

リスキリング

JavaScript 第2回 / 全6回

関数に分離してみましょう。

```
// 関数を定義
function display() {
    var d = new Date();
    var h = d.getHours();
    var m = d.getMinutes();
    var s = d.getSeconds();
    document.writeln(h + "時" + m + "分" + s + "秒");
}

display(); // 関数を実行
```

時計を更新してみましょう。

```
// 関数を定義
function display() {
    var d = new Date();
    var h = d.getHours();
    var m = d.getMinutes();
    var s = d.getSeconds();
    document.writeln(h + "時" + m + "分" + s + "秒");
}

setInterval(display, 1000); // 1000ミリ秒毎に関数を実行
```

変数

- 値（数値や文字列等）を入れる入れ物
- 自由に名前（変数名）を付けることが出来る

```
var name = "Tam";    // 名前は Tam  
var age = 17;        // 年齢は 17歳  
let address = "高松市"; // 住所は高松
```

※var と let どちらも変数を定義できますが、できる限り let を使いましょう。

定数

値の変わることのない変数

```
const adult_age = 18;           // 成人年齢は 18歳  
const country = "日本";         // 国籍は日本  
const sales_tax_rate = 0.10;    // 消費税率は 10%
```

定数をうまく利用すると、成人年齢が引き下げられたときや、消費税率が変更されたときなどに、必要最低限の変更で対応することが出来るようになる。

スコープ1

以下を実行すると、何が表示されるでしょうか？

```
let a = 1;

function test() {
  let a = 2;
}

test();
console.log(a);
```

スコープ2

以下を実行すると、何が表示されるでしょうか？

```
function test() {  
  let a = 1;  
  {  
    console.log(a);  
    let a = 2;  
    console.log(a);  
  }  
  console.log(a);  
}  
  
test();
```

スコープ3

以下を実行すると、何が表示されるでしょうか？

```
function test() {  
  var a = 1;  
  {  
    var a = 2;  
  }  
  console.log(a);  
}  
  
test();
```


算術演算子

- 加算： +
- 減算： -
- 乗算： *
- 除算： /
- 余算： %

```
var a = 2 + 3 * 4;  
console.log(a);  
var b = 5 - 6 / 3 + 7;  
console.log(b);  
var c = 10 % 3;  
console.log(c);
```

代入演算子

```
var a = 2;  
a = a + 3;  
a += 3; // a = a + 3 の省略形  
  
var b = 10;  
b = b * 10;  
b *= 10; // b = b * 10 と省略形
```

代入演算子 2

インクリメント： `a = a + 1` の部分は `a++` や `++a` とも記述できます。

デクリメント： `a = a - 1` の部分は `a--` や `--a` とも記述できます。

```
var a = 1;
a++;      // a = a + 1;
console.log(a);
a--;      // a = a - 1;
console.log(a);
console.log(a++);    // 表示した後にインクリメント
console.log(++a);    // 表示する前にインクリメント
```

文字列連結演算子

```
var str = "香川" + "県";  
console.log(str);  
var loc = str + "高松市";  
console.log(loc);  
var name = "Tam";  
var age = 17;  
var msg = name + "さんは" + age + "歳"; // 数値は自動で文字列に変換されて結合  
console.log(msg);
```

比較演算子

比較演算子	意味
==	左右が等しければ true 、それ以外は false
>	左が右より大きければ true 、それ以外は false
<	左が右より小さければ true 、それ以外は false
>=	左が右以上のとき true 、それ以外は false
<=	左が右以下のとき true 、それ以外は false
!=	左右が等しくなければ true 、等しいとき false
===	左右の「値」と「型」がどちらも一致すれば true 、それ以外は false

※注意： `=>` や `=<` といった比較演算子は間違いです。

論理演算子

基本

- true (真) または false (偽)

論理演算子

- AND 「&&」 : 左右の両方が true のとき、全体を true とする
- OR 「||」 : 左右のどちらかまたは両方が true のとき、全体を true とする
- NOT 「!」 : 「!」 の後の式の論理を反転する

```
var x = 1;
var y = 1;
var result1 = (x == 1) && (y == 2);
console.log(result1);
var result2 = (x == 1) || (y == 2);
console.log(result2);
var result3 = !((x == 1) && (y == 2));
console.log(result3);
```

配列変数

- 変数が列になったもの
- 複数の値を一括で扱える
- 「変数名[添字]」で指定
- 添字は定数でも変数でも良い

```
var students = ["tanaka", "sato", "suzuki"];  
console.log(students[0]);  
var index = 2;  
console.log(students[index]);
```

条件分岐 1

```
if (式) {  
    // 式が true ならここ  
}
```

```
if (式) {  
    // 式が true ならここ  
} else {  
    // 式が false ならここ  
}
```


条件分岐 2

```
if (式1) {  
    // 式1が true ならここ  
} else if (式2) {  
    // 式1が false かつ式2が true ならここ  
} else if (式3) {  
    // 式1, 式2が false かつ式3が true ならここ  
} else {  
    // 式1, 式2, 式3が false ならここ  
}
```

switch 文

```
switch (a) {  
  case 1:  
    console.log("1です。");  
    break;  
  case 2:  
    console.log("2です。");  
    break;  
  case 3:  
    console.log("さあーん!");  
    break;  
  default:  
    console.log("それ以外です。");  
    break;  
}
```

while 文

```
while (式) {  
    // 式が true の間、ここを実行  
}
```

```
do {  
    // 文  
} while (式); // 式が true なら、もう一度文を実行
```

for 文

```
for (初期化式1; 条件式; 繰返式) {  
    // ここを実行  
}
```

```
for (var i = 0; i < 10; i++) {  
    console.log(i);  
}
```

break 文

for文, while文 などのループ内で使用。

break が実行されるとループを脱出

```
for (var i = 0; i < 100; i++) {  
    console.log(i);  
    if (i > 5) {  
        break;  
    }  
}
```

continue 文

for文, while文 などのループ内で使用。

continue が実行されると、すぐさま次のループを実行

```
for (var i = 0; i < 100; i++) {  
    if (i <= 95) {  
        continue;  
    }  
    console.log(i);  
}
```

配列と繰り返し

```
var list = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37];
for (var i = 0; i < list.length; i++) {
  console.log(i + "番目の素数は" + list[i] + "です。");
  var str = String(list[i]); // 数値型を文字列型に変換
  if (str.indexOf("3") !== -1) { // "3" が str に存在しなければ -1 を返す
    console.log(str + "は 3 を含みます。");
  }
}
```

作ってみよう

お題

1. 1 秒に 1 つずつ、1 から数字をインクリメントしながら表示する。
2. 3 の倍数のときにだけ「！」を付けて表示する。
3. 3 の倍数のとき、もしくは "3" を含む数字のときだけ「！」を付けて表示する。

休憩

ストレッチして、すこし体を動かしましょう。

動かしてみよう

3の倍数、もしくは"3"を含む数字のときだけ派手に表示

```
<html>
<head>
  <script type="text/javascript" src="index0100.js"></script>
</head>
<body>
  <p id="number">0</p>
</body>
</html>
```

```
const countmax = 100;  
var count = 1;  
var timer = setInterval(function() {  
    var msg = String(count);  
    if (count % 3 === 0 ||  
        String(count).indexOf("3") !== -1) {  
        msg = "<font color='red' size=30>" + count + "</font>";  
    }  
    document.getElementById("number").innerHTML = msg;  
    if (count >= countmax) {  
        clearInterval(timer);  
    }  
    count++;  
}, 1000);
```

関数

- 同じ処理を何度も書きたくない場合に利用
- 可読性（読みやすさ）のために利用

```
function 関数名(引数リスト) {  
    // ここに実行したい文を書く  
    return 値;  
}
```

変数型

- JavaScript の処理系がほぼ自動で処理してくれる
- 変数には「型情報」と「値」が格納されている

```
var a;  
a = 1;  
console.log(a);  
a = "こんばんは。";  
console.log(a);  
a = true;  
console.log(a);
```

for in 文

繰り返し for 文の派生系

```
var obj = {a:1, b:2, c:3};  
for (var k in obj) {  
    console.log(k);  
    console.log(obj[k]);  
}
```

エラーハンドリング

```
try {  
    var a = 5 / 0; // 0 で割っているのでエラーが起こる  
    a *= 5;  
    console.log(a);  
} catch (err) {  
    console.log("エラーが起きました。");  
}
```

やってみよう

```
var nums = [56, 78, 83, 64, 100, 87, 98, 43, 95, 83, 60, 74, 36];
```

上の数値配列について、以下のプログラムを書いてください。

1. 各値を並べられた順番に表示する
2. 平均を算出する
3. 値の大きい方から順番に表示する