SECTION 01

# 変数を操作してみよう

プログラムで何かの処理を実行するときは、値を格納している変数を介して何らかの操作する場合がほとんどです。宣言した変数は、自由に値を変更したり、処理を行ったりできます。その際、プログラミング言語ごとの規則に従う必要があります。このSECTIONでは、JavaScriptにおける変数の基本的な扱い方を解説します。

## ◎ 演算子とは

**演算子**とは、プログラム内で**演算**（計算）のほか特別な処理をするための記号の総称です。たとえば、変数に値を代入する際に利用した「=」（P.37～38参照）や、オブジェクトのインスタンスを生成する際に利用したnew（P.45参照）なども演算子の一種です。

前述のように、プログラムで処理をするということは、変数に対して何らかの処理することと同じです。以降では、変数に対して基本的な処理を行うための演算子を紹介します。

## ◎ 四則演算を行う

**四則演算**（加算、減算、乗算、除算）は数値型の変数に対する基本的な処理です。四則演算で使う演算子を**算術演算子**といいます。算術演算子には、表3-1にあげた種類があります。





表 3-1 算術演算子の種類

| 演算子 | 演算名 | 利用例 |
|---|---|---|
| + | 足し算（加算） | a + b |
| - | 引き算（減算） | a - b |
| * | 掛け算（乗算） | a* b |
| / | 割り算（除算） | a / b |
| % | 余り（除算の余り） | a % b |

リスト3-1は、変数の値で加算と乗算を実行するサンプルです。

リスト 3-1 calculation.html

```
省略
// 数値型の変数を宣言
var numA = 2;
var numB = 3;

// 2つの変数を加算して出力
var numC = numA + numB;
console.log(numC);

// 2つの変数を乗算して出力
var numD = numA * numB;
console.log(numD);
省略
```

リスト3-1を実行すると、コンソールには以下のように出力されます。

```
5
6
```

リスト3-1のサンプルのように、四則演算は数値型の変数の間に算術演算子を置いて記述します。

## ◎ 変数の値を比較する

変数の値を比較するための演算子を**比較演算子**といいます。比較演算子には、表3-2にあげた種類があります。

**表3-2 比較演算子の種類**

| 演算子 | 利用例 | 概要 |
|---|---|---|
| == | a == b | aとbが等しければtrue、そうでなければfalse |
| === | a === b | aとbが型を変えずに等しければtrue、そうでなければfalse |
| != | a != b | aとbが等しくなければtrue、そうでなければfalse |
| !== | a !== b | aとbが型を変えずに等しくなければtrue、そうでなければfalse |
| < | a< b | aがbよりも小さければtrue、そうでなければfalse |
| <= | a <= b | aがb以下であればtrue、そうでなければfalse |
| > | a> b | aがbよりも大きければtrue、そうでなければfalse |
| >= | a >= b | aがb以上であればtrue、そうでなければfalse |

比較演算子による比較の結果は、真偽型（P.38参照）のtrueまたはfalseが返されます。リスト3-2は、比較演算子で変数の値を比較するサンプルです。

**リスト3-2 comparison.html**

```
省略
// 数値型の変数を宣言
var numA = 1;
var numB = 2;

// numAとnumBが等しいか？
var boolC = numA == numB;
console.log(boolC);

// numAよりnumBが大きいか？
var boolD = numA < numB;
console.log(boolD);
省略
```

リスト3-2を実行すると、コンソールには以下のように出力されます。

```
false
true
```

リスト3-2のサンプルでは、変数numAとnumBを比較して、結果を変数boolCとboolDに格納しています。boolCとboolDに渡される値は、==演算子と<演算子による比較の戻り値です。

### ◎ ===演算子と!==演算子

前述の比較演算子のうち、==演算子と!=演算子で変数どうしを比較すると、型の違いは無視して値を比較した結果を返します。一方、===演算子と!==演算子は型も含めて値を比較した結果を返します。

リスト3-3は、==演算子と===演算子を利用して、数値型と文字列型の変数を比較するサンプルです。

**リスト3-3 comparison_strictly.html**

```
省略
// 数値型の変数を宣言
var numA = 1;
// 文字列型の変数を宣言
var numB = "1";

// numAとnumBは等しいか？
console.log(numA == numB);

// 型まで参照してnumAとnumBは等しいか？
console.log(numA === numB);
省略
```

リスト3-3を実行すると、コンソールには以下のように出力されます。

```
true
false
```

リスト3-3のサンプルのように、数値型の1と文字列型の"1"を比較すると、==演算子と===演算子では結果が異なります。変数の型まで含めて判断する===演算子のほうが、厳密に値を比較します。これは!=演算子と!==演算子の場合でも同じです。

JavaScriptでは、変数を宣言する際に型を指定しません。このため、変数の比較で==演算子や!=演算子を使うと、プログラムが予期したとおりに動作しなくなる可能性があります。このような事態を避けるため、変数の値を比較する際は、型を含めて比較する===演算子または!==演算子を使うようにしましょう。

## ◎ 文字列を結合／比較する

文字列型の変数を結合する場合は、+演算子を利用します。リスト3-4は、+演算子で文字列を結合するサンプルです。

**リスト3-4 string_plus.html**

```
省略
// 文字列型の変数を宣言
var strA = "吾輩は";
var strB = "猫である";

// 2つの文字列を結合して出力
var strC = strA + strB;
console.log(strC);

// 2つの文字列を比較
console.log(strA === strB);
省略
```

リスト3-4を実行すると、コンソールには以下のように出力されます。

```
吾輩は猫である
false
```

リスト3-4のサンプルのように、文字列型の変数を結合する場合でも+演算子が利用できますが、数値型の変数を扱う場合と書式が同じなので注意が必要です。文字列型の変数を比較する場合も、前述と同様に===演算子または!==演算子を利用しましょう。





SECTION 02

# 処理の流れを制御してみよう

よく利用されるプログラムの処理の流れには、条件によって処理を分ける「条件分岐」と、同じ処理を繰り返す「繰り返し」の2つがあります。これらの処理は、**制御構文**と呼ばれるしくみで実現します。このSECTIONでは、よく利用されるif、for、whileなどの制御構文について説明します。

## ◎ 条件分岐

条件によって処理を分けるには、条件を満たす（条件に合致する）場合に処理を実行する**if文**、複数の条件を組み合わせる**if else文**を利用します。

### 条件を満たす場合に処理を実行する

if文は条件分岐の基本的な構文です。ifの後の( )内に**条件式**を記述し、続く{ }内に条件式を満たす場合に実行する処理を記述します。この{ }で囲まれた部分を**ブロック**といいます。

#### ▶ if文

**書式**
```
if(条件式){
    // 条件を満たす場合の処理
}
```

**概要** 1つの条件式を満たす場合の処理を実行

**パラメータ** **条件式** 結果をtrue／falseで返す条件を指定する式

条件式とは、前述の比較演算子を利用した処理のように、結果をtrueまたはfalseで返す式のことです。条件式の戻り値がtrueの場合はブロック内の処理が実行されますが、falseの場合は実行されません。リスト3-5は、if文による条件分岐のサンプルです。