価する二項演算子	x == 10	x の値が 10 と等しい場合 true、等しくない場合 false を返す
===	同一であるか(値と型が等しいか)を評価する二項演算子	x === 10	x の値と型が数値の 10 と等しい場合 true、等しくない場合 false を返す
!=	値が等しくないかを評価する二項演算子	x != 10	x の値が 10 と等しくない場合 true、等しい場合 false を返す
!==	同一でないか(値と型が等しくないか)を評価する二項演算子	x !== 10	x の値と型が数値の 10 と等しくない場合 true、等しい場合 false を返す
<	左辺 < 右辺であるかを評価する二項演算子	x < 10	x の値が 10 未満の場合 true、10 以上の場合 false を返す
>	左辺 > 右辺であるかを評価する二項演算子	x > 10	x の値が 10 超の場合 true、10 以下の場合 false を返す
<=	左辺 <= 右辺であるかを評価する二項演算子	x <= 10	x の値が 10 以下の場合 true、10 超の場合 false を返す
>=	左辺 >= 右辺であるかを評価する二項演算子	x >= 10	x の値が 10 以上の場合 true、10 未満の場合 false を返す

【補足】厳密等価演算子

等価演算子「==」と厳密等価演算子「===」は、ともにオペランドに任意の型を指定でき、2 つのオペランドの値が等しいかどうかを比較する演算子ですが、「等しい」と評価する基準が異なります。

===演算子は型を含めて等しいかどうかを評価します。==演算子はオペランドに異なる型が指定された場合、オペランドは次のように比較されます。

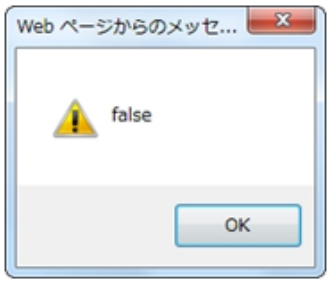
- オペランドの組み合わせが数値と文字列の場合、文字列を数値に変換して比較する。
- オペランドの組み合わせに論理値が含まれる場合、数値に変換して比較する。
- オペランドの組み合わせがオブジェクトと数値または文字列の場合、オブジェクトを valueOf()メソッドなどを利用してプリミティブ型の数値に変換してから、比較する。
- オペランドの組み合わせが null と undefined の場合、等しい値として判定される。

また、等価演算子「==」と厳密等価演算子「===」では、オペランドに数値や文字列などのプリミティブ型が指定された場合は値そのものを比較しますが、オブジェクトや配列などの参照型が指定された場合は参照先で比較します。たとえば、以下のようなサンプルコードの場合、配列 a と配列 b は参照先がそれぞれ別となるため、等しくないと判断されます。

参照型を比較した例

```
var a = ['1', '2', '3'];
var b = ['1', '2', '3'];
alert(a == b);
```

実行結果



4　論理演算子

論理演算子は、理論値のオペランドに対して、否定(NOT)、論理積(AND)、論理和(OR)などの論理演算をする演算子です。JavaScript で利用可能な論理演算子を示します。

演算子	説明	利用例	処理内容
!	否定(NOT)を演算する単項演算子	!x	x の値が false の場合 true、true の場合 false を返す
&& または &	論理積(AND)を演算する二項演算子	x>0 && y>0	「x>0」かつ「y>0」の場合 true、それ以外の場合 false を返す
|| または |	論理和(OR)を演算する二項演算子	x>0 || y>0	「x>0」または「y>0」の場合 true、それ以外の場合 false を返す

論理演算式のうち、「&&」と「&」、「||」と「|」は、いずれも左側のオペランド(式)から順番に評価します。
では、「&&」と「&」、「||」と「|」の違いは何でしょうか。
「&」と「|」は左右両方のオペランド(式)を必ず評価します。それに対して、「&&」と「||」は、まず左側のオペランド(式)を評価して、次に必要な場合のみ右側のオペランド(式)を評価します。
たとえば、下記の式において、左側のオペランド(式)「x > 0」の評価が false の場合、右側のオペランド(式)「y > 0」は評価されません。

```
var result = (x > 0) && (y > 0);
```

これは、「&&」の左側の式が false の場合、右側の式の評価結果にかかわらず、式全体の結果が false となるためです。
「&&」と「&」、「||」と「|」では、このように処理の仕方に違いがあります。

5　代入演算子

代入演算子は、変数にデータ値を設定する演算子で「=」を利用します。また、代入演算子である「=」以外に、代入演算と算術演算やビット演算を組み合わせた「複合代入演算子」があります。たとえば、変数 x に対して「100」を加算したい場合、次のような記述になります。

```
x = x + 100 ;
```

上記のコードは演算子「+=」を利用して、次のような記述も可能です。

```
x += 100;
```

6　その他の演算子

JavaScript では、ここまでに紹介してきたもの以外にも次の演算子をサポートしています。

JavaScript で利用可能な演算子

演算子	説明
delete	オブジェクトのフィールドなどを削除する単項演算子
in	オブジェクトにフィールドが含まれているかを確認するための二項演算子
instanceof	オブジェクトの種類を確認するための二項演算子
new	新しいオブジェクトを生成する単項演算子
typeof	型を調べるための単項演算子
void	未定義値を返却する単項演算子
カンマ(,)	左側、右側の順にオペランドを評価し、右側の式の結果を返却する二項演算子

6. 条件分岐

1　if 文

if 文は条件分岐の処理に利用します。はじめに、if 文の構文を見てみましょう。

```
if　(条件式) { 文 }
```

if 文では「条件式」を評価し、その結果が true の場合、「文」を実行します。結果が false の場合、「文」は実行されず if 文の次の文が実行されます。「条件式」には比較演算子がよく利用されます。なお、条件式の評価結果が true もしくは false でない場合は、論理値に変換されて評価されます。

if 文の利用例として、変数 age が 20 以上の場合、if 文の内容が実行されアラートダイアログに「成人」と出力するプログラムを以下に示します。

sample06_01.js

```
var age = 30;
if (age >= 20) {
  alert(age + '歳は成人です。');
}
```

実行結果（sample06_01.html を実行）

2　if～else 文

前節のサンプルコードでは、変数 age を 20 未満の値に修正して実行した場合、if 文の内容が実行されないため、何も起きません。if 文での条件式の評価結果が true の場合だけでなく、false になった場合にも何らかの処理を実行したい場合があります。この場合、if 文に else 文を追加した構文（if～else 文）を使用します。

```
if (条件式) {
  文 1
} else {
  文 2
}
```

sample06_02.js

```
var age = 10;
if (age >= 20) {
  alert(age + '歳は成人です。');
} else {
  alert(age + '歳は未成年です。');
}
```

実行結果（sample06_02.html を実行）

3　if～else if～else 文

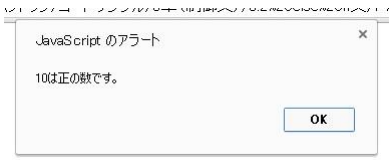
3 つ以上の分岐処理の場合、if～else 文のみではネストする必要があります。このような場合、if～else if～else 文を利用します。

```
if (条件式 1) {
  文 1
} else if (条件式 2) {
  文 2
    ・
    ・
    ・
} else if (条件式 m) {
  文 m
} else {
  文 n
}
```

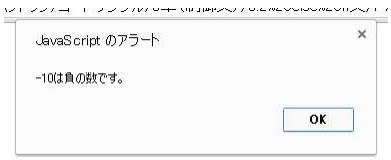
sample06_03.js

```
var num = 10;
if (num > 0) {
  alert(num + 'は正の数です。');
} else if (num < 0) {
  alert(num + 'は負の数です。');
} else {
  alert('0 です。');
}
```

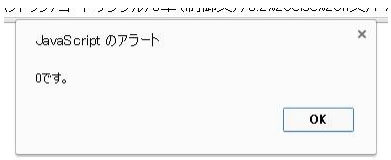
実行結果（sample06_03.html を実行）



変数 num が
10(正の数)の場合



変数 num が
-10(負の数)の場合



変数 num が
0 の場合

7. 反復処理

1　for 文

for 文を利用すると、文を繰り返し実行できます。for 文の書式は次の通りです。

```
for (初期化式; 条件式; 更新式) { 文 }
```

「初期化式」は最初に 1 回実行されます。「初期化式」ではループを制御する変数に初期値を与えます。この変数を「ループ制御変数」と呼びます。「初期化式」が実行されると、ループが開始されます。ループ本体を実行する前に、まず「条件式」が評価されます。「条件式」が true である場合、ループ本体の文を実行します。最後に「更新式」を実行します。「更新式」はループ制御変数の値を一定量増減させる式となります。

for 文の利用例として、0～9 までの数値を HTML ドキュメントに出力するプログラムを以下に示します。このプログラムを実行すると、0～9 までの数値が順番に出力されます。

sample07_01.js

```
for (var i = 0; i < 10; i++) {
  document.write(i + '<br/>');
}
```

実行結果（sample07_01.html を実行）

2　while 文

for 文の他に繰り返し処理するための制御文として while 文があります。for 文は繰り返しの回数があらかじめ決まっている場合は便利な制御文です。しかし、繰り返し回数をあらかじめ決められない場合があります。このような場合に有効な制御文が while 文となります。while 文の書式は次の通りです。

```
while (条件式) { 文 }
```

while 文はループ処理の最初に「条件式」を評価します。「条件式」が真の場合、「文」を実行し、「条件式」が偽の場合、次の処理に進みます。

while 文の処理の流れを見ると、処理を繰り返すかどうかの判定を繰り返しの先頭部分で実施します。そのため、while 文は「条件式」が満たされなくなるまで処理を繰り返す文といえます。while 文の利用例として、0～9 までの数字を順番に HTML ドキュメントに出力するプログラムを以下に示します。

sample07_02.js

```
var i = 0;
while (i < 10) {
  document.write(i + '<br/>');
  i++;
}
```

実行結果（sample07_02.html を実行）

0
1
2
3
4
5
6
7
8
9

8. 関数

関数は大きく以下の 2 種類に分けられます。

- **JavaScript に用意されている関数(ビルトイン関数)**
- **プログラム内で新たに定義する関数(ユーザ定義関数)**

ここでは、ユーザ定義関数(自作関数)の定義の仕方について説明します。

1　関数の定義

function 文では、function キーワードの後に関数名を記述し、その後ろに()で囲んで引数を定義します。引数が複数ある場合は、カンマ(,)で区切ります。

そして、{} 内に変数の定義や、別の関数の呼び出し処理などの文を記述できます。

また、戻り値がある場合は「return」キーワードの後に式や値を指定することで、関数の呼び出し元に結果を返せます。

```
function 関数名(引数、引数、……) {
  複数の文
  return 戻り値;
}
```

sample08_01.js

```
// 円錐の体積を求める関数
function calcCone(radius, height) {
  // 底面の面積
  var base = radius * radius * Math.PI;
  // 体積 = 1/3 * 底面積 * 高さ
  return base * height / 3;
}
```

なお、JavaScript では、戻り値の型を指定しません。そのため、関数からどのような型が返却されたかを、呼び出し元は把握している必要があります。不具合の原因となるためあまり利用されませんが、関数の実行結果によって、異なる型の戻り値を返すことも可能です。

2　関数の利用

次は、関数を呼び出す方法を紹介します。

関数名の後ろに( )を付け、この中に引数を指定して関数を呼び出します。引数が定義されていない関数の場合、引数の指定は省略します。

```
関数名([引数、引数、……]);
```

関数の呼び出し例

```
calcCone(10, 5);
```

関数を右辺にして左辺に変数を指定することで、関数の戻り値を変数に代入できます。

sample08_01.html

```
var base = calcCone(100, 50);
alert('計算結果：' + base);
```

また、関数の引数や if 文などの条件文に、関数の戻り値をそのまま使用することも可能です。

```
if (calcCone(100, 50) >= 50) {
  alert('計算結果を表示します');
}
```

【補足】関数式

JavaScript では、処理結果をそのまま変数に代入できる関数を定義できます。この関数のことを「関数式」と呼びます。

関数式を利用する場合も function による関数定義と同様の記述形式となりますが、関数名は省略できます。この場合、関数は型として扱われるのが特徴です。

```
var　変数名　=　function(引数, 引数, …){
  複数の文
  return　戻り値
}
```

【補足】無名関数

関数名がない関数のことを「無名関数（匿名関数）」と呼びます。無名関数は関数の利用箇所に合わせて関数を定義するため、呼び出し箇所とその処理の対応関係が理解しやすくなります。さらに、識別子が不要となり、識別子の重複に気を配る必要もありません。

9. String オブジェクト

JavaScript での文字列に関する処理では String オブジェクトを頻繁に利用します。文字列を変数に代入するには文字列「" "」を使用する方法もありますが、String オブジェクトでも同様の処理が可能です。String オブジェクトには、文字列を格納するだけでなく、「文字列の加工」や「特定の文字の検索」など、文字列を扱う際に便利なメソッドが用意されています。

String オブジェクトは、引数に文字列を指定して生成します。

```
var 変数名 = new String(文字列);
```

また、String オブジェクトが持つフィールド、メソッドを以下に記載します。

フィールド名	説明
length	文字列の長さ(文字数)を示す

メソッド名／構文名	説明
charAt(index)	引数に指定した位置(index)にある文字を返す
charCodeAt(index)	引数に指定した位置(index)にある文字コードを返す
concat(str1[,str2...])	String オブジェクトが保持している文字列から、引数に指定した文字列(str)を追加した内容を返す
indexOf(searchStr[,start])	String オブジェクトが保持している文字列から、引数に指定した文字列(searchStr)を検索し、その位置を返す(第 2 引数に開始位置(start)が指定されている場合はその位置から検索を開始する)
lastIndexOf(searchStr[,start])	String オブジェクトが保持している文字列の後ろから前へ、引数に指定した文字列(searchStr)を検索し、その位置を返す(引数に開始位置(start)が指定されている場合はその位置から前に向かって検索する)
search(regex)	引数に指定した正規表現のパターン(regex)に一致した場合、文字列内で一致した箇所のインデックスを返し、一致しなかった場合は-1 を返す
match(regex)	引数に指定した正規表現のパターン(regex)に一致した場合、一致したパターンの全情報を配列で返し、一致しなかった場合は null を返す
replace(condition,replacement)	引数に指定した検索条件(condition)に一致する文字列を、別の文字列(replacement)に置換する(検索条件には文字列または正規表現が指定できる)
toLowerCase()	文字列をすべて小文字に変換した文字列を返す
toUpperCase()	文字列をすべて大文字に変換した文字列を返す
toString()	String オブジェクトが保持している文字列を返す
trim()	文字の両端のスペース(半角、全角、タブ、改行)を取り除く
valueOf()	String オブジェクトが保持している文字列を返す

slice(start[,end])	引数に指定した開始位置(start)と終了位置(end)で、文字列を切り出して返す(引数 end を省略した場合は文字列の最後まで切り出す)
split(delimiter[,limit])	String オブジェクトが保持する文字列を、引数に指定した区切り文字(delimiter)で分割した配列を返す(引数に上限要素数(limit)が指定されている場合は、この数を超えた分は無視される)
substring(start[,end])	引数に指定した開始位置(start)と終了位置(end)で文字列を切り出して返す(引数 end を省略した場合は文字列の最後まで切り出す)

たとえば、match()メソッドを使用すると、指定した正規表現のパターンに一致する文字が、文字列中に存在するか否かを調べることができます。

sample09_01.js

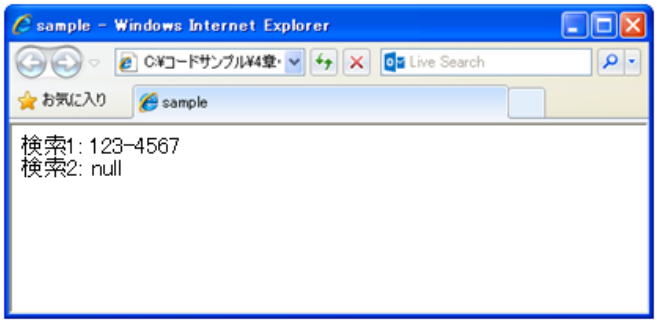
```
var target1 = '123-4567';
// 正規表現を使用して任意の数字を検索する
var result1 = target1.match(/^[0-9]{3}-[0-9]{4}$/);

document.write('検索 1: ' + result1 + '<br>');

var target2 = '1234567';
// 正規表現を使用して任意の数字を検索する
var result2 = target2.match(/^[0-9]{3}-[0-9]{4}$/);

document.write('検索 2: ' + result2 + '<br>');
```

実行結果（sample09_01.html を実行）



match()メソッドは、引数に指定した正規表現のパターンのうち、どのパターンが一致したのかを配列で返します。

また、「/」で囲まれた文字の並びは「正規表現」と呼ばれるものです。ここでは、「/^[0-9]{3}-[0-9]{4}$/」という正規表現を使用して、指定した文字列が郵便番号の形式であるか否かを調べています。正規表現のパターンに一致しなかった場合は null が返されます。

【補足】正規表現

「正規表現」とは、文字の並びのパターンを表記した式のことです。正規表現は JavaScript や Java のソースコード中に記述でき、「半角数字 5 文字」や「半角英小文字、または半角記号 250 文字」など、特定の文字の並び方にある文字列が一致しているかの判定処理などに利用できます。

正規表現は、様々な特殊文字を組み合わせて記述します。以下に代表的な特殊文字を記載します。

特殊文字	説明
^	文字列の先頭
$	文字列の末尾
*	直前の文字を 0 回以上繰り返す
+	直前の文字を 1 回以上繰り返す
?	直前の文字を 0 回または 1 回繰り返す
{n, m}	直前の文字を n 回以上 m 回以下繰り返す
{n,}	直前の文字を n 回以上繰り返す
{n}	直前の文字を n 回繰り返す
[a-z]	a から z の小文字アルファベットを表す
[A-Z]	A から Z の大文字アルファベットを表す
[0-9]	数値を表す
[^a-z]	a から z の小文字アルファベット以外を表す

また、以下に正規表現の記述例を記載します。

記述例：文字列の先頭が半角数字 3 文字

```
^[0-9]{3}
```

記述例：文字列の末尾が半角数字 3 文字

```
[0-9]{3}$
```

記述例：文字列がちょうど半角数字 3 文字

```
^[0-9]{3}$
```

記述例：文字列が 0 文字以上の半角英小文字

```
^[a-z]*$
```

記述例：文字列が 1 文字以上の半角英小文字

```
^[a-z]+$
```

記述例：文字列が 0 文字もしくは 1 文字の半角英小文字

```
^[a-z]?$
```

記述例：文字列が 3 文字以上かつ 5 文字以下の半角英小文字

```
^[a-z]{3, 5}$
```

記述例：文字列が 1 文字以上の半角英数字

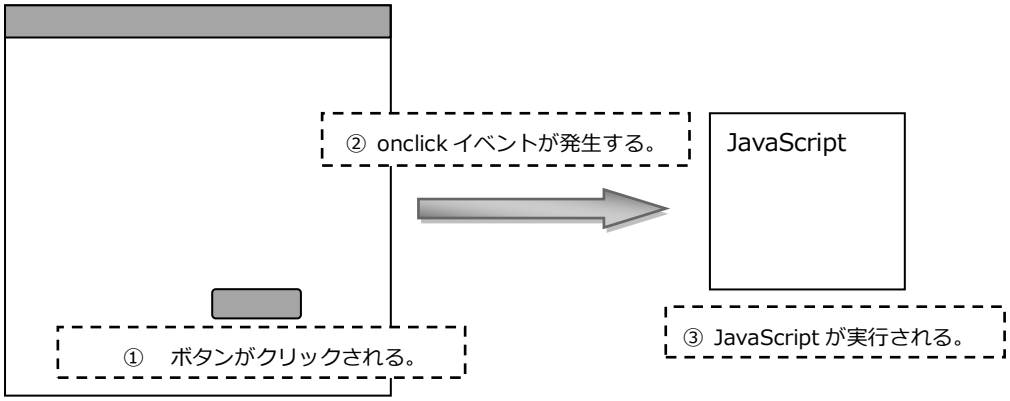
```
^[a-zA-Z0-9]+$
```

記述例：文字列が半角数字以外の文字列（1 文字以上）

```
^[^0-9]+$
```

10. イベントハンドラ

イベントとは「ボタンがクリックされた」「入力項目からフォーカスが外れた」「マウスが特定の要素に重なった」など、ユーザの操作に応じて発生する信号のようなものです。イベント発生時に処理を実行する特殊なコードのことをイベントハンドラと呼びます。イベントハンドラにより、任意の JavaScript の処理を実行できます。この手法は JavaScript に限らず、GUI を伴うアプリケーションでは一般的で、「イベント駆動型」のプログラミングモデルと呼ばれています。



JavaScript でイベント処理を記述するには、HTML タグのイベントハンドラ属性を用います。イベントハンドラには、「on イベント名」という名称の属性で、イベントの種類に応じて様々なものが存在します。

```
<input type="button" value="クリック"  onclick="alert('ボタンが押されました')"/>
```

イベントハンドラ属性にはセミコロン(;)で区切ることで、複数の文を記述することもできます。

```
<input type="button" value="クリック" onclick="alert('セミコロンで区切って'); alert('複数の文を記述できます')"/>
```

しかし、属性値に多くの JavaScript を記述すると HTML が見づらくなります。そのような場合には、イベント処理を script 要素や外部ファイルに関数として定義しておき、イベントハンドラ属性にはそれらの関数を呼び出すコードを記述します。

sample10_01.js

```
function showDialog () {
  alert(document.form.name.value + 'さん、こんにちは！');
}
```

sample10_01.html

```
～略～
<body>
  <form name="form">
    <input type="text" name="name">
    <input type="button" value="クリック" onclick="showDialog()"/>
  </form>
</body>
～略～
```

HTML タグのイベントハンドラ属性には次のようなものがあります。

属性名	説明(呼び出されるタイミング)
onclick	クリックされたとき
ondblclick	ダブルクリックされたとき
onmousedown	マウスのボタンが押下されたとき
onmouseup	マウスのボタンを放したとき
onmouseover	マウスが要素上に移動したとき
onmouseout	マウスが要素上から外れたとき
onmousemove	マウスが要素内で移動したとき
onchange	フォーム内の要素について値が変更されたとき
onblur	フォーム内の要素についてフォーカスが外れたとき
onfocus	フォーム内の要素についてフォーカスが当たったとき
onkeydown	キーを押したとき
onkeypress	キーを押して放したとき
onkeyup	キーを放したとき
onselect	テキストフィールドやテキストエリアでテキストが選択されたとき
onsubmit	フォームが送信される前。属性値に記述した JavaScript が true を返す場合はフォームが送信され、false を返す場合はフォームが送信されない。
onreset	フォームがリセットされる前。属性値に記述した JavaScript が true を返す場合はフォームがリセットされ、false を返す場合はフォームがリセットされない。
onload	Web ページがロードされたとき
onunload	Web ページがアンロードされるとき

11. Web ブラウザの制御

JavaScript の処理により Web ブラウザを制御するためには、Window オブジェクトを使用します。

「Window オブジェクト」とは、Web ブラウザ環境で動作する JavaScript のグローバルオブジェクトです。Window オブジェクトは「window」という変数で参照できます。Window オブジェクトは Web ブラウザを制御したり、Web ページのコンテンツにアクセスしたりするための様々なフィールドやメソッドを提供しています。たとえば、アラートダイアログを表示する alert()関数は、実は Window オブジェクトのメソッドとして定義されています。つまり、以下の 2 つのコードは等価です。

```
alert('Hello JavaScript ! ');
window.alert('Hello JavaScript ! ');
```

Window オブジェクトのフィールドとメソッドを示します。

フィールド名	説明
closed	Window が閉じているかどうか
document	Document オブジェクト
frames[]	このウィンドウ内のフレーム
history	History オブジェクト
innerHeight,innerWidth	ウィンドウの表示領域の高さと幅(ピクセル単位)。Internet Explorer 8 以前はこれらのフィールドをサポートしていない。
localStorage	永続的なクライアントサイドストレージ
location	Location オブジェクト
name	ウィンドウの名前。open()メソッドの引数や、frame 要素の name 属性で指定された値。この名前は a 要素や form 要素の target 属性の値として指定できる。
navigator	Navigator オブジェクト
opener	open()メソッドでこのウィンドウを開いたウィンドウ
outerHeight,outerWidth	ウィンドウの高さと幅(ピクセル単位)。Internet Explorer 8 以前はこれらのフィールドをサポートしていない。
pageXOffset,pageYOffset	水平方向、垂直方向へのスクロール量(ピクセル単位)。Internet Explorer 8 以前はこれらのフィールドをサポートしていない。
parent	このウィンドウがフレーム内のウィンドウである場合、親ウィンドウへの参照。このウィンドウ自身がトップレベルウィンドウの場合、自分自身の参照
screen	Screen オブジェクト
screenLeft,screenTop	画面上でのウィンドウの左上の座標。Firefox はこのフィールドをサポートしていない。
screenX,screenY	画面上でのウィンドウの左上の座標。Internet Explorer 8 以前はこれらのフィールドをサポートしていない。
self	window フィールドと同じくこのウィンドウ自身への参照

sessionStorage	ウィンドウやタブを閉じるまで有効なクライアントサイドストレージ
status	Web ブラウザのステータスバーに表示する文字列
top	このウィンドウがフレーム内のウィンドウである場合、トップレベルウィンドウへの参照。このウィンドウ自身がトップレベルウィンドウの場合、自分自身の参照
window	self フィールドと同じくこのウィンドウ自身への参照

メソッド名／構文	説明
alert(message)	引数に指定したメッセージ(message)で警告ダイアログを表示する
blur()	ウィンドウからフォーカスを外す
clearInterval(intervalID)	引数に指定した ID(intervalID)の繰り返し処理をキャンセルする
clearTimeout(timeoutID)	引数に指定した ID(timeoutID)のタイマー処理をキャンセルする
close()	ウィンドウを閉じる
confirm(question)	引数に指定したメッセージ(question)で確認したダイアログを表示する
focus()	ウィンドウにフォーカスを与える
moveBy(x,y)	ウィンドウを、引数に指定したピクセル分(x,y)移動する
moveTo(x,y)	ウィンドウを、引数に指定した座標(x,y)に移動する
open()	新しいウィンドウを開く
print()	表示しているドキュメントを印刷する
prompt(message,default)	ユーザからの入力値を取得するためのダイアログを表示する。第 1 引数にはダイアログに表示する文字列(message)、第 2 引数には入力エリアの初期値(default)を指定する
resizeBy(width,height)	ウィンドウを、引数に指定したピクセル数(width,height)分広げる
resizeTo(width,height)	ウィンドウを、引数に指定したサイズ(width,height)にリサイズする
scrollBy(x,y)	ウィンドウを、引数に指定したピクセル数(x,y)分スクロールする
scrollTo(x,y)	ウィンドウを、引数に指定した座標(x,y)にスクロールする
setInterval(code,interval)	引数に指定したコード(code)を一定間隔(interval)で実行する
setTimeout(code,delay)	引数に指定したコード(code)を指定時間(delay)経過後に実行する
postMessage(message,targetOrigin)	異なるサーバのコンテンツを表示しているウィンドウ間で通信を行なう

警告ダイアログ

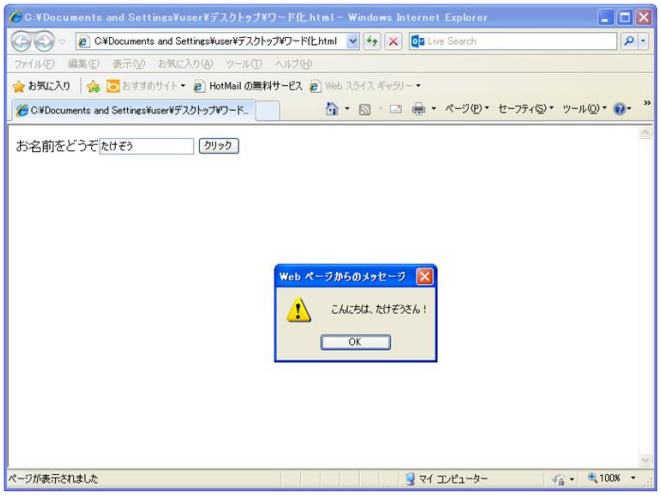
alert()メソッドは警告ダイアログを表示します。作成中のスクリプト内で変数の値を表示させるなど、簡単なデバッグ用途に利用することもあります。

alert()メソッドの利用例を次に示します。

sample11_01.html

```
<!DOCTYPE html>
<html>
  <body>
    <form name = "form">
      お名前をどうぞ<input type= "text" name= "name"/>
      <input type= "button" value= "クリック"
          onclick= "alert('こんにちは、' + document.form.name.value + 'さん！')"/>
    </form>
  </body>
</html>
```

実行画面



確認ダイアログ

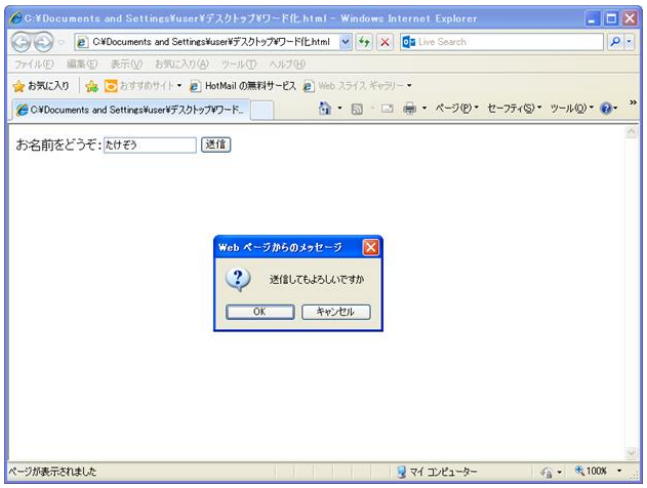
confirm()メソッドは確認用のダイアログを表示します。[OK]が選択された場合は戻り値が true、[キャンセル]が選択された場合は戻り値が false になります。

confirm()メソッドの利用例を次に示します。form 要素の onsubmit 属性はスクリプトが true を返す場合のみフォームを送信します。したがって、この例では確認ダイアログで[OK]ボタンが押された場合のみフォームが送信されます。

sample11_02.html

```
<!DOCTYPE html>
<html>
  <body>
    <form name="form" onsubmit="return confirm('送信してもよろしいですか')">
      お名前をどうぞ：<input type="text" name="name"/>
      <input type="submit" value="送信"/>
    </form>
  </body>
</html>
```

実行画面



12. フォームの制御

HTML では form 要素によって入力フォームを作成できます。入力フォームは Web アプリケーションにおいてユーザからの入力値を取得するために欠かせません。JavaScript を利用すると HTML の入力フォームに以下のような機能を付け加え、Web アプリケーションの使い勝手を向上させることができます。

入力チェック

Web アプリケーションでは通常サーバサイドで入力チェックを行いますが、JavaScript を利用することで、クライアントサイドでも入力チェックを行うことが可能です。

ただし、JavaScript は Web ブラウザの設定で無効にすることもできるため、JavaScript で入力チェックする場合でもサーバサイドでの入力チェックはあわせて実施する必要があります。

入力補助

フォーカスが外れたタイミングで入力値をフォーマットしたり、ある項目の入力内容によって別の項目の値を自動的に設定するなど、JavaScript によってユーザの入力を補助します。

もちろん工夫次第で他にも様々な活用方法が考えられます。

JavaScript から入力フォームへのアクセス

JavaScript から入力フォームへアクセスするには Document オブジェクトを使用します。例として以下のような入力フォームの場合を考えてみましょう。

```
<form name="loginForm">
  ユーザ ID：<input type="text" name="userId"/><br>
  パスワード：<input type="password" name="password"/>
  <input type="submit" value="ログイン"/>
</form>
```

フォームには Document オブジェクトの forms フィールド、フォームの入力項目には Form オブジェクトの elements フィールドを利用してアクセスできます。たとえば、ユーザ ID のテキストフィールドに値を取得/設定するには次のようにします。

```
// テキストフィールドの値を取得
var userID = document.loginForm.userId.value;
// テキストフィールドに値を設定
document.loginForm.userId.value = 'たろう';
```

フォームの入力項目

フォーム内にはテキストフィールドやラジオボタン、チェックボックスなど、様々な入力項目を配置できます。HTML で利用可能なフォームの入力項目には次のようなものがあります。

それぞれの入力項目によってサポートされているフィールドは異なります。入力項目ごとにサポートされているフィールドを示します。

フィールド	入力項目						
	text password textarea	checkbox	radio	select	option	file	submit reset button
checked		○	○				
defaultChecked		○	○				
defaultSelected					○		
defaultValue	○						
index					○		
length			○	○	○		
name	○	○	○	○		○	○
options				○			
selected					○		
selectedIndex				○			
text					○		
type	○	○	○	○		○	○
value	○	○	○		○	○	○

入力値の取得と設定

これまでの例でも見てきたように、JavaScript では入力項目の value フィールドで入力値の取得/設定ができます。

sample12_01.html

```
～略～
<form name="form">
  <input type="text" name="name" value="">
</form>
<script  type="text/javascript">
  // テキストフィールドの値を取得
  var name = document.form.name.value;
  // テキストフィールドに値を設定
  document.form.name.value = 'たろう';
</script>
～略～
```

入力項目の状態を変更する

フォームの入力項目には状態を変更するためのフィールドが用意されています。disabled フィールドに true を設定すると、その項目は操作ができなくなります。

```
<input type="text" name="userId" disabled>
```

HTML の input 要素に disable 属性を指定した場合と同様、その項目は無効となり、入力はもちろんフォーカスを当てることもできなくなります。フォームの送信時にも値は送信されなくなります。

また、disable 属性に false を設定すると、その項目は有効となり、操作可能になります。

sample12_02.html

```
～略～
<form name="form">
  <input type="text" name="userId">
</form>
<script   type="text/javascript">
  // userId フィールドを無効にする
  document.form.userId.disabled = true;
</script>
～略～
```

■型
基本的なjavascriptでは、変数に対して型の宣言を行うことは出来ない。
そのため変数に代入できる値に対し型の制限を行うことも出来ない。
また値が代入されていない変数については、初期値として
undefined(未定義)が代入される。
■イベントハンドラ
onclick・・・対象をクリックしたときにイベントが発生する
onfocus・・・対象にフォーカスしたときにイベントが発生する。
※input textなど
onmouseover・対象にマウスオーバーしたときにイベントが発生する
onload・・・対象が呼び込まれた際にイベントが発生する。

13. DOM 操作

1　概要

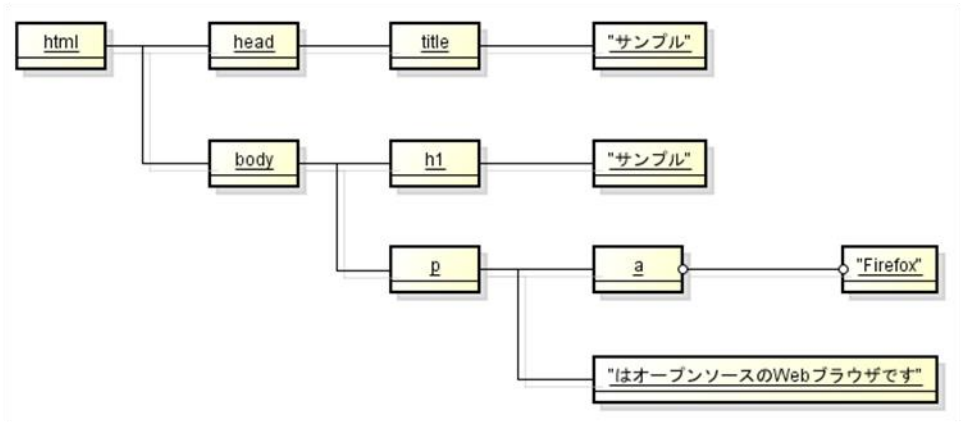
DOM(ドキュメントオブジェクトモデル)とは、XML や HTML などのツリー構造を操作するための機能です。DOM を使用することで、HTML 文書のすべての要素にアクセスし、変更を加えることができます。

たとえば、次のような HTML のソースコードがあったとします。

```
<!DOCTYPE html>
<html>
  <head>
    <title>サンプル</title>
  </head>
  <body>
    <h1>サンプル</h1>
    <p id="paragraph1">
      <a href="http://mozilla.jp/firefox/">Firefox</a>
          はオープンソースの Web ブラウザです。
    </p>
  </body>
</html>
```

DOM は、HTML の階層構造を持ったドキュメントをツリー構造の形式で表現します。HTML の要素間の親子関係をそのままツリーにしたものと考えると分かりやすいでしょう。

たとえば、上記の HTML のソースコードは、次のようなツリー構造で表せます。



ツリーの各要素を「ノード」と呼びます。ノードは、Document オブジェクトから各階層を辿って、目的のノードにアクセスできます。

なお、本書では、W3C によって標準化されている DOM（W3C DOM）の基本的な利用方法を紹介します。

2　利用方法

要素の取得

ドキュメントの DOM ツリーには Document オブジェクトからアクセスできます。Document オブジェクトは Window オブジェクトの document フィールドとして定義されており、

```
window.document
```

もしくは単純に、

```
document
```

と記述することで参照できます。

Document オブジェクトには、「要素名」を指定して要素を取得する getElementsByTagName()メソッド、要素の「id 属性」を指定して要素を取得する getElementById()メソッド、要素の「name 属性」を指定して要素を取得する getElementsByName()メソッドがあります。

```
// 要素名を指定し、配列として取得
var h1 = document.getElementsByTagName('h1')[0];

// id 属性の値を指定して要素を取得
var p = document.getElementById('paragraph1');

// name 属性の値を指定し、配列として取得
var empIdField = document.getElementsByName('empId')[0];
```

要素内容の取得

要素内容（タグに囲まれたテキスト）を取得するには、textContent フィールドを使用します。

下記のサンプルコードは、<p>タグ上でマウスをクリックすると、<p>タグに囲まれたテキストを取得します。

sample13_01.html

```
～略～
<p id="dom" onclick="replaceText()">テキスト</p>
<script>
  function replaceText() {
    var element = document.getElementById('dom'); // id 属性の値が"dom"である要素を取得
    alert(element.textContent); // タグに囲まれたテキストを取得し、警告ダイアログに出力
  }
</script>
～略～
```

要素内容の変更

要素内容（タグに囲まれたテキスト）を変更する際にも、textContent フィールドを使用します。
下記のサンプルでは、<p>タグ上でマウスをクリックすると、<p>タグに囲まれたテキスト変更します。

sample13_02.html

```
～略～
<p id=”dom” onclick="replaceText()">元のテキスト</p>
<script>
  function replaceText() {
    var element = document.getElementById(‘dom’); // id 属性の値が”dom”である要素を取得
    element.textContent = ‘置換後のテキスト’; //  タグに囲まれたテキストを引数 text の値に変更
  }
</script>
～略～
```

また、innerHTML フィールドを使用することで、HTML 文書も挿入できます。

```
<p onclick="this.innerHTML = '<a href=”〇〇〇”>置換後のテキスト</>'">
  元のテキスト
</p>
```

フォームの入力値の取得

フォームの入力値（入力項目の入力値）を取得するには、以下の 2 段階の処理を実装します。

1. 入力項目のオブジェクトを取得する。
2. 入力項目のオブジェクトが持つフィールド「value」の値を取得する。

入力項目のオブジェクトには、入力値を保存するためのフィールド value が存在します。そのフィールドを参照することで、入力値を取得できます。

下記のサンプルコードでは、テキストボックスに入力した値を取得します。

sample13_03.html

```
～略～
<script>
  function getInput() {
    var textBox = document.getElementById('empId'); // id 属性の値が"empId"である要素を取得
    var empId = textBox.value; // テキストボックスの入力値を取得
    alert(empId);
  }
</script>
<form name="form">
    社員 ID<input type="text" name="empId" id="empId" />
    <input type="button" value="Click!" onclick="getInput();"/>
</form>
～略～
```

属性値の取得

属性値を取得するには、getAttribute()メソッドを使用します。
下記のサンプルコードでは、ボタンをクリックすると、<a>要素の href 属性の値を取得します。

sample13_04.html

```
～略～
<a id="link" href="http://www.google.co.jp/">Google</a>
<input type="button" value="クリック" onclick=" getLinkTarget()"/>
<script>
  function getLinkTarget() {
    var link = document.getElementById('link');
    var href = link.getAttribute('href'); // href 属性の値を取得
    alert(href);
  }
</script>
～略～
```

また、属性名と同名のフィールドを参照することでも、属性の値を取得できます。

```
// 「var href = getAttribute('href');」と同じ処理
var href = link.href;
```

東京ITスクール
TOKYO IT SCHOOL

属性値の変更

属性値を変更するには、setAttribute()メソッドを使用します。
下記のサンプルコードでは、ボタンをクリックすると、<a>要素の href 属性の値を「http://www.google.co.jp/」から「http://www.yahoo.co.jp/」に変更します。

sample13_05.html

```
～略～
<a id="link" href="http://www.google.co.jp/">Google</a>
<input type="button" value="クリック" onclick="changeLinkTarget()"/>
<script>
  function changeLinkTarget() {
    var link = document.getElementById('link');
    link.setAttribute('href', 'http://www.yahoo.co.jp/'); // href 属性の値を変更
  }
</script>
～略～
```

また、属性名と同名のフィールドに値を代入することでも、属性の値を変更できます。

属性値の変更

```
//  「link.setAttribute('href', 'http://www.yahoo.co.jp/');」と同じ処理
link.href = ‘http://www.yahoo.co.jp/’;
```

■DOM（ドキュメントオブジェクトモデル）
・HTMLタグの階層構造をツリー形式で表現したもの。
html->head->title
　　->body->section->h1
　　->body->header->ul
・DOMツリーの各要素には階層的にアクセス出来る。
・DOMツリーの各要素はノード。
・属性値やタグに囲まれたテキストなども、要素の保持する情報として扱うことが出来る。
■ノード取得
document.getElementById('ID名')
　　・・・ID指定で要素を取得
document.getElementsByTagName('タグ名')
　　・・・タグで指定した要素を配列で取得
document.getElementsByName('name属性の値')
　　・・・name属性で指定した要素を配列で取得
など