今回は JavaScript の基本について話をします。

一つ一つの要素は簡単だけど、少し量が多いというのが大変なところ。ですが、この記事だけで基本的な項目は一通り学ぶことができるようになっています。

1. JavaScript とは
2. JavaScript の使い方
    1. HTML 上で読み込む
    2. 【練習用】ブラウザのコンソールを使う
    3. 【練習用】練習に使えるサイト
3. JavaScript の基本的な書き方
    1. 文
    2. ブロック
    3. コメント
    4. 出力
        1. コンソールに出力
        2. ダイアログに出力
        3. HTML として出力
    5. 変数宣言
    6. 数値
        1. 数値にも色々ある
        2. 算術演算
        3. ビット演算
        4. 代入演算
        5. インクリメント・デクリメント
    7. 文字列
        1. 文字列の書き方と特殊な文字
        2. 文字列の長さ
        3. 文字列の結合
        4. 文字列の分割
    8. 真偽値
        1. 比較演算
    9. データ構造
        1. 配列
        2. オブジェクト
    10. 条件文
        1. if 文・if-else 文
        2. switch-case 文
    11. 繰り返し文
        1. for 文

    2. while 文
12. 関数
4. JavaScript の型（動的型付け）
    1. 型とは
    2. 動的型付け
    3. 演算やその結果の型に注意
5. 第一級関数
    1. 他の値と同様に名前無しで扱える
    2. 関数の引数・返り値にできる
    3. プロパティが持てる
6. JavaScript におけるオブジェクト指向
    1. オブジェクト指向とは
    2. プロトタイプベースのオブジェクト指向
7. JavaScript を勉強する上での注意！
8. まとめ

JavaScript とは

JavaScript は、様々な環境で動作する軽量なプログラミング言語です。主に Web ブラウザ上で使われており、Web ページに動きをつけたり、計算をさせたり、様々なことができます。

プログラミング言語として、特徴的な性質がいくつかあるのですが、ここで説明してもわかりにくいので、キーワードだけ挙げておきます。すべて後述されておりますので、そこまでしばしお待ちを。

- **オブジェクト指向**
- **第一級関数**
- **動的型付け言語**

JavaScript の使い方

さて、JavaScript の実際の書き方を見ていく前に、**どうやって JavaScript のプログラムを動かしたらいいのか**話しておきます。

HTML 上で読み込む

まず、一般的な HTML 上で動かす場合です。この場合 2 つの書き方があります。

1 つ目は JavaScript をそのまま HTML の中に書く方法です。以下のようにして書きます。

```
<script type="text/javascript">
// ここにコードを書く
</script>
```

この場合は、<script></script>で囲まれた部分に JavaScript のコードを書いていきます。

2 つ目は外部ファイルとして読み込む方法です。以下のようにします。

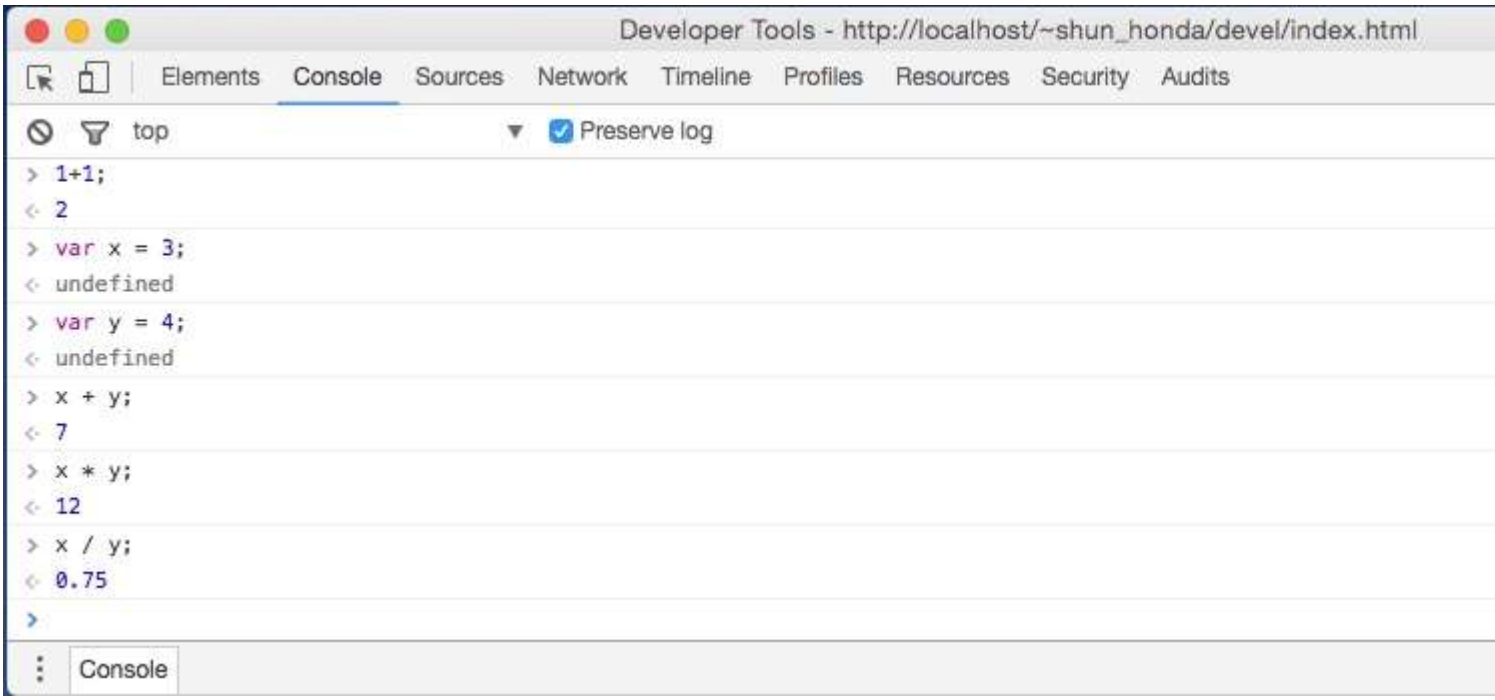
```
<script type="text/javascript" src="JavaScript のファイルパス"></script>
```

この場合は、HTML とは別のファイルに JavaScript を書いておき、それを読み込みます。

【練習用】ブラウザのコンソールを使う

手軽に JavaScript を動かして、試してみたい時にはブラウザで用意されたコンソールに打ち込むのがいいですね。

例えば、Google Chrome の場合、以下のようなコンソールです。このように打ち込んでいけば JavaScript を試すことができます。



【練習用】練習に使えるサイト

ブラウザのコンソールでなくても、JavaScript を試すことができるサービスはいくつかあります。その中で、「CODEPEN」というサービスを僕はオススメしています。

JsFiddle とおなじです

codepen.io

これを使って JavaScript の練習をするといいです。今回の内容に関しても、この CodePen で練習するのが一番やりやすいかと思います。

アカウント登録をしなくても使えるので、まずは開いてみましょう。開くと以下のような画面になります。

JavaScript の基本的な書き方

ここからは、実際の JavaScript の記法を解説していきます。

文

JavaScript のプログラムは「文」の集まりです。以下のような感じです。

```
var x = 3;
var y = 4;
var z = x + y;
console.log(z);
```

このコード自体の意味は、今はわからなくて大丈夫です。重要なのは文の終わりについているセミコロン(;)です。**文の終わりには基本的にセミコロンをつけます。**今後セミコロンが最後についているものが出てきても、「何だこれ？」ではなく「文の終わりなんだ」と理解してください。(セミコロンをつけなかった場合、自動的に挿入されますが、思わぬ動きをすることがあるので、きちんとつけましょう。)

ただし、ブロックが最後にくる文の終わりにはセミコロンをつけません。ブロックについては、次で説明します。

JavaScript はこのように、「文」からできているのですが、この文の種類が膨大にあります*1。ですから、それをひたすら覚える作業です。ただ、そう身構えなくても使っている間に慣れてくるものが多いので大丈夫です。また、この記事では基本的な項目を網羅しているので、忘れたらその都度読み返せばいいです。

ブロック

ブロックというのは、0 個以上の文をグループにするためのものです。以下のように書きます。

```
{
  var x = 3;
  var y = 4;
  var z = x + y;
  console.log(z);
}
```

これについても、今はコードの意味はわからなくていいです。中括弧で囲まれた部分がグループになっているということだけ覚えてください(オブジェクトとは違うので注意)。

先ほど説明した通り、ブロックの最後にはセミコロンをつけません。今後、ブロックが最後についた文が出てきますが、その時はセミコロンをつけないというように思ってください。

コメント

まず、実際の動作とは全く関係ない部分で、コメントについて説明します。コメントとは、その名の通り、コメントです。**実際の動作とは全く関係ないメモのようなもの**だと思ってください。これを適切に使うことで、コードの読みやすさが上がります。また、他人や数週間後の自分が理解しやすくするため、という意味も大きいです。

書き方は以下の 2 通りあります。1 行のみのコメントと、複数行書けるコメントがあるので使い分けるといいです。

```
// 1行のコメント

/*
 複数行の
 コメント
*/
```

出力

JavaScript では、様々な計算や複雑な動作などをさせることができますが、その結果が見えないとわかりにくいですよね。ですので、**まずは数値とか文字を出力する方法を説明します**。（数値や文字列などの違いはのちほど説明しますので、今は雰囲気で察してください←ごめんなさい）

ここでは、出力の仕方を 3 つご紹介します。それぞれ使う場面が違うので、きちんと覚えるようにしましょう。

コンソールに出力

コンソールに表示する方法です。これは、自分の手元で正しく動いているかを確認したりするために使うことが多いです。多くのブラウザで使うことができます。

書き方は以下のような感じです。

```
console.log("Hello World"); // 文字列をコンソールに出力
```

もちろん、数値でもできます（それ以外のものも出力できます）。

```
console.log(1); // 数値をコンソールに出力
```

この結果を、CodePen で見るときは以下のようにコンソールを出してください。

左下のコンソールボタンをクリックします。

するとこのようにコンソール画面が出てきます。

ここから先の、JavaScript の書き方を勉強するときには、コンソールを使って結果を確認するといいです。

結果が見たいときには、console.log( )のカッコの中に見たいものを書くようにしましょう。

ダイアログに出力

ダイアログに表示する方法です。下の画像のような感じです。多分一度はみたことあると思います。

このように表示させるには、alert というものを使います。以下のように書くと出てきます。もちろん、数値でもできます。（CodePen でやるとダイアログが結構激しく出てくるので、確認程度で OK です）

```
alert("Hello World!!"); // ダイアログに出力
```

HTML として出力

実際に HTML として出力するための書き方です。ここでは innerText を使った書き方を解説します（document.write じゃないの？　と思った方。確かにできるのですが、現在非推奨ですのであしからず）。

実際にやってみた例としては以下のような感じ。このようにすると、<body></body>の中に「Hello World!!」という文字が入ります。

```
document.body.innerText = "Hello World!!"; // body に出力
```

変数宣言

変数は、扱いたい数値や文字列などのデータを格納し、名前をつけたものです。数学で使う変数と同じようなものだと思っていただいて大丈夫です。

変数の宣言の仕方は以下のようになります。

```
var value = 1; // 変数 value を宣言し、数値 1 を代入
```

変数宣言は var を先頭につけ、その後変数の名前を書き、値を代入します。ここでは数値を代入していますが、他のものでも構いません。

また、**一度宣言した変数には、後から違う値を代入することができます。**

```
var value = 1;
value = 2; // 再代入
```

ここで注意ですが、var で宣言していない変数への代入も行うことはできてしまうということです。ただし、var で宣言しなかった場合は少し面倒な挙動をしますので、推奨はされていません。**できる限りちゃんと var で宣言した方がいいよ**、ということです。

数値

ここでは、数値と、それを用いた演算についての説明をします。

数値にも色々ある

数値の書き方は色々あります。

```
15; // 10 進数の整数（ただし、実数として処理される）
15.5; // 実数
0x000f; // 16 進数
0017; // 8 進数
0b1111; // 2 進数
```

ですが、基本的に 10 進数を使うことが多いので、特に意識しなくてもいいように思います。注意点としては、**JavaScript には整数という概念がなく、全て実数として処理されるということです**。その点だけ注意しましょう。

算術演算

数値を使った演算です。足し算とか引き算とかですね。以下のように書きます。

```
1 + 2; // => 3
3 - 1; // => 2
2 * 3; // => 6
3 / 2; // => 1.5 (JavaScript では全て実数)
3 % 2; // => 1 (余り)
```

ビット演算

ビット演算の書き方です。ビット単位での論理演算やシフト演算を行います。

```
7 & 2; // => 2 (AND)
5 | 2; // => 7 (OR)
7 ^ 2; // => 5 (XOR)
~105; // => -106 (NOT)
16 << 1; // => 32 (左シフト)
15 >> 1; // => 7 (右シフト)
```

代入演算

代入演算は、ただ単に変数に値を入れるだけでなく、他の演算と組み合わせた書き方ができます。 以下のように書きます。

```
var a = 0;
a = 1; // => 1
a += 1; // => 2 (a = a + 1;)
a -= 1; // => 1 (a = a - 1;)
a *= 6; // => 6 (a = a * 6;)
a /= 2; // => 3 (a = a / 2;)
a %= 2; // => 1 (a = a % 2;)

a = 5;
```

```
a &= 3; // => 1 (a = a & 3;)
a |= 3; // => 3 (a = a | 3;)
a ^= 2; // => 1 (a = a ^ 2;)
a <<= 1; // => 2 (a = a << 1;)
a >>= 1; // => 1 (a = a >> 1;)
```

例えば、a += 1;は a = a + 1;という意味になります。他の演算も同様です。

インクリメント・デクリメント

変数の値を 1 増やす、もしくは減らすための書き方です。

```
var a = 1;
++a; // => 2 (a の値も 2)
a++; // => 2 (a の値は 3)
--a; // => 2 (a の値も 2)
a--; // => 2 (a の値は 1)
```

このとき、変数の前に演算子をつけると、式の返す値が評価後の値になります。逆に変数の後につけると、式の返す値が評価前の値になります。

例えば、++a なら、a = a + 1 を評価し、式の返す値は a+1 になります。しかし、a++ では、a = a + 1 を評価しますが、式の返す値は a になるんです。

ややこしいかもしれませんが、そのように動きます。

文字列

ここでは、文字列と、文字列の操作について説明します。

文字列の書き方と特殊な文字

文字列は、ダブルクオーテーション(")かクオーテーションで囲みます。

```
"shun";
'shun';
```

また、改行などの特殊な文字を表す時には、バックスラッシュ(¥)を先頭につけた特殊文字を使います。

```
"¥n"; // 改行文字
```

ここで、いくつかよく使う特殊文字をピックアップしておきます。

```
"¥n"; // 改行文字
"¥t"; // タブ
"¥""; // ダブルクオーテーション
'¥'' // シングルクオーテーション
"¥¥" // バックスラッシュ
```

¥が続くメタ文字を文字列化する

文字列の長さ

文字列の長さを取得するには、文字列の後に「.length」をつけます。

```
'shun'.length; // => 4
```

Js ではドットの役割が php と違う

「 ~の」 の意味

文字列の結合

JavaScript では複数の文字列を、結合することができます。

```
"sh" + "un"; // => "shun"
```

文字列の分割

結合だけでなく、分割も行うことができます。ただし、区切り文字が必要です。文字列の後に「.split()」と書き、括弧の中に区切り文字の文字列を書きます。すると、その区切り文字で文字列が分割されるのです。

```
"a,b,c".split(","); // => ["a", "b", "c"]
```

このとき、分割された結果は配列というものになるのですが、これについては配列の節で説明します。

真偽値

真偽値は、正しいか、間違っているかの true/false を表すものです。

```
true;
false;
```

この真偽値は、比較演算の結果として使われます。

比較演算

比較演算は、2 つの値の関係を比較し、正しければ true を、間違っていれば false を返します。

```
var a = 1;
var b = 2;
a == b; // => false (a と  b は等しい)
```

```
a === b; // => false (a と b は厳密に等しい)
a != b; // => true (a と b は等しくない)
a !== b; // => true (a と b は厳密に等しくない)
a > b; // => false (a は b より大きい)
a >= b; // => false (a は b より大きいまたは等しい)
a < b; // => true (a は b より小さい)
a <= b; // => true (a は b より小さいまたは等しい)
```

ここで言う「厳密に」というのは、型を変えずに比較するという意味ですが、詳しくは JavaScript の型の章で説明します。

データ構造

ここまでで、数値や文字列、真偽値を使えるようになりましたが、それらを組み合わせたり、複数個の数値を一緒に持っていたかったり、といった使い方をしたくなります。そのために、JavaScript では便利なデータ構造が用意されています。

配列

配列は、いくつかの値を一緒にまとめておくための構造です。以下のように書きます。

```
var a = [8, 17, 13];
```

このようにしたときに、配列は下の図のような状態になっています。数値が、このようなコンテナに格納されているイメージです。

この状態で、0 番目に入っている数値を取り出そうとすると、次のようになります。

```
var a = [8, 17, 13];
a[0]; // => 8 (0 番目の数値)
```

また、配列は数値だけでなく、文字列や真偽値など、他のものでも格納できます。

さらに、0 番目の格納されている数値を変更してしまうこともできます。

```
var a = [8, 17, 13];
a[0] = 3; // このときの配列 a は [3, 17, 13]
```

オブジェクト

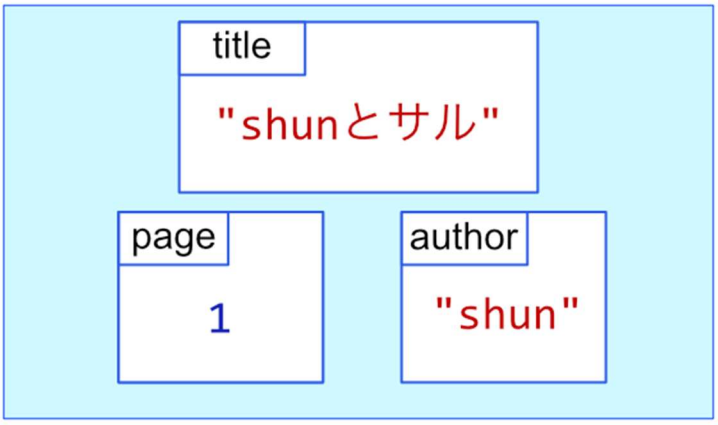
配列では、番号でアクセスしましたが、今度は**名前でアクセスできるオブジェクト**です。

そもそも、オブジェクトというのが何かですが、**その名の通り「もの」です**。例えば、本であれば、タイトル、ページ数、著者など、様々な情報を持っていると思います。**それの情報が組み合わさって、本というひとつの「もの」ができているわけです**。ここでも同じように、それらの情報をプロパティとしてオブジェクトに登録しておくことで、現実世界と同様に扱えます。

実際に書くと以下のような感じになります。

```
var textbook = { title: "shun とサル", page: 1, author:"shun" }; // 1 行でも書けるし、
textbook = { //複数行でも書ける
            title: "shun とサル",
            page: 1,
            author:"shun"
          };
```

これは、先ほどの配列とは違って、順番にとらわれない構造となり、名前で情報を取り出すことができる構造です。



ここから、実際に情報を取り出すときは次のように書きます。

```
var textbook = { title: "shun とサル", page: 1, author:"shun" };
textbook.title; // => "shun とサル" (.の先に要素の名前を書く)
textbook["page"]; // => 1 ([]の中に文字列で要素の名前を書く)
```

要素の変更・追加もできます。

```
var textbook = { title: "shun とサル", page: 1, author:"shun" };
textbook.title = "shun とサルは仲良し" // 要素の変更
textbook.price = 10; // 要素の追加
```

条件文

JavaScript のプログラムは、基本的に上から順番にすべて実行していきます。しかし、**条件によって実行のフローを変えたいことがでてきますよね**。そのための条件文です。

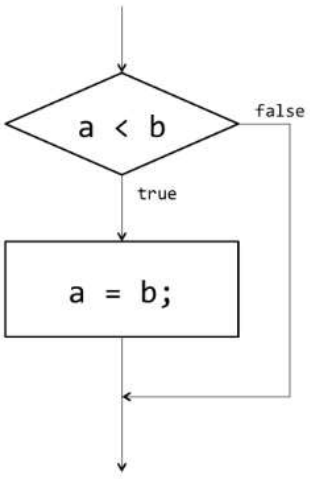
if 文・if-else 文

if 文は、条件式（比較演算などの true/false を返すもの）の結果に応じて実行のフローを変えます。このように言うと話が難しく聞こえますが、例えば僕たちも、授業に行ったときに学生なら授業を聞く、先生なら授業で教える、といったように条件によって行動が変わります。それと同じです。

if 文は以下のように書きます。

```
var a = 1;
var b = 2;
if(a < b) { // a < b なら中身を実行
  a = b;
}
```

このときの実行フローは、次のような形になります。



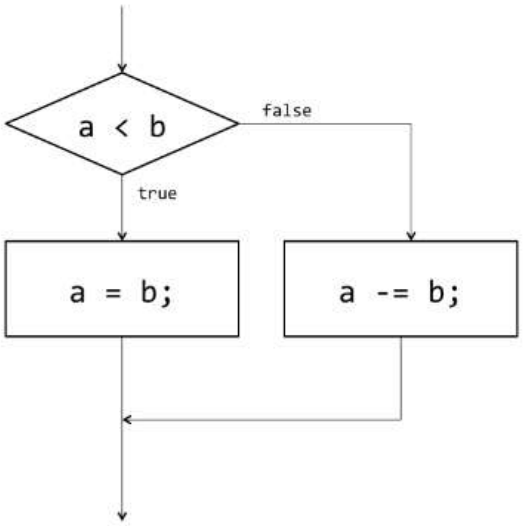
このように、条件によって実行フローを変えるのです。

では、条件に合わなかった場合の書き方も覚えておきましょう。以下のように書きます。

```
var a = 1;
var b = 2;
if(a < b) { // a < b なら中身を実行
  a = b;
} else { // a < b ではないなら中身を実行
```

```
  a -= b;
}
```

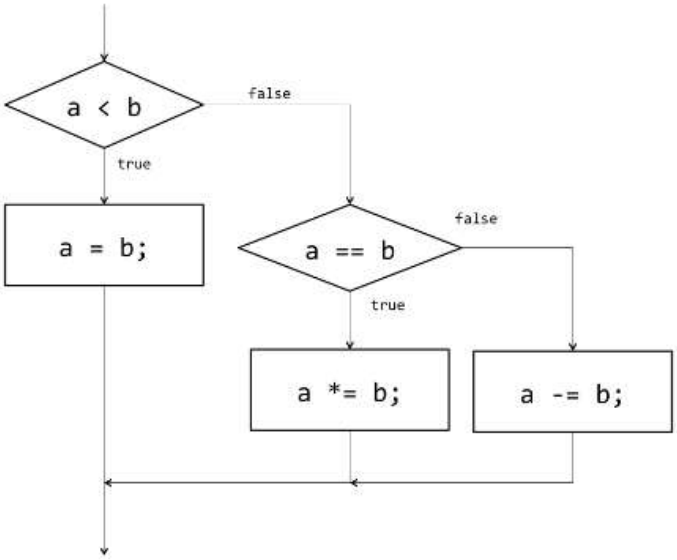
このように書くことで、条件に合わなかった場合のフローが追加されます。



また、さらに条件に合わなかったとき用に、新しく条件を付け足したいときには、以下のようにします。

```
var a = 1;
var b = 2;
if(a < b) { // a < b なら中身を実行
  a = b;
} else if(a == b) { // a < b でないかつ a == b であるとき中身を実行
  a *= b;
} else { // a < b でないかつ、 a == b でないとき中身を実行
  a -= b;
}
```

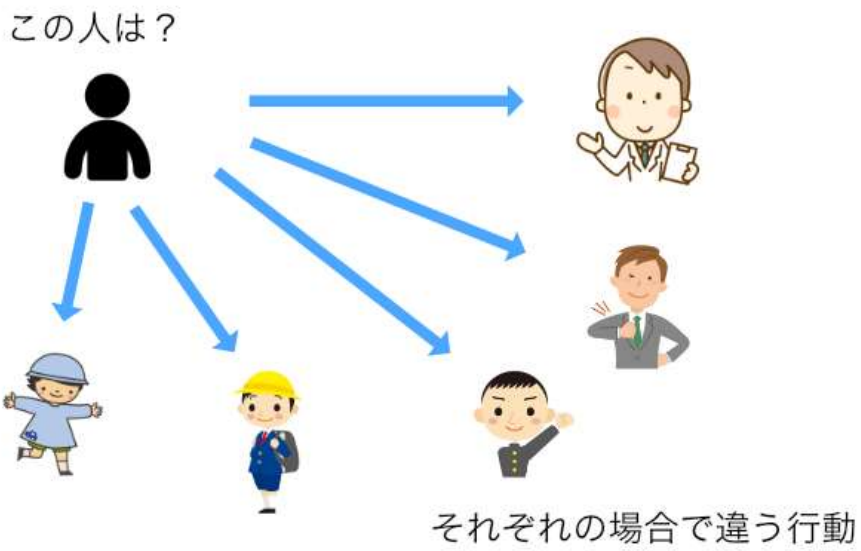
こうしたときには、さらにフローが細かく分かれます。



switch-case 文

条件文には、もう 1 つ switch-case 文というものがあります。switch-case は、より多くの場合分けをするときに便利です。

if 文のときには、分岐をすべて辿りながら進んでいましたが、この switch-case 文では、**直接それぞれの場合に飛ぶことができます**。それによって、それぞれの場合毎に実行される内容が変えるわけです。



実際にコードにすると以下のようになります。

```
var a = 2;
switch(a) { // a の値が
  case 0: // 0 であるとき実行
    a++;
    break;
  case 1: // 1 であるとき実行
    a--;
    break;
  case 2: // 2 であるとき実行
    a *= 2;
    break;
  default: // どれにも当てはまらなければ実行
    a = 0;
    break;
}
```

このとき注意してほしいのは、break をきちんとつけるということです。break はそれぞれの case の終わりにつけます。そうすると、その case が終わったときに switch の中括弧の外まで飛んでくれます。もし case がなかったら次の case まで実行してしまうので注意です。

繰り返し文

繰り返し文は、**ある処理を繰り返し行いたいときに使うものです。**例えば、1 から 10 までの数値を全て足した値を求めたいなどは、繰り返しが使えますね。ここでは、この繰り返し文の中で基本的な性質を持つ for 文と while 文について説明します。

for 文

for 文では指定した範囲で繰り返しを行うことができます。書き方は次のような感じです。

```
var sum = 0;
for(var i = 0; i < 10; i++) { // i が 0 から 9 までの間の繰り返し（毎回最後に i++ を実行）
```

```
  sum += i;
}
```

このときは、i が 0 から 10 未満の範囲での繰り返しになっています。それぞれ以下のような意味です。

最初の var i = 0 は最初のみ実行される条件の初期化

真ん中の i < 10 は繰り返しを終える条件

最後にある i++は毎回最後に実行される式

さらに、JavaScript の for 文には色々な書き方があり、少々挙動も複雑です。以下の記事では、それらをまとめていますので、ぜひ確認するようにしてください。

- JavaScript の色々な for 文 | for・for-each・for-in・for-of

while 文

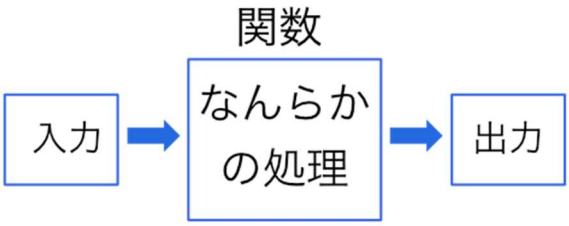
while 文は、条件を満たしている間だけ繰り返すための構文です。次のように書きます。

```
var i = 5;
var sum = 0;
while(i > 0) { // i > 0 の間繰り返す
  sum += i;
  i--;
}
```

while の後の括弧の中にある条件が、true である間繰り返し続けます。

関数

関数は、ある値が入力されたときに、何かしらの処理をして値を返すための命令の集まりです。



今まで紹介してきたようなものを組み合わせてプログラムを書くときに、**よく使うコードはひとまとめにして使いまわせるようにしよう**、というような目的で使います。

例えば、0 から n までの数を足した結果を出力する、といったことも毎回計算するのは面倒なので、関数にしておきたいと思ったとします。そのときには、以下のよう関数を定義します。

```
function sum(n) {
  var sum = 0;
  for(var i = 0; i <= n; i++) {
    sum += i;
  }
  return sum; // 関数が返す値
}

sum(3); => 6
```

function が関数を宣言するときのキーワードになっており、その次の sum というのが、この関数の**関数名**です。関数名は、この関数を後で使う時の名前として使います（最後の行で関数を使っている）。括弧の中身は**引数**といって、入力に当たるものです。そして処理をした結果、何を出力するのかというと、return の中に書かれた値です。この関数が出力する値を**返り値**と言います。

実際に、最後の行では sum という関数に対して、3 を入力して使っています。その結果、0 から 3 までの値が足されて 6 が出力されるわけです。

このようにして、複雑な処理などをあらかじめ関数として用意しておくことで、プログラムを書く量も減り、見やすく便利になります。

JavaScript の型（動的型付け）

型とは

さて、JavaScript の特徴的な性質である、動的型付けについて説明したいのですが、その前にまず型とは何かということをざっくりと説明しますね。

ここまで、数値や文字列といったデータを特に意識せずに使ってきました。では、それぞれの種類はどのように分類されているのか。**その分類が「型」というものです。**数値ならば、数値型が付けられ、文字列なら文字列型が付けられるようになっています。

このようにすることで、プログラムを正常に動作させることができます。

動的型付け

動的型付けでは、型をプログラムを実行しながら決めていきます。ざっくりと言ってしまえば、僕たちがプログラムに型を書かなくていいということです*2。

また、JavaScript では、同じ変数でも実行している間に型が変わっていきます。これについて、型を取得できる「typeof」という機能が JavaScript にはあるので、これを使って見ていきましょう。

```
var a = 1;
typeof a; // => "number" (数値型)
a = "name";
typeof a; // => "string" (文字列型)
a = true;
typeof a; // => "boolean" (ブーリアン型)
```

まず、最初の「var a = 1;」のときには、数値を代入しています。これが実行された後には、a は数値型になっています。そして、その次の「a = ″name″;」では、正しく″name″という文字列が代入されます。この後で a の型は文字列型に変わっています。

このように、JavaScript では、実行している間に型の評価が変わっていくわけです。そのため、その時点でどの型になっているかには注意する必要があります。

演算やその結果の型に注意

JavaScript の演算には、型を変更してしまうようなものがあります。例えば、以下のようなものです。

```
var a = 1;
typeof a; // => "number" (数値型)
a = a + "a"; // => "1a"
typeof a; // => "string" (文字列型)
```

JavaScript では、数値と文字列を足すことができてしまいます。ここでは、数値が文字列に変換されて、その上で結合されます。このような動きには注意したいですね。

また、比較演算について、型の説明をしますね。比較演算には、「厳密な比較」と「厳密でない比較」があるという話をしました。それは、型について厳密かどうか、という意味なのです。以下のコードを見ていきましょう。

```
12 == 12; // => true
12 === 12; // => true
12 == "12"; // => true （！？）
12 === "12"; // => false
```

まず、最初の 2 つは問題ないと思います。同じ数値同士の比較ですので、true が返って当然です。

ですが、3 つ目はどうでしょう。**数値と文字列を比較しているのに、true が返っています。**これが、型に対して厳密でない演算なのです。型について厳密でない演算では、文字列であっても数値に変換して比較を行ってしまうんです。

一方で、**4 つ目の型について厳密な演算では、型の変換は起こらずに比較をしますので、false という結果が返っていますね。**基本は厳密な比較の方が安全ですので、こちらを使うことが多いです。

第一級関数

JavaScript は第一級関数という性質を持っています。これは、次のような性質を持つということです。

- 他の値と同様に名前無しで扱える
- 変数に代入できる
- 配列に格納できる
- オブジェクトのプロパティにできる
- 関数の引数にできる
- 関数の返り値にできる
- プロパティが持てる

それぞれどのように使うか見ていきましょう。

他の値と同様に名前無しで扱える

JavaScript の関数は、他の数値や文字列と同じように扱うことができます（一種のオブジェクト）。つまり名前がなくても関数を作れるということです。

例えば、このようにして書けます。

```
// 関数を変数に代入 ファイルを作って試す
var inc = function (x) {
  return x + 1;
};
```

```
inc(3); // => 4
```

これは、変数に代入した例です。もちろん、変数に代入せずにそのまま使うこともできます。

```
(function(x) { return x + 1; })(1); // => 2
```

これ以外にも配列に格納したり、オブジェクトのプロパティにしたりできます。

配列に格納するときは、以下のように書きます。

```
// 配列に格納
var funcList = [
  function() { return 1; },
  function() { return 2; },
  function() { return 3; }
];

funcList[1](); // => 2
```

オブジェクトのプロパティにするときは、以下のように書きます。

```
// オブジェクトのプロパティに設定
var obj = { func: function() { return 1; } };

console.log( obj.func()); // => 1
```

関数の引数・返り値にできる

関数は、他の関数の引数や返り値にも使えます。数値などと同じ扱いですからね。

まず、引数に使うときは次のような感じに書けます。

```
// 関数を引数として使う
function func(callback) {
  return callback();
}

func(function (){ return 1; }); // => 1
```

ちょっとわかりにくいですが、func という関数の引数として、名前の無い関数を使っています。

次は関数を、他の関数の返り値として使うときの書き方です。

```
// 関数を返り値として使う
function func() {
  return function() { return 1; };
}

func()(); // => 1
```

func という関数の返り値として、関数が返ってくるのでそれを実行しています。

プロパティが持てる

関数はオブジェクトとして扱うことができますので、プロパティを持たせることができます。

次のような感じです。

```
// 関数にプロパティを持たせる
var func = function() { return 1; };
```

```
func.retType = "number";
```

JavaScript におけるオブジェクト指向

さて、これが最後の章となります。オブジェクト指向についてです。ここでは、**基本的なオブジェクト指向の概念と、JavaScript でどのようにオブジェクト指向プログラミングを行うのか、ということの導入部分について説明します**。

オブジェクト指向とは

オブジェクト指向とは、現実世界で起こることを**抽象化**して、プログラムに落とし込むための技法です。

高校生を生成するコードを書いた場合

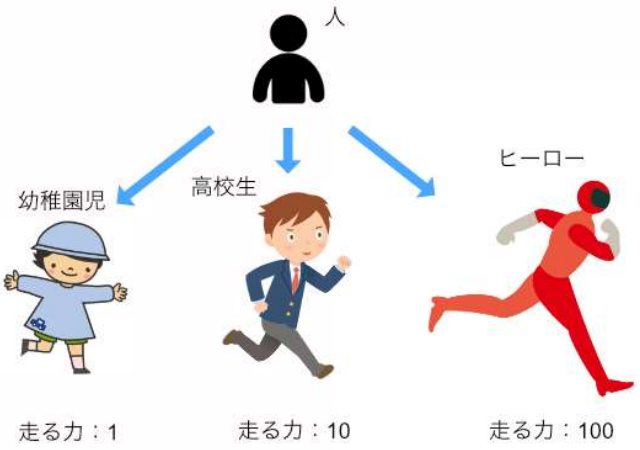
幼稚園児を作る場合

これらのコードは、違う部分がわずかしか無い

ただし、全く同じではない。

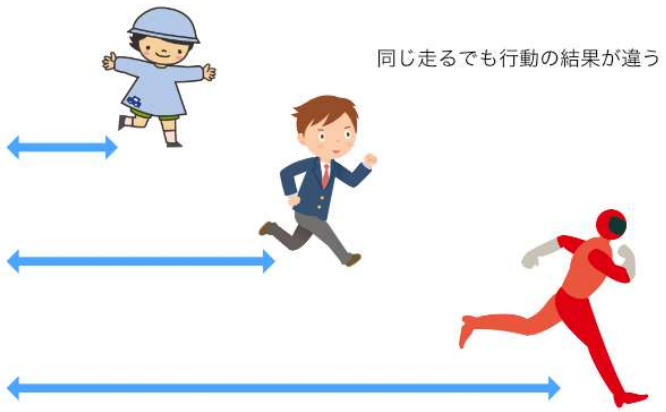
→ 違う部分だけ切り離すか、上書きするほうが効率

よく書ける



例えば、幼稚園児も、高校生も、ヒーローも同じ「人」です。ですから、**「人」という集まりの一部分とみなすことができます**。

唐突にはなりますが、人は「走る」という行動を行うことができますよね。そこで、彼らに走るという行動をさせてみましょう。



このようにすると同じ走るでも、その能力が違いますよね。

ここで話したようなことを、うまく抽象化してモデルに落とし込み、プログラムするのがオブジェクト指向です。

プロトタイプベースのオブジェクト指向

さて、ここまで説明をしてきた内容を実際に

まず、書き方です。

コピペで新規ファイルに書いてみましょう

```
var Human = function(speed) { // オブジェクトの設計
  this.pos = 0; // 位置
  this.speed = speed; // 走る速度
};
Human.prototype.run = function() { this.pos += this.speed; return this.pos; }; // 走るという機能の定義 prototype は継承するという意味

var kindergartener = new Human(1); // 幼稚園児オブジェクトを作る
var highSchoolStudent = new Human(10); // 高校生オブジェクトを作る
var hero = new Human(100); // ヒーローオブジェクトを作る
```

```
kindergartener.run(); // => 1
highSchoolStudent.run(); // => 10
hero.run(); // => 100
```

このような感じです。まず、オブジェクトの設計のところでは、無名関数のような書き方をします。ここでいう this は、この設計から実際にオブジェクトを作ったときの、自分自身にあたるものです。

このとき、run という関数を Human というオブジェクトに付けておきます*3。ここで、**prototype というのは共通の機能（値）を定義するためのものです**。run は、走る能力に関わらず、行う内容は共通しているので、prototype として定義します。

これで設計ができたので、次は**実際にオブジェクトを作る部分の書き方**です。 new というキーワードを最初に付けた関数呼び出しが、オブジェクトを作る部分になります。これで、それぞれのオブジェクトを作ります。

そして、実際に共通の run という機能を使うと、得られる結果がそれぞれによって違うというわけです。

JavaScript における、基本的なオブジェクト指向の使い方はこのような感じです。（実際には prototype チェーンなどの大切な概念があるのですが、結構複雑で説明が長くなるのと、実際の開発でしようされることが少ないので、またの機会に解説しようと思います。*4）

JavaScript を勉強する上での注意！

ここから先では、JavaScript の使い方や基本的な書き方、特徴的な概念について説明していきます。しかし、その中で分かりにくいものも出てくるかもしれません。ですが、その一つ一つで悩んでいては、身につけるのに時間がかかってしまいます。

ですから、もしわからないことがあれば、誰かに聞くようにしてください。**聞く相手がいない時は、プログラミングの Q&A サイトを使うのがいいです**。特に以下の記事で紹介しているサイトでは、現場で開発をしているプログラマーに直接聞くこともできます。ぜひ利用してみてください。