ななちゃんのIT教室

配列列操作に挑戦の巻

by nara.yasuhiro@gmail.com

ななちゃんが
配列列操作の世界を探るという お話

第 0.1 版　2017 年 6 月 6 日

フリー素材
http://freeillustration.net

いらすとやフリー素材
http://www.irasutoya.com/

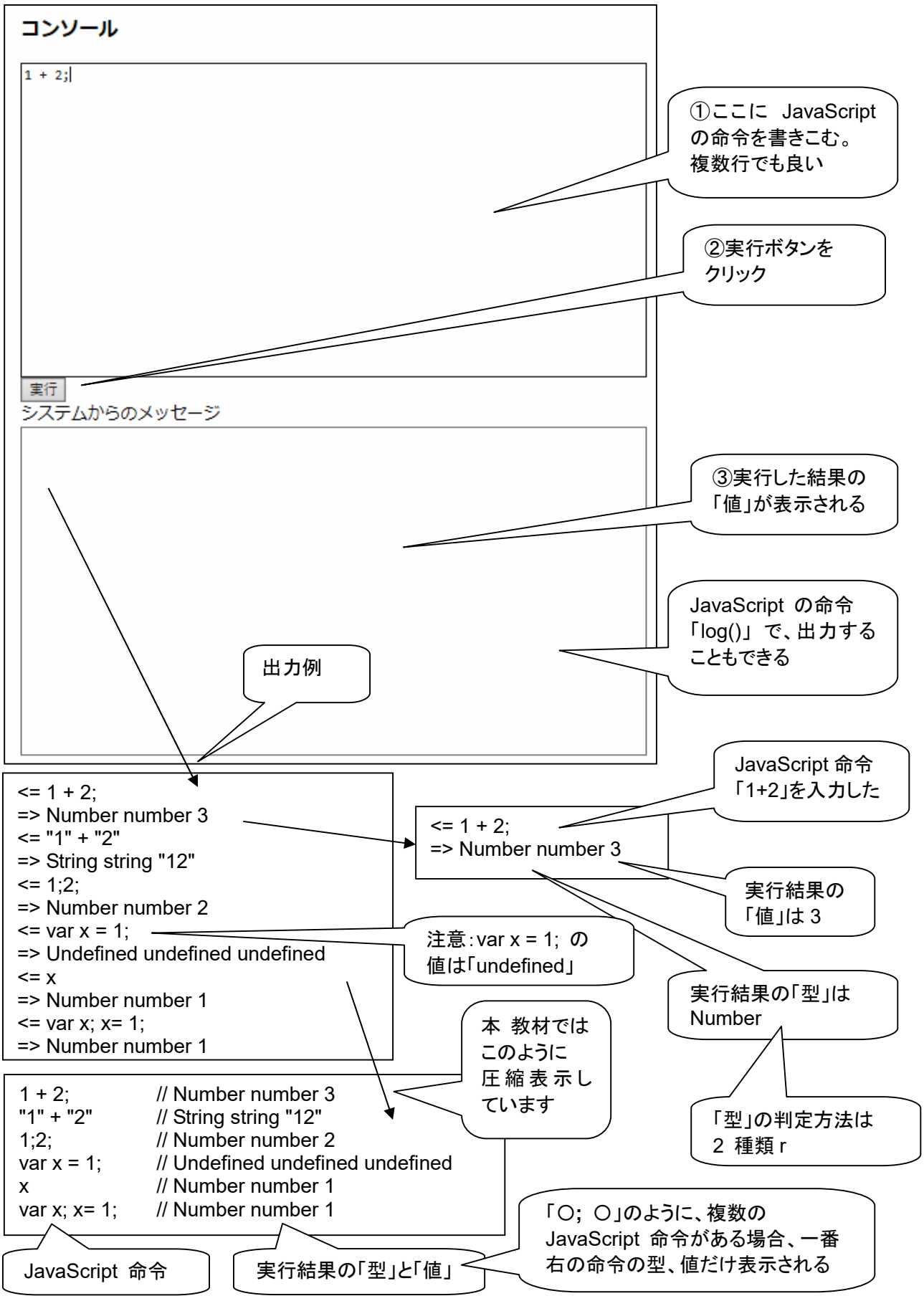
もくじ

第１回　秘密道具：マイコンソール
第２回　配列の基礎
第３回　元の配列を変化させない、複製的操作
第４回　元の配列を変化させる、万能操作
第５回　元の配列の両端の挿入、削除操作
第６回　ちょっと便利な操作
第７回　型付き配列
第８回　多次元配列

第1回　秘密道具：マイ・コンソール

なな：クリじい、配列操作の勉強をするんだけど、便利な秘密道具はない？

クリ：あるぞ、あるぞ。「マイ・コンソール」。



```
<!DOCTYPE html>
<html>
 <head>
    <meta charset="utf-8">
    <title>コンソール</title>
 </head>
 <body>
    <h3>コンソール</h3>
    <textarea rows="19" cols="80" id=pg autofocus>1 + 2;</textarea>
    <br><input type=button onClick=go() value="実行">
    <br>システムからのメッセージ
    <br><textarea rows="20" cols="80" id=log></textarea>
    <script>
var geval = eval;

var logp = document.getElementById("log");
var pgp  = document.getElementById("pg");
var logd;

function clog(s) { logp.value += s; }

function log(s) { logd += s; }

function typeIs(obj) {
      return(Object.prototype.toString.call(obj).slice(8, -1)); }

function isPrimitive(x) {
    return (typeof x)!="object";
}

function toLiteral(x) {
    return JSON.stringify(x);
}

function type(x) { return "" + (typeof x); }

function isInteger(n) { return n%1 === 0; }

function keys(obj) { return Object.keys(obj); }

function go() {
 logd = "";
 try {
            var v = geval(pgp.value);
            clog("<= " + pgp.value + "\n=> "
               + typeIs(v) + " " + type(v) + " " + toLiteral(v) + "\n");
            pgp.value = "";
            logp.scrollTop = logp.scrollHeight;
            pgp.focus();
 }
 catch(e) { clog(pgp.value + "\n! " + e + "\n");
                  pgp.value = "";
                  logp.scrollTop = logp.scrollHeight;
                  pgp.focus();
 }
 if (logd != "") clog(logd + "\n");
}
   </script>
 </body>
</html>
```

第2回 配列の基礎

なな：配列ってどうやって作るの？　先生：いろいろあるのよ。

```
[1,2,3]                 // Array object [1,2,3]
new Array(1,2,3)        // Array object [1,2,3]
Array(1,2,3)            // Array object [1,2,3]
Array.of(1,2,3)         // Array object [1,2,3]
[ ]                     // Array object []
new Array()             // Array object []
new Array(3)            // Array object [null,null,null]
Array(3)                // Array object [null,null,null]
```

配列 [1,2,3] を作る 3 つの方法

空の配列を作る 2 つの方法

先生：独立して作ったふたつの配列は、内容が同じでも別物。片方の要素を書き換えても互いに無関係。

```
var a; a = [1,2,3]          // Array object [1,2,3]
var b; b = [1,2,3]          // Array object [1,2,3]
a == b                      // Boolean boolean false
(a+"") == (b+"")            // Boolean boolean true
JSON.stringify(a) == JSON.stringify(b)  // Boolean boolean true
b[1] = 9                    // Number number 9
b                           // Array object [1,9,3]
a                           // Array object [1,2,3]
```

別物

内容は同じ

内容書き換えは独立

[1,2,3] ← a
[1,2,3] ← b

先生：配列変数の値コピーは、同一データを指し示すポインタになります。

```
var a; a = [1,2,3]          // Array object [1,2,3]
var b; b = a                // Array object [1,2,3]
a == b                      // Boolean boolean true
b[1] = 9                    // Number number 9
b                           // Array object [1,9,3]
a                           // Array object [1,9,3]
a == b                      // Boolean boolean true
```

同一物

片方を書き換えると相手も変化する

[1,2,3] ← a
　　↖ b

先生：配列の**コピー**（**複写**）には **slice()** を使います。

```
var a; a = [1,2,3]          // Array object [1,2,3]
var b; b = a.slice();       // Array object [1,2,3]
a == b                      // Boolean boolean false
b[1] = 9                    // Number number 9
b                           // Array object [1,9,3]
a                           // Array object [1,2,3]
```

内容は同じだが別物

[1,2,3] ← a
[1,2,3] ← b

先生：length は配列の長さですが、その値を書き換えると長さも変化します。

```
var a; a = [1,2,3]          // Array object [1,2,3]
a.length                    // Number number 3
a[5] = 6                    // Number number 6
a                           // Array object [1,2,3,null,null,6]
a.length                    // Number number 6

var a; a = []               // Array object []
a.length                    // Number number 0
a.length = 3                // Number number 3
a                           // Array object [null,null,null]
a = [1,2,3,4,5]             // Array object [1,2,3,4,5]
a.length = 3                // Number number 3
a                           // Array object [1,2,3]
a.length = 5                // Number number 5
a                           // Array object [1,2,3,null,null]
```

範囲外に書き込み

サイズが増加した

length を増やすと中身も増える

length を減らすと中身も消える

先生：Array は、Object の一種なので、Object としても使えます。でも、混乱するので非推奨。

```
var a; a = [1,2,3]          // Array object [1,2,3]
a[“0”]                      // Number number 1
a[-3] = 9                   // Number number 9
a                           // Array object [1,2,3]
a.length                    // Number number 3
a[-3]                       // Number number 9
a["abc"] = 8                // Number number 8
a                           // Array object [1,2,3]
a.length                    // Number number 3
a["abc"]                    // Number number 8
a.abc                       // Number number 8
a                           // Array object [1,2,3]
```

Array は Object の一種なので 連想配列としても 使える

ややこしくなるので 配列と連想配列 の混在使用は非推奨

先生：Array データを生成するためのメソッドを紹介しましょう。

Array.from()　配列型 (array-like) や反復型 (iterable) オブジェクトから、新しい Array インスタンスを生成。

```
Array.from(new Set([1,3,5,2,4,1,2,3]))  // Array object [1,3,5,2,4]
```

配列←集合←配列

```
var a; a = [1,2,3,4,5]          // Array object [1,2,3,4,5]
var it; it = a.values()         // Array Iterator object {}
var b; b = Array.from(it)       // Array object [1,2,3,4,5]
```

配列←反復←配列

```
var a; a = [1,2,3,4,5]          // Array object [1,2,3,4,5]
var it; it = a[Symbol.iterator]();  // Array Iterator object {}
var b; b = Array.from(it)       // Array object [1,2,3,4,5]
```

配列←反復←配列

```
var s; s = "あいう"             // String string "あいう"
var ar; a = Array.from(s)       // Array object ["あ","い","う"]
```

配列←文字列

Array.of()　可変個の引数から新しい Array インスタンスを生成。

```
var a; a =    Array.of(1,2,3,4,5)  // Array object [1,2,3,4,5]
```

第3回 元の配列を変化させない、複製的操作

先生：元の配列を変化させずに、複写的に加工物（部分配列など）を作る方法を紹介します。

先生：配列の一部分をコピー、抽出するメソッド slice()。全体のコピーにも使えます。

slice([開始位置[, 終了位置]]) :Array
開始位置 Number 開始位置を指定。(0 ～)デフォルトは 0
終了位置 Number 終了位置を指定。(0 ～)デフォルトはすべての要素
指定要素の手前までを抽出。
マイナス値を指定すると「length + 終了位置」の位置。

```
var a; a = ["A","B","C","D","E","F"];     // Array object ["A","B","C","D","E","F"]
a.slice(1, 5);                            // Array object ["B","C","D","E"]
a                                         // Array object ["A","B","C","D","E","F"]
var x; x = a.slice(0,-1);                 // Array object ["A","B","C","D","E"]
var y; y = a.slice(2);                    // Array object ["C","D","E","F"]
var b; b = a.slice();                     // Array object ["A","B","C","D","E","F"]
a == b                                    // Boolean boolean false
```

要素 1～4
先頭～末尾-1
要素 2～末尾
全体複写

先生：ふたつ以上の配列をつないで、ひとつの配列を作るメソッド concat です。材料となる元の配列はそのまま、変化しません。

concat(array0 [, array1] [, ..., arrayN]]) :Array

```
var a; a = ['a', 'b']                     // Array object ["a","b"]
var b; b = ['c', 'd']                     // Array object ["c","d"]
var c; c = a.concat(b)                    // Array object ["a","b","c","d"]
a                                         // Array object ["a","b"]
b                                         // Array object ["c","d"]

var a; a = ["A","B"]                      // Array object ["A","B"]
var b; b = ["C","D"]                      // Array object ["C","D"]
var c; c = ["E","F","G"]                  // Array object ["E","F","G"]
var d; d = ["H"]                          // Array object ["H"]
var e; e = a.concat(b,c,d)                // Array object ["A","B","C","D","E","F","G","H"]
var f; f = a.concat()                     // Array object ["A","B"]
f == a                                    // Boolean boolean false
```

結合
元の配列は無変化
複写
いくつも結合できる

第4回　元の配列を変化させる、万能操作

先生：**splice()** だけで、挿入、削除、置換（削除＋挿入）ができます。元の配列は変化します。

splice(index, howMany, [element1][, ..., elementN]) :Array

index:　　配列を変化させ始める要素の添え字。デフォルトは 0

howMany:配列から**取り除く**古い要素の**数**を示す整数。デフォルトは 全要素

返値　　切り取った部分配列

```
var a; a = ["A","B","C","D","E","F"]     // Array object ["A","B","C","D","E","F"]
a.splice(1,0,"G")                        // Array object []
a                                        // Array object ["A","G","B","C","D","E","F"]
var a; a = ['a', 'b', 'c']               // Array object ["a","b","c"]
a.splice(2, 0, 'A', 'B')                 // Array object []
a                                        // Array object ["a","b","A","B","c"]
var a; a = ["A","B","C","D","E","F"]     // Array object ["A","B","C","D","E","F"]
a.splice(3,2)                            // Array object ["D","E"]
a                                        // Array object ["A","B","C","F"]
var a; a = ["A","B","C","D","E","F"]     // Array object ["A","B","C","D","E","F"]
a.splice(3)                              // Array object ["D","E","F"]
a                                        // Array object ["A","B","C"]
var a; a = ["A","B","C","D","E","F"]     // Array object ["A","B","C","D","E","F"]
a.splice(1,4,"G","H","I")                // Array object ["B","C","D","E"]
a                                        // Array object ["A","G","H","I","F"]
```

挿入

挿入

削除

要素 3～末尾を**削除**

要素 1 から 4 つの要素を削除し、代わりに“G”,”H”,”I” を挿入：→**置換**

第5回 元の配列の両端の挿入、削除操作

先生： 元の配列の、先頭や末尾の削除、挿入は、splice でもできるけど、分かりやすい専用の関数もあります。

先生： Array オブジェクトをスタックとして使用するには、**push()**、**pop()** メソッドを使用します。

shift()、unshift() メソッドでも実現できますが、使用すべきではありません。これらは、配列の先頭を変更するので、実行するたびに、配列のすべての要素に対して、番地の割振り直しが発生します。このため、要素数が多ければ多いほど、処理量が増えていくので注意が必要というか、使用すべきでありません。

<スタックにデータを追加>

```
var stack; stack = new Array()  // Array object []
stack.push(1)                   // Number number 1
stack                           // Array object [1]
stack.push(2); stack            // Array object [1,2]
stack.push(3); stack            // Array object [1,2,3]
```

<スタックからデータを取り出す>

```
var a; a = stack.pop()          // Number number 3
stack                           // Array object [1,2]
var b; b = stack.pop()          // Number number 2
stack                           // Array object [1]
var c; c = stack.pop()          // Number number 1
stack                           // Array object []
```

先生： Array オブジェクトをキュー（FIFO）として使用するには、push()、shift() メソッドの組み合わせ、または、unshift()、pop() メソッドの組み合わせを使います。shift()、unshift() メソッドは、配列の最先頭を変更するので、実行するたびに、配列のすべての要素に対して、番地の割振り直しが発生します。要素数が多ければ多いほど、処理量が増えていくので注意が必要です。やむを得ないというか、少ない要素で使いましょう、ということです。根本解決には、少し複雑なプログラムが必要になります。今回は説明しませんが、巡回バッファなどのテクニックがあります。

<push()、shift() メソッドを使ってキューを実現する場合>

```
var queue; queue = new Array()  // Array object []
queue.push(1)                   // Number number 1
queue                           // Array object [1]
queue.push(2); queue            // Array object [1,2]
queue.push(3); queue            // Array object [1,2,3]
var a; a = queue.shift()        // Number number 1
queue                           // Array object [2,3]
var b; b = queue.shift()        // Number number 2
queue                           // Array object [3]
var c; c = queue.shift()        // Number number 3
queue                           // Array object []
```

<unshift()、pop() メソッドを使ってキューを実現する場合>

```
var queue; queue = new Array()  // Array object []
queue.unshift(1)                // Number number 1
queue                           // Array object [1]
queue.unshift(2); queue         // Array object [2,1]
queue.unshift(3); queue         // Array object [3,2,1]
var a; a = queue.pop()          // Number number 1
var b; b = queue.pop()          // Number number 2
var c; c = queue.pop()          // Number number 3
```

第6回　ちょっと便利な操作

先生：Array データに対して使える、ちょっと便利な操作をいくつか説明します。

先生：reverse() は、要素の順番をひっくり返します。元の配列を変化させます。

```
var a; a = [1,2,3,4,5]          // Array object [1,2,3,4,5]
var b; b = a.reverse()          // Array object [5,4,3,2,1]
a == b                          // Boolean boolean true
```

先生：join() は、配列の全要素を結合して、文字列化します。結合部に使う文字（列）を設定できます。

```
var a; a = [1,2,3,4,5]          // Array object [1,2,3,4,5]
var b; b = a.join()             // String string "1,2,3,4,5"
var c; c = a.join("/")          // String string "1/2/3/4/5"
var c; c = a.join("")           // String string "12345"
```

先生：sort() は、配列の要素を、辞書順、大きさ順などに並べ替えます。元の配列を変化させます。

```
var a; a = [1,2,3,11,12,13]     // Array object [1,2,3,11,12,13]
var b; b = a.sort()             // Array object [1,11,12,13,2,3]
a                               // Array object [1,11,12,13,2,3]
a == b                          // Boolean boolean true
```

デフォルトは辞書順

```
var a; a = [1,11,2,12,3,13]                 // Array object [1,11,2,12,3,13]
a.sort(function(a,b){return a-b;})          // Array object [1,2,3,11,12,13]
a.sort(function(a,b){return b-a;})          // Array object [13,12,11,3,2,1]
```

小さい順

大きい順

```
var a; a = ["abc", "q", "zz", "eeee"]                  // Array object ["abc","q","zz","eeee"]
a.sort(function(a,b){return a.length-b.length;})       // Array object ["q","zz","abc","eeee"]
```

長さ順

先生：indexOf(検索値 [, 検索開始位置]) は、正順検索です。
戻り値は、一致する値が見つかった場合は、その番地、存在しない場合は -1 になります。

```
var a; a = [1,2,3,1,2,3]        // Array object [1,2,3,1,2,3]
a.indexOf(2)                    // Number number 1
a.indexOf(5)                    // Number number -1
a.indexOf(2,3)                  // Number number 4
```

先生：lastIndexOf(検索値 [, 検索開始位置]) は、逆順検索です。
戻り値は、一致する値が見つかった場合は、その番地、存在しない場合は -1 になります。

```
var a; a = [1,2,3,1,2,3]        // Array object [1,2,3,1,2,3]
a.lastIndexOf(2)                // Number number 4
a.lastIndexOf(5)                // Number number -1
a.lastIndexOf(2,3)              // Number number 1
```

先生: forEach(callback [, thisObj]) は、配列の各要素にメソッドを適用します。戻り値は undefined です。
this.Obj は、コールバック関数内で this のアクセス先となるオブジェクトを指定するものです。
callback は、適用するメソッドです。第1引数は要素、第2引数はインデックス、第3引数は対象配列です。

```
var a; a = [1,2,3,4,5]                                  // Array object [1,2,3,4,5]
var s = 0; a.forEach(function(d){s+=d;}); s             // Number number 15
var a; a = [1,3,5,4,2]                                  // Array object [1,3,5,4,2]
var m = 3; a.forEach(function(d){if(d>m)m=d;}); m       // Number number 5
var a; a = [1,2,3,4,5]                                  // Array object [1,2,3,4,5]
a.forEach(function(d,i,ar){ar[i]=2*d;}); a              // Array object [2,4,6,8,10]
var a; a = [1,2,3,4,5]                                  // Array object [1,2,3,4,5]
var b = []; a.forEach(function(d,i){this[i]=d;},b); b   // Array object [1,2,3,4,5]
```

合計 / 最大値 / 要素 2 倍 / 複写

先生: every(callback [, thisObj]) は、各要素にメソッドを適用し、返値の論理値をANDしたものを返します。
メソッドのfalse返値で停止します。

```
var a; a = [1,2,3,4,5]                  // Array object [1,2,3,4,5]
a.every(function(d){return d>0})        // Boolean boolean true
a.every(function(d){return d>2})        // Boolean boolean false
```

先生: some((callback [, thisObj]) は、各要素にメソッドを適用し、返値の論理値をORしたものを返します。
メソッドのrue返値で停止します。

```
var a; a = [1,2,3,4,5]                  // Array object [1,2,3,4,5]
a.some(function(d){return d>4})         // Boolean boolean true
a.some(function(d){return d>5})         // Boolean boolean false
```

先生: filter(callback [, thisObj]) は、メソッドを使用して、新しい配列を生成(データ抽出)して返します。
callback はメソッドで、現在の要素を、結果に含める場合は、true、含めない場合は、false を返します。

```
var a; a = [1,2,3,4,5]                  // Array object [1,2,3,4,5]
a.filter(function(d){return d%2==0})    // Array object [2,4]
```

偶数抽出

先生: map(callback [, thisObj]) は、メソッドを使用して、新しい配列を生成(データの変換)して返します。
callback はメソッドで、この返値を要素として、新しい配列を生成します。

```
var a; a = [1,2,3,4,5]                  // Array object [1,2,3,4,5]
a.map(function(d){return 2*d;})         // Array object
```

要素 2 倍

先生: reduce(callback[, initialValue]) は、メソッドを使用して、値を集約(昇順)します。
reduceRight(callback[, initialValue]) は、メソッドを使用して、値を集約(降順)します。
callback は、適用するメソッドで、第1引数は中間値、第2引数は要素、第3引数はインデックス、第4引数は対象配列です。

```
var a; a = [1,2,3,4,5]                  // Array object [1,2,3,4,5]
a.reduce(function(m,d){return m+d;},0)  // Number
```

合計

先生：ES2015 から導入された操作を紹介します。

先生：copyWithin(書込開始位置 , 読取開始位置[, 読取終了位置]) は、:Array　配列の一部を自分に上書きコピーします。

書込開始位置　負の値を指定した場合、「length + 値」と同等

読取終了位置　デフォルトは最後尾まで

配列のサイズが増えるような書き込みが行われると、その手前で終了。

```
var a; a = [1,2,3,0,0,0,0,0,0,0]          // Array object [1,2,3,0,0,0,0,0,0,0]
a.copyWithin(4 , 0 , 3)                   // Array object [1,2,3,0,1,2,3,0,0,0]
```

先生：fill(指定値 [[, 開始位置] , 終了位置]) は、Array 配列の指定範囲を指定値で上書きします。

開始位置のデフォルトは 0。

終了位置のデフォルトは最後尾まで。

```
var a; a = new Array(7)                   // Array object [null,null,null,null,null,null,null]
a.fill(0)                                 // Array object [0,0,0,0,0,0,0]
a.fill(1,2)                               // Array object [0,0,1,1,1,1,1]
a.fill(2,4,5)                             // Array object [0,0,1,1,2,1,1]
```

先生：includes(検索値 [, 検索開始位置]) は、配列内を昇順に検索し、指定値が含まれるか調べます。

検索開始位置のデフォルトは 0。

```
var a; a = [1,2,3,4,5]                    // Array object [1,2,3,4,5]
a.includes(3)                             // Boolean boolean true
a.includes(9)                             // Boolean boolean false
```

先生：find(callback [, thisObj]) は、メソッド使用して、データを順番に検索（**1つのデータの**インデックスを取得）。

戻り値は、見つかったデータのインデックス。**見つからなかった場合**は、**undefined**

callback はメソッドで、目的のデータが見つかった場合、true 値を返し、その時点で検索終了になります。

```
var a; a = [1,2,3,4,5]                    // Array object [1,2,3,4,5]
a.find(function(d){return d>2;})          // Number number 3
a.find(function(d){return d>7;})          // Undefined undefined undefined
```

先生：findIndex(callback [, thisObj]) は、メソッドを使用して、データを順番に検索（**1つのインデックス**を取得）。

戻り値は、見つかったデータのインデックスです。**見つからなかった場合**は、**-1** になります。

callback はメソッドで、目的のデータが見つかった場合、true 値を返し、その時点で検索終了になります。

```
var a; a = [1,2,3,4,5]                    // Array object [1,2,3,4,5]
a.findIndex(function(d){return d>2;})     // Number number 2
a.findIndex(function(d){return d>7;})     // Number number -1
```

先生： entries( ) は、**Iterator オブジェクトを取得**（**番地と値**の列挙）します。

```
var a; a = [1,2,3]                  // Array object [1,2,3]
var it; it = a.entries()            // Array Iterator object {}
it.next()                           // Object object {"value":[0,1],"done":false}
it.next()                           // Object object {"value":[1,2],"done":false}
it.next()                           // Object object {"value":[2,3],"done":false}
it.next()                           // Object object {"done":true}
```

先生：keys() は、**Iterator オブジェクトを取得**（**番地**の列挙）します。

```
var a; a = [1,2,3]                  // Array object [1,2,3]
var it; it = a.keys()               // Array Iterator object {}
it.next()                           // Object object {"value":0,"done":false}
it.next()                           // Object object {"value":1,"done":false}
it.next()                           // Object object {"value":2,"done":false}
it.next()                           // Object object {"done":true}
```

先生： values() は、**Iterator オブジェクトを取得**（**値**の列挙）します。

```
var a; a = [1,2,3]                  // Array object [1,2,3]
var it; it = a.values()             // Array Iterator object {}
it.next()                           // Object object {"value":1,"done":false}
it.next()                           // Object object {"value":2,"done":false}
it.next()                           // Object object {"value":3,"done":false}
it.next()                           // Object object {"done":true}
```

```
var a; a = [1,2,3]                  // Array object [1,2,3]
var it; it = a[Symbol.iterator]();  // Array Iterator object {}
it.next()                           // Object object {"value":1,"done":false}
it.next()                           // Object object {"value":2,"done":false}
it.next()                           // Object object {"value":3,"done":false}
it.next()                           // Object object {"done":true}
```

第7回　型付き配列

先生：Array の要素には、任意の型のデータを混在させることができます。これに対し、特定の型だけに固定するのが「型付き配列」（Typed Array）です。連続したメモリを効率よく使えるので、アクセスが高速になります。

その代表例が「Uint8Array」です。8 ビット符号なし整数値の配列。他の Typed Array 同様、各要素は0で初期化されます。

生成方法：

```
new Uint8Array(length);
new Uint8Array(typedArray);
new Uint8Array(object);
new Uint8Array(buffer [, byteOffset [, length]]);
```

Typed Ayyay の一覧

型	バイト数	ビット数	説明	Web IDL type
Int8Array	1	8	整数（2の補数）	byte
Uint8Array	1	8	整数（符合なし）	octet
Uint8ClampedArray	1	8	整数（符合なしclamped）	octet
Int16Array	2	16	整数（2の補数）	short
Uint16Array	2	16	整数（符合なし）	unsigned short
Int32Array	4	32	整数（2の補数）	long
Uint32Array	4	32	符号なし整数	unsigned long
Float32Array	4	32	IEEE 浮動小数点数	unrestricted float
Float64Array	8	64	IEEE 浮動小数点数	unrestricted double

Web IDL: Web Interface Definition Language。2017 年 6 月 1 日付で、W3C が、編集者草案（Editor's Draft）として第2版仕様を公開。

clamped：範囲の指定された整数変数に、その範囲を超えた値を代入したとき、通常であればmoduloで処理する（上位ビット棄却）ところを、範囲の上限の値を代わりに代入する、というもの。

先生：ついでに、関連項目の、「ArrayBuffer」を説明しておきます。

一般的な固定長のバイナリデータのバッファを示すために使われるデータタイプ。ArrayBuffer の内容物を直接操作することはできない。バッファの内容物を読み書きするためにはtyped array object や、DataView オブジェクトを作成して使う。

例：new Uint8Array(arrayBuffer);

例：var dataview = new DataView(buffer [, byteOffset [, byteLength]]);
　　dataview.getUint8(byteOffset);

第8回 多次元配列

先生: JavaScript の配列は、各要素に任意の型のデータを記憶できます。そこで、各要素に配列データを記憶するという形で多次元配列を実現できます。

用途は、もともと二次元のデータの処理。画像とか、ボードゲームの盤面とか。変わった例では、掛け算九九の計算結果を二次元配列に格納しておくことで、計算を高速化できます。

「function mat(x,y) { retuen 100*y + x; }」のような関数を作っておき、一次元配列を二次元配列に見せるというテクニックもあります。「data[mat(x,y)] = d;」、「d = data[mat(x,y)];」のような使い方をします。