ななちゃんのIT教室

文字列操作に挑戦の巻

by nara.yasuhiro@gmail.com

ななちゃんが
文字列操作の世界を探るという お話

第 0.2 版　2017 年 6 月 16 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　文字列操作とは
第2回 「文字列変換銃」
第3回 「文字列変換銃」を使ってみる
第4回　正規表現
第5回　文字列の抽出と検索
第6回　文字列と配列の相互変換
第7回　文字列－配列相互変換を使った文字列操作
第8回　メソッドチェーン
第9回　うまい方法
第10回　文字列操作のちょっとしたノウハウ
第11回　いろいろな文字列変換
第12回　その他

第1回　文字列操作とは

なな：文字列操作って何？

先生：文字列の一部分を取り出したり、ルールに従って文字列を書き換えたりすることよ。
たとえば、大文字と小文字の混ざっているテキストを、すべて大文字に変えるとか。

なな：フリーソフトとか、エディタにもそういう機能がありそうだけど

先生：そうね。でも、専用のフリーソフトだと、機能を修正したり、複数のソフトを組み合わせるのが大変だったりするわね。それから、エディタの機能だと、足りない機能があっても機能追加が難しかったりするわね。JavaScriptで実現すれば、修正や組み合わせが自由になるの。それから、そういう知識があると、JavaScript のプログラム作成で重宝するわ。

クリ：わしも、重宝しておるぞ。コンソールや、自分で作ったコンソールプログラムで、変数の値の型を確認するのに、下記のようなのを使っておる。「slice(8, -1)」の部分が文字列操作になっておる。

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

たとえば、数字の 123 も、文字列の “123” も、alert で表示すると「123」と、同じになってしまうが、数字に「+」を適用すると足し算、文字列に適用すると文字列連結になるので、区別しないといけない。そういう場合に、「型判定器！」を使うと便利なんじゃ。

```
var a = 123;
var b = "123";
typeIs(a);        // Number
typeIs(b);        // String

Object.prototype.toString.call(a);        // [object Number]
```

数字なら「Number」、文字列なら「String」という値を返してくる。 slice(8, -1) を入れてないと、「[object Number]」という値になって、ちょっと見にくい。

先生：それから、ユーザに 7 桁の郵便番号を入力してもらった時、「253-0025」のように、真ん中に「-」が入っていたら、それを取り除いてから検索処理に渡したり、全角文字の数字だったら、それを半角の数字に変えるとか、いろいろ使い道があるのよ。

なな：なるほど。勉強しなくちゃ。

第2回 「文字列変換銃」

なな：文字列操作の勉強って、どうすれば良いのかしら？

クリ：秘密道具「文字列変換銃」！

なな：それって、ドラえもん秘密道具の「物体変換銃」のパロディ？ 物体に銃口を向け、元素組成を操作して別の物体に作り変えることのできる銃でしょ。ダイコンに向けて、銃についている指令マイクに「ダイコン マイナス ダイ、プラス ラジ」と言うと、ラジコンに変わる。「スズメ」を「スルメ」に変え、「ネコ」を「ネッコ」に変えてしまう。

クリ：まあ、いいではないか。

「編集コマンド」の初期値は 「out.value=ipt.value」 で、入力データをそのまま出力にコピーする命令になっておる。これを、文字列操作命令に修正してから実行すれば、いろいろな実験ができる。出力欄へのキーボードからの入力、ファイル保存も可能なので、簡易エディタにも使えるというスグレものじゃ。

```
<!DOCTYPE html>
<html><head><meta charset="utf-8">
                    <title>File menu</title>
                    <style> #cmnd { font-size: large; } </style>
        </head><body>
   <h3>String manipulator</h3>
   <input id="fileSel" type="file" multiple="true" onchange="readFile()" size=50>
   <br>入力データ:
   <br><textarea rows=15 cols=50 id=ipt></textarea>
   <br>編集コマンド:
   <br><textarea id=cmnd rows=3 cols=40>out.value=ipt.value</textarea>
        <input type=button onclick=change() value="実行">
   <br>コマンドログ:<br><textarea rows=5 cols=50 id=log></textarea>
   <br>変換結果:<br><textarea rows=15 cols=50 id=out></textarea>
   <br>ファイルタイプ:<input type=text id=type size=10>
   <br>ファイル名:<input type=text id=fname>
        <input type=button value="ファイル保存" onclick=save()><br>
   <script>
var reader, geval = eval;

function printFile(evt) {
    var cntnt = evt.target.result;
    ipt.value = cntnt;
    ipt.scrollTop = ipt.scrollHeight;
}
function readFile() {
    reader = new FileReader();
    reader.onload = printFile;
    var files = document.getElementById("fileSel").files;
    reader.readAsText(files[0]);
    type.value = files[0].type;
    fname.value = files[0].name;
}
function change() {
    log.value = "";
    try { log.value += JSON.stringify(geval(cmnd.value)); }
    catch(e) { log.value += e + ""; }
    out.scrollTop = out.scrollHeight;
    log.scrollTop = log.scrollHeight;
}
function fsave(data,fname,ftype) {
    var blob = new Blob([data], {type: ftype});
    if(window.navigator.msSaveBlob) {
        window.navigator.msSaveBlob(blob, fname); }
    else {
        var a = document.createElement("a");
        a.href = URL.createObjectURL(blob);
        a.target = '_blank';
        a.download = fname;
        var pp=document.body.appendChild(a);
        pp.click();
        pp.parentElement.removeChild(pp); }
}
function save() {
    try { fsave(out.value,fname.value,type.value); }
    catch(e) { log.value += e + ""; }
}
   </script>
  </body>
</html>
```

第3回 「文字列変換銃」を使ってみる

クリ：しばらくは、下記のような入力テキストを、いろいろな命令で変換してみよう。

```
line-1 abcABCabc
line-2  abcABCabc
line-3 This is a program.
line-4 line
dog
cat
ant
いぬ
ねこ
あり
```

先生：out.value=ipt.value.toUpperCase()　　out.value=ipt.value.toLowerCase()

LINE-1 **ABC**ABC**ABC**
LINE-2 **ABC**ABC**ABC**
LINE-3 T**HIS IS A PROGRAM**.
LINE-4 **LINE**
DOG
CAT
ANT
いぬ
ねこ
あり

line-1 abc**abc**abc
line-2 abc**abc**abc
line-3 **t**his is a program.
line-4 line
dog
cat
ant
いぬ
ねこ
あり

小文字を大文字にしたり、大文字を小文字にしたりできます。半角も全角も大丈夫。

先生：out.value=ipt.value.replace("line","LINE")　　out.value=ipt.value.replace(/line/**g**,"LINE")
out.value=ipt.value.replace(/line/,"LINE")

LINE-1 abcABCabc
line-2 abcABCabc
line-3 This is a program.
line-4 line
dog
cat
ant
いぬ
ねこ
あり

LINE-1 abcABCabc
LINE-2 abcABCabc
LINE-3 This is a program.
LINE-4 **LINE**
dog
cat
ant
いぬ
ねこ
あり

半角文字の line を LINE に変換。「g」（グローバルフラグ）をつけるとすべてが対象。フラグがないと、最初の 1 つだけが対象になります。

先生：out.value=ipt.value.replace(/abc/g,"xyz")　out.value=ipt.value.replace(/abc/g**i**,"xyz")

line-1 **xyz**ABC**xyz** line-2 abcABCabc line-3 This is a program. line-4 line dog cat ant いぬ ねこ あり	line-1 xyz**xyz**xyz line-2 abcABCabc line-3 This is a program. line-4 line dog cat ant いぬ ねこ あり

半角の abc を xyz に変換。「i」（case-insensitive 大文字小文字区別なしフラグ）もあります。

先生：out.value=ipt.value.replace(/line$/,"LINE")　out.value=ipt.value.replace(/line$/m,"LINE")

line-1 abcABCabc line-2 abcABCabc line-3 This is a program. line-4 line dog cat ant いぬ ねこ あり	line-1 abcABCabc line-2 abcABCabc line-3 This is a program. line-4 **LINE** dog cat ant いぬ ねこ あり

「$」は行末にマッチング。すべての行でチェックするには「m」（multi-line 複数行フラグ）を使います。これがないと、最後の行の行末だけが対象になってしまいます。

なな：ふんふん。

第4回　正規表現

なな：正規表現って何？　ずいぶん、固いことばね。

先生：正規表現は、英語の regular expression の訳で、いくつかの規則性をもった文字列群を一つの形式でまとめて表現する方法の一つなの。正則表現、正規式と呼ばれることもあるわ。この表現方法を利用すれば、たくさんの文章の中から容易に見つけたい文字列を検索することができるのよ。たとえば、「Windows 10、Windows10、WINDOWS 10、WINDOWS10、Windows　10」などをまとめて表現できれば、ウェブ検索が効率良くおこなえたり、文章中の表記を「Windows 10」に統一するように書き換えたりすることができるの。元になる動詞の regulate は、規則などで取り締まる、統制する、規則正しくするという意味。

なな：それが、JavaScript で利用できるの。どうやって？

先生：正規表現に関する事項をまとめておきましょう。まずは、正規表現の生成。

```
var re = /ab+c/;
var re = new RegExp("ab+c");1
var re = new RegExp("pattern", "flags");
```

先生：次に、正規表現を使う 検索、置換命令。

・match　　文字列を正規表現で検索して、結果を配列で取得

```
"123456".match(“345”)                    // ["345"] これは、正規表現ではない、文字列。
"123456".match(/345/)                    // ["345"] これは、正規表現。意味は上記と同じ。
"123456".match(/3.*5/)                   // ["345"]
"12,34,56,78".match(/(\d*),(\d*),(\d*)/);  // ["12,34,56","12","34","56"]
```

配列の要素 0 は、マッチした部分全体、要素 1 以降は、( ) に対応。 3 つ目の例は、
「0 個以上の数字 + カンマ + 0 個以上の数字 + カンマ + 0 個以上の数字」にマッチ。
「0 個以上の数字」の部分が、切り出されている。

・replace　　文字列を置換

```
"abcdefABCDEFabcdefABCDEF".replace( "ABC" , "XYZ" );
                                          // abcdef**XYZ**DEFabcdefABCDEF
"abcdefABCDEFabcdefABCDEF".replace( /ABCDEF/**g** , "XYZ" );
                                          // abcdef**XYZ**abcdef**XYZ**
```

・search　　文字列を正規表現で検索。

```
"abcdefAB**CDE**F".search( /CDE/ );        // 8
"ab**cde**fABCDEF".search( /CDE/**i** );    // 2
```

・exec

```
var myRe = /d(b+)d/g;
var myArray = myRe.exec("c**dbbd**bsbz");
myRe.lastIndex;          // 5
```

先生：つぎに、正規表現を構成する記号の説明。

^	入力の先頭にマッチ
$	入力の末尾にマッチ
*	直前の文字の 0 回以上の繰り返しにマッチ
+	直前の文字の 1 回以上の繰り返しにマッチ。{1,} と同じ意味
?	直前の文字の 0 回か 1 回の出現にマッチ。{0,1} と同じ意味
.	改行文字以外のどの 1 文字にもマッチ
x|y	'x' または 'y' にマッチ

[xyz]	文字集合。角括弧で囲まれた文字のいずれか1個にマッチ。特殊文字（. や *）はエスケープ不要
[a-z]	ハイフンを用いて文字の範囲を指定することも可能
[^xyz]	角括弧で囲まれた文字ではない文字にマッチ
\d	数字にマッチ。[0-9] に相当
\D	数字以外の文字にマッチ。[^0-9] に相当
\n	改行文字 (U+000A) にマッチ
\r	復帰文字 (U+000D) にマッチ
\s	スペース、タブ、改ページ、改行を含む 1 個のホワイトスペース文字にマッチ
\S	ホワイトスペース以外の 1 文字にマッチ
\t	タブ (U+0009) にマッチ
\w	アンダースコアを含むどの英数字にもマッチ。[A-Za-z0-9_] に相当
\W	前述以外の文字にマッチ
\xhh	hh（2 桁の 16 進数）コードからなる文字列にマッチ
\uhhhh	hhhh（4 桁の 16 進数）コードからなる文字列にマッチ
{n}	n には 0 以上の整数を指定。正確に n 回一致。
{n,}	n には 0 以上の整数を指定。少なくとも n 回一致。
{n,m}	m および n には 0 以上の整数を指定。n は m 以下。n ～ m 回一致。カンマと数の間には、スペースを入れない。
?	他の修飾子 （*, +, ?, {n}, {n,}, {n,m}） の直後に指定すると、一致パターンを制限。 既定のパターンでは、できるだけ多数の文字列と一致するのに対し、できるだけ少ない文字列と一致。文字列 "oooo" に対し、'o+?' は 1 つの "o" と一致し、'o+' はすべての 'o' と一致。 最短マッチ： var url = “http://hoge.com/hoge/fuga/goods/” から、スラッシュで囲まれた最初のブロックの文字列、hoge だけ取り出したい。 url.match(/^http:\/\/hoge.com\/(.*)\//) だと、引っかかるのは「hoge/fuga/goods」。最短マッチのメタ文字を追加。 url.match(/^http:\/\/hoge.com\/(.*?)\//) 任意の1文字の繰り返しを最短で切り上げるため、(.*)に「?」を入れる。すると、期待通りに「hoge」だけ取れる。

先生： つぎは、正規表現の後ろに添えるフラグ

g	グローバルなマッチ。最初のマッチの後に止まることなくすべてのマッチを探す
I	大文字・小文字の無視
m	複数行に渡るマッチ。先頭および終端を示す文字 (^ や $) が、複数の行で機能（すなわち、入力文字列全体の先頭および終端だけでなく、各々の行（\n や \r で区切られる）の先頭および末尾にマッチ
u	Unicode。パターンを Unicode コードポイントの羅列として扱う
y	先頭固定 （sticky）。対象文字列中の正規表現の lastIndex プロパティによって示された位置からのみマッチするようになる（それより後の位置からのマッチは試みない）。

なな： 文法書みたいで、頭が痛くなりそう。

先生： 具体例をあげてみるわね。

bから始まってkで終わる3桁以上の文字列	b.+k
数字列	\d+
大阪府または大阪市	大阪(府|市)
全角数字	[0-9]
全角英大文字列	[A-Z]+
ひらがな	[ぁ-ん]

応用例：HTML タグを除去

```
<!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>
```

```
out.value=ipt.value.replace(/<[^>]*?>/g,"")
```

コンソール

コンソール
1 + 2;

システムからのメッセージ

第5回　文字列の抽出と検索

先生：こんどは、正規表現から離れ、文字列のインデックス（何文字目か）による、文字列の抽出、検索。

・index　　抽出だけ。書き込み（修正）は不可。
　　for (var c of　"こんにちは") c…　や、for (var i in str) str[i]…　も可。

"こんにちは"[1]　　// “ん”

・charAt()　　文字列の指定した位置の1文字を取得。

"こんにちは".charAt(2);　　// “に”

・slice()　　文字列を切り取る。

result = s.slice(start [, end])

start　切り取りを開始する　0　から始まる文字位置。

end　　切り取りを終了する　0　から始まる文字位置。
　　指定された文字位置の　1つ前の文字までが切り取られる。
　　省略した場合は、文字列の末尾までを切り取る。

start　にマイナス値を指定した場合は、s.length + start　になる。

end　　にマイナス値を指定した場合は、s.length + end　になる。

・substr()　　文字列を切り取る。

result = s.substr(start [, length])

start　切り取りを開始する　0　から始まる文字位置。

length　切り取る文字数。

省略した場合は、文字列の末尾までが切り取られる。

start　にマイナス値を指定した場合は、s.length + start　になる。

"お元気ですか".substr(1, 4); // “元気です”

・substring()　　文字列を切り取る。

result = s.substring(start [, end])

start　切り取りを開始する　0　から始まる文字位置。

end　　切り取りを終了する　0　から始まる文字位置。

指定された文字位置の　1つ前の文字までが切り取られる。

省略した場合は、文字列の末尾までを切り取る。

start　、end　にマイナス値(もしくは　NaN)を指定した場合は、 0　になる。

start　、end　が　s.length　よりも大きい場合は、 s.length　になる。

・split()　　文字列をセパレータで分解。

result = s.split(separator [, limit])

separator　区切り記号

limit　　配列に返される要素の数を指定。

"Mon,Tue,Wed,Thu,Fri,Sat,Sun".split(",");　　// ["Mon","Tue","Wed","Thu","Fri","Sat","Sun"]

・trim()　　文字列の両端の空白を削除。全角空白も。

"　　Hello World　".trim();　// "Hello World"

・indexOf()　　文字列を前方検索。

"こんにちは、こん**にち**は。".indexOf("にち", 3); // 8　4文字目から「にち」を探す

・lastIndexOf()　　文字列を後方検索。

"Hello,**He**llo.".lastIndexOf("He");　　// 6

第6回　文字列と配列の相互変換

先生：文字列は、配列に比べて、書き換えや並べ替えが自由ではありません。そこで、文字列を配列に変換してから処理をして、結果を文字列に再変換する、という方法が考えられます。

・配列→文字列　区切り文字列で結合

```
['Spring','Summer','Fall','Winter'].join(',');        // "Spring,Summer,Fall,Winter"
```

・文字列→配列　区切り文字列で配列にする

```
'Spring,Summer,Fall,Winter'.split(',');               // ["Spring", "Summer", "Fall", "Winter"]
```

・文字列←→配列（一文字ずつ）

```
"Hello World".split("");                              // [“H”,”e”,”l”,”l”,”o”,” “,”W”,”o”,”r”,”l”,”d”]
[“H”,”e”,”l”,”l”,”o”,” “,”W”,”o”,”r”,”l”,”d”].join(''); // "Hello World"

Array.from("Hello World")                             // ["H","e","l","l","o"," ","W","o","r","l","d"]
```

先生：つぎに、配列を操作するのに便利なメソッドをみましょう。

・Array.prototype.map()

与えられた関数を、配列のすべての要素に対して呼び出し、その結果からなる新しい配列を生成。呼び出された配列は変化しない。

一般形　arr.map(callback[, thisArg]);

callback　現在の配列の要素から新しい配列の要素を生み出すための関数。

thisArg　省略可能。callback を実行するときに this として使用するオブジェクト。

- callback は undefined を含め、値が代入されている配列のインデックスに対してのみ呼び出される。すでに削除されたインデックスや、まだ値が代入されていないインデックスに対しては呼び出されない。
- callback は、「要素の値」、「要素のインデックス」、「走査されている Array オブジェクト全体」という3 つの引数をともなって呼び出される。
- map に thisArg パラメータが与えられると、callback の呼び出しのたびにそのオブジェクトが this として使用される。パラメータが省略された場合、this の値としてundefined が渡され、通常の this を決定するルールに従って決まる。
- map によって処理される要素の範囲は、callback が最初に呼び出される前に設定される。map の呼び出しが開始された後に追加された要素に対しては、callback は実行されない。既存の配列要素が変更されたり、削除された場合、callback に渡される値は map がそれらを訪れた時点での値になる。map が削除された要素を訪問することはない。
- map が呼び出された配列がまばらである場合、結果の配列も同じインデックスを空白に保つ。

```
["123","456","789"].map(function (element) { return Number(element); }); // [123,456,789]

var numbers = [1, 5, 10, 15];
var roots = numbers.map(function(x){ return x * 2; });  // roots は [2, 10, 20, 30]
                                                        // numbers は [1, 5, 10, 15] のまま
```

・Array.sort()　ソート

一般形　array.sort(compareFunction)

compareFunction 省略可。例：function(a, b){return a-b}

```
["Banana", "Orange", "Apple", "Mango"].sort();
    // ["Apple", "Banana", "Mango", "Orange"]
```

・ Array.prototype.splice()　　配列削除。

古い要素を取り除きつつ新しい要素を追加することで、配列の内容を変更。

一般形　array.splice(index, howMany, [element1][, ..., elementN]);

index 配列を変化させ始める要素の添え字。値が配列の長さより大きい場合、配列の長さと同じ値となる。値が負数の場合、配列の終端からその値を引いた数が開始位置となる。

howMany 配列から取り除く古い要素の数を示す整数。howMany が 0 の場合、どの要素も取り除かれない。この場合、少なくとも 1 つの新しい要素を指定する必要がある。

element1, ..., elementN配列に追加する要素。要素を指定しなかった場合、splice は単に配列から要素を取り除く。

戻り値 取り除かれた要素を含む配列。要素がひとつのみ削除された場合は、要素数 1 の配列が返される。

array.splice(i,1);　要素 i を削除

・ Array.prototype.filter()

引数として与えられたテスト関数を各配列要素に対して実行し、それに合格したすべての配列要素からなる新しい配列を生成。

一般形　var filteredArray = array.filter(callback[, thisObject]);

callback　　配列の各要素に対して実行するテスト関数

thisObject　callback を実行するときに this として使用するオブジェクト

先生：ついでに、文字コードに関する文字悦操作

・ charCodeAt　　文字列の指定した位置の1文字をUnicodeで取得。

"こんにちは".charCodeAt(2);// 12395

・ fromCharCode 指定したUnicode、ASCII コードを文字で返す。

String.fromCharCode(0x3042, 0xff21, 0xff41, 0xff1f); // "あAa？"

第7回　文字列ー配列相互変換を使った文字列操作

先生：文字列ー配列相互変換を利用した文字列操作の例です。

```
function toUpper(c) {
    var cd = c.charCodeAt(0);
    if ((cd >= "a".charCodeAt(0)) && (cd <= "z".charCodeAt(0)))
                    cd = cd - "a".charCodeAt(0) + "A".charCodeAt(0);
    else if ((cd >= "a".charCodeAt(0)) && (cd <= "z".charCodeAt(0)))
                    cd = cd - "a".charCodeAt(0) + "A".charCodeAt(0);
    return String.fromCharCode(cd);
}
```

小文字→大文字変換の例。

実行例：`"abcABCabcABC".split("").map(toUpper).join('');　// "ABCABCABCABC"`

「文字列操作銃」で使うなら、`out.value=ipt.value.split("").map(toUpper).join('');`

```
function toLower(c) {
    var cd = c.charCodeAt(0);
    if ((cd >= "A".charCodeAt(0)) && (cd <= "Z".charCodeAt(0)))
                    cd = cd - "A".charCodeAt(0) + "a".charCodeAt(0);
    else if ((cd >= "A".charCodeAt(0)) && (cd <= "Z".charCodeAt(0)))
                    cd = cd - "A".charCodeAt(0) + "a".charCodeAt(0);
    return String.fromCharCode(cd);
}
```

大文字→小文字変換の例

実行例：`"abcABCabcABC".split("").map(toLower).join(''); // "abcabcabcabc"`

「文字列操作銃」で使うなら、`out.value=ipt.value.split("").map(toLower).join('');`

```
function toZen(c) {
    var cd = c.charCodeAt(0);
    if ((cd >= "A".charCodeAt(0)) && (cd <= "z".charCodeAt(0)))
                    cd = cd - "A".charCodeAt(0) + "A".charCodeAt(0);
    else if ((cd >= "0".charCodeAt(0)) && (cd <= "9".charCodeAt(0)))
                    cd = cd - "0".charCodeAt(0) + "０".charCodeAt(0);
    return String.fromCharCode(cd);
}
```

半角→全角変換の例。

実行例：`"abcABC123".split("").map(toZen).join('');　// "abcABC１２３"`

```
function toHan(c) {
    var cd = c.charCodeAt(0);
    if ((cd >= "A".charCodeAt(0)) && (cd <= "z".charCodeAt(0)))
                    cd = cd - "A".charCodeAt(0) + "A".charCodeAt(0);
    else if ((cd >= "０".charCodeAt(0)) && (cd <= "9".charCodeAt(0)))
                    cd = cd - "０".charCodeAt(0) + "0".charCodeAt(0);
    return String.fromCharCode(cd);
}
```

全角→半角変換の例。

実行例：`"abcABC１２３".split("").map(toHan).join('');　// "abcABC123"`

第8回　メソッドチェーン

先生：英文テキストを単語に分割し、abc順に並べ替え、重複を除くことで、単語リストを作る例を使います。ソーティングが済んだ配列で、隣り合う、同じ内容の項を統合する uniq（unique） 関数を作ります、メソッドチェーンを使わないと、下記のように、形が複雑になります。

```
function uniq(a) {
    for (var i=0; i<a.length-1; i++)
        if (a[i]==a[i+1]) a.splice(i+1,1);
    return a;
}
```

例：uniq(["abc","abc","def","def","ghi"]); // ["abc","def","ghi"]

文字列変換銃の場合：

```
out.value=uniq(ipt.value.toLowerCase().replace(/[.,\n\s]+/g," ").split(" ").sort()).join(', ');
```

This is a dog, and that is a cat.	→	, a, and, cat, dog, is, that, this

先生：これを、メソッドチェーンの形にしてみましょう。

```
Array.prototype.uniq = function() {
    for (var i=0; i<this.length-1; i++)
        if (this[i]==this[i+1]) this.splice(i+1,1);
    return this;
}
```

例：["abc","abc","def","def","ghi"].uniq()　// ["abc","def","ghi"]

文字列変換銃の場合：

```
out.value=ipt.value.toLowerCase().replace(/[.,\n\s]+/g," ").split(" ").sort().uniq().join(', ');
```

Array クラスの、プロトタイプ関数として定義します。処理結果の配列を return するのがポイントです。

メソッドチェーンは、プログラムがきれいになるというメリットがあります。

しかし、一般的なクラスにプロトタイプ関数を定義するのは、プログラムを複数人で共同開発する場合に、他のプログラマに影響を与えるとか、プログラムを読みにくくするなどの問題点もあります。

第9回　うまい方法

先生：前回紹介した uniq 関数を、filter 関数で実現する方法がインターネットで紹介されています。事前にソーティングしておく必要がない、すぐれものです。（http://qiita.com/cocottejs/items/7afe6d5f27ee7c36c61f）

```
[1,2,3,3,2,2,5].filter(function (x,i,a){return a.indexOf(x)==i;}); // [1,2,3,5]
```

```
out.value=ipt.value.toLowerCase().replace(/[.,\n\s]+/g," ").split(" ")
                                        .sort().filter(function (x,i,a){return a.indexOf(x)==i;}).join(', ');
```

先生：ES2015 で導入された、アロー関数式を使うと、さらにコンパクトになります。

```
[1,2,3,3,2,2,5].filter((x,i,a)=>a.indexOf(x)==i);
```

```
out.value=ipt.value.toLowerCase().replace(/[.,\n\s]+/g," ")
                                        .split(" ").sort().filter((x,i,a)=>a.indexOf(x)==i).join(', ');
```

アロー関数式

```
(param1, param2, …, paramN) => { statements }
(param1, param2, …, paramN) => expression          // これと等価:  => { return expression; }
singleParam => { statements }      // 引数を 1 個しか取らない場合、丸括弧 () は任意
() => { statements }                // 引数を取らない場合、丸括弧が必要
```

先生：ES2015 で導入された Set オブジェクトと、Array.from() を使う、コンパクトな例も提案されています。

```
Array.prototype.uniq = function() {
    return Array.from(new Set(this));
}
```

```
out.value=ipt.value.toLowerCase().replace(/[.,\n\s]+/g," ").split(" ").sort().uniq().join(', ');
```

・ Set オブジェクト　ES2015 で追加された、集合を扱うオブジェクト

生成方法：setObj = new Set()

プロパティ size セット内の要素数を返す

メソッド	説明
add	セットに要素を追加
clear	セットからすべての要素を削除
delete	指定された要素をセットから削除
forEach	セットの各要素に対して、指定された処理を実行
has	セットが指定した要素を格納している場合、true を返す
toString	セットの文字列表現を返す
valueOf	指定されたオブジェクトのプリミティブ値を返す

・ Array.from()

機能：配列ライク (Array-like) オブジェクトや反復可能 (iterable) オブジェクトから新しい Array インスタンスを生成。ArrayLike（または FakeArray）Object は、length プロパティを持ち、数字の添字でアクセス可能な、配列のようなオブジェクト。

構文 Array.from(arrayLike[, mapFn[, thisArg]])

arrayLike	配列に変換するための array-like オブジェクトまたは iterable オブジェクト
mapFn	任意。配列のすべての要素に対して呼び出される Map 関数
thisArg	任意。mapFn を実行する時に this として使用する値
戻り値	新しい Array インスタンス

```
var a = [1,2,3,3,2,2,5];    var b = Array.from(new Set(a));      → b =  [ 1, 2, 3, 5 ]
```

第10回　文字列操作のちょっとしたノウハウ

先生：いくつかのノウハウを紹介しましょう。

・ 最後の行を取得

s = "Hello\nBye";　result = s.substring(s.lastIndexOf("\n"));　→ result = Bye

・
を取り除く

s = "あい
うえ
お
";　result = s.replace(/
/gi , "");　→ result = あいうえお

・ trim()を使わずに両端のスペース（全角含む）を除去

str="␣ Hello World␣ ␣"; result=str.replace(/(^[\s\u3000]+)|([\s\u3000]+$)/g, "");

→ result = "Hello World"

␣は表示上、全角スペースとする。 \u3000 は全角スペースのunicode

・文字列は数字だけか

s = "20150218";　result = s.search(/^[0-9]+$/);　→ result = 0

・6桁でゼロ埋め

i = 12;　result = ("000000"+i).slice(-6);　→ result = 000012

・[ ] の中身を抽出する。

s = "[Hello.]";　start = s.indexOf("[");　end = s.indexOf("]");　result = s.slice(start+1, end);

→ result = Hello.

→文字列内に "/" がいくつあるか数える

s = "/tatsuya/java/browser/";　result = s.split("/").length-1;　result = 4

・文字列を分割する。「か」から分割する。

s = "あいうえおかきくけこ";　str1 = s.substr(0, s.indexOf("か"));　str2 = s.substr(s.indexOf("か"));

→ str1 = あいうえお　str2 = かきくけこ

・文字列を分割する。「お」までを分割する。

s = "あいうえおかきくけこ";　str1 = s.substr(0, s.indexOf("お")+1);　str2 = s.substr(s.indexOf("お")+1);

→ str1 = あいうえお　str2 = かきくけこ

・左1文字を削除する。

s = "お元気ですか";　result = s.substring(1);　→ result = 元気ですか

・右1文字を削除する。

s = "お元気ですか";　result = s.substring(0, s.length-1);　→ result = お元気です

第11回　いろいろな文字列変換

先生：文字列が関係する変換関数を紹介します。

・ toString　　数値を文字列へ変換。基数の指定が可能。
i = 65535;　result = i.toString(16);　→　result = "ffff"

・ JSON.stringify()　　各種オブジェクトを、eval などで評価可能な文字列表現に変換する。
d = { a:1, b: [1,2]}; JSON.stringify(d); // "{\"a\":1,\"b\":[1,2]}"

・ JSON.parse()　　JSON.stringify() の結果の文字列を、元のオブジェクトに戻す。
JSON.parse("{\"a\":1,\"b\":[1,2]}"); // {"a":1,"b":[1,2]}

・escape()
ISO Latin 文字セットで表された引数（文字列）の 16 進エンコーディングを返す。古くからある関数。仕様が不明確で全角文字の扱いがブラウザやバージョンにより異なる実装がされたため。推奨されなくなった。
escape("http://www.co.jp/ファイル"); // **"http%3A//www.co.jp/%u30D5%u30A1%u30A4%u30EB"**

・ unescape()
escape() などによる 16 進エンコーディングの値に対する ASCII 文字列を返す。
unescape("http%3A//www.co.jp/%u30D5%u30A1%u30A4%u30EB"); // "http://www.co.jp/ファイル"

・encodeURI　　ECMAScript 3 で導入され、IEでは5.5から採用。
UTF-8ベースでエンコード。URI において特別な意味を持つ予約文字「/」「:」「&」「+」「=」などはエンコードしない。対象のURI文字列内のパラメータの値に予約文字が存在しないことが前提。文字を UTF-8 文字エンコーディングで表された 1 個から 4 個のエスケープシーケンスに置き換える。
encodeURI("http://www.co.jp/ファイル");
// **"http://www.co.jp/%E3%83%95%E3%82%A1%E3%82%A4%E3%83%AB"**

・decodeURI()
encodeURI 関数あるいは同様のルーチンによって事前に作成された URI をデコード。
decodeURI("http://www.co.jp/%E3%83%95%E3%82%A1%E3%82%A4%E3%83%AB");
// "http://www.co.jp/ファイル"

・ encodeURIComponent　　ECMAScript 3 で導入され、IEでは5.5から採用。
URIで使用する記号をすべてエンコードするため、完全なエンコードになる。「/」「:」「&」「+」「=」なども
エンコードする為、URI全体に適用するとそれ自体はURIとして機能しなくなる。
文字を UTF-8 文字エンコーディングで表された 1 個から 4 個のエスケープシーケンスに置き換える。
encodeURIComponent("http://www.co.jp/ファイル");
// **"http%3A%2F%2Fwww.co.jp%2F%E3%83%95%E3%82%A1%E3%82%A4%E3%83%AB"**

・decodeURIComponent()
encodeURIComponent 関数または同様の処理によって事前に作成された URI の構成要素をデコード。
decodeURIComponent("http%3A%2F%2Fwww.co.jp%2F%E3%83%95%E3%82%A1%E3%82%A4%E3%83%AB") // "http://www.co.jp/ファイル"

・ Base64

```
window.btoa("Hello, world");　　// "SGVsbG8sIHdvcmxk"　全角文字不可
window.atob("SGVsbG8sIHdvcmxk");　// "Hello, world"

var Base64 = { encode: function(str) { return btoa(unescape(encodeURIComponent(str))); },
               decode: function(str) { return decodeURIComponent(escape(atob(str))); }
};

Base64.encode("http://www.co.jp/ファイル"); // "aHR0cDovL3d3dy5jby5qcC/jg5XjgqHjgqTjg6s="
Base64.decode("aHR0cDovL3d3dy5jby5qcC/jg5XjgqHjgqTjg6s="); // "http://www.co.jp/ファイル"
```

第12回　その他

先生：その他、知っておいたほうが良いことを説明します。

・エスケープ文字 特別な意味を持つ文字。

s = "<A href=\"javascript:func('id')\">";　→　s = ‘<A href="javascript:func('id')">’

エスケープ文字	意味
\b	バックスペース
\f	フォームフィード
\n	改行（ニューライン）
\r	復帰（キャリッジリターン）
\t	タブ
\\	バックスラッシュ（¥）
\"	ダブルクォーテーション（"）
\'	シングルクォーテーション（'）
\nnn	8進数
\xnn	16進数
\unnnn	Unicode文字

・+演算子　　文字列と文字列を連結。

s = "Hello, " + "World.";　→　s = Hello, World.

・length　　文字列の長さを取得。

s = "Hello";　result = s.length;　→　result = 5