JavaScript覚書

岐阜経済大学　経営学部 経営情報学科　井戸　伸彦

来歴:

0. 0版　2017年11月30日　JavaのスライドをJavaScriptへ書き換え

スライドの構成

はじめに

- 本スライドは、JavaScriptによるプログラミングを説明するために使用するスライドです。単独で教科書となることは意図していません。
- JavaScriptプログラムはブラウザ上で動作することがほとんどですが、本スライドではブラウザ上で動作することに関わる内容よりも、プログラミング言語としての内容を中心に記載しています。
- 本スライドでは、JavaScriptの限られた機能についてのみ扱っています。また、オブジェクト指向に関する機能については扱っていません。
- JavaScriptについてきちんと勉強したい方は、プログラミング等の講義を受講して頂くことをお願いします。
- 説明は直感的な分かりやすさを重視し、厳密には不正確な言い方もしています。

（１）JavaScriptの動作環境

■本スライドでは、JavaScriptの動作環境に関する一般的な説明は、省略します。

■本スライドの（１）～（１６）のJavaScriptプログラムは、ブラウザ上で動作します。

■本スライドの（N1）～（N2）のJavaScriptプログラム（Node.js）は、コマンドプロンプト（もしくはVScode）から起動します。これらのプログラムを動作させる環境は、大学の第2情報演習室に設定してあります。

■なお、本スライドに掲載したすべてのプログラムは、大学の次のネットワークドライブに置いてあります。

[コンピュータ]-[public]-[ido]-

[programSamples]-[javascript]

（1．1）JavaScriptの書き方

■JavaScript は、HTML文書中で〈script〉～〈/script〉 の間に記述します。

■一般的は、head要素内に関数等を、body要素内に実際に表示を行う処理等を記述します（本スライドでは主に記載の都合上、body要素内に記述しています）。

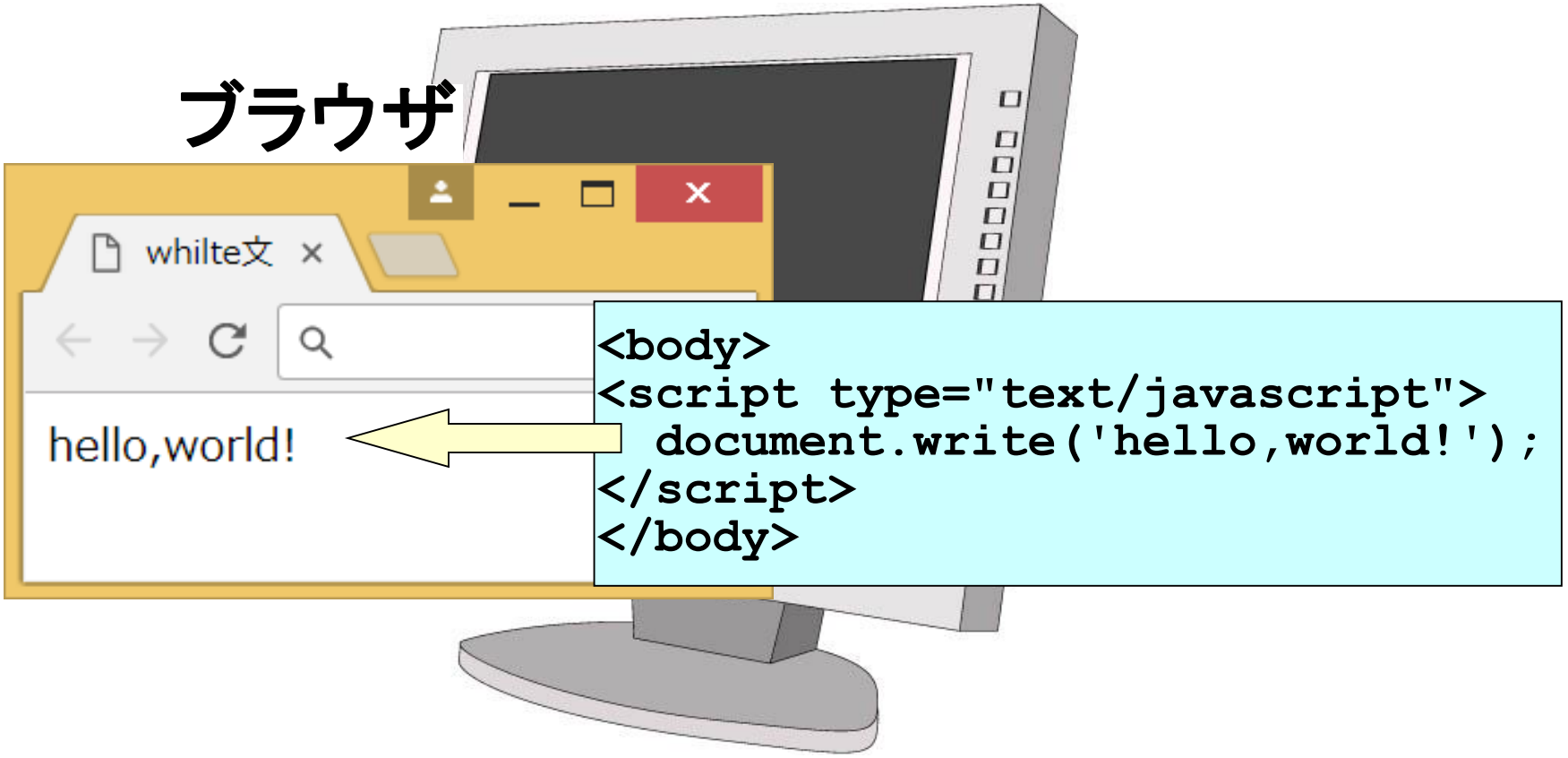
```
<html>
<head>
<script type="text/javascript">
 // JavaScriptのプログラム（関数など）
</script>
</head>
<body>
<script type="text/javascript">
  // JavaScriptのプログラム（実際に表示を行う処理など）
</script>
</body>
</html>
```

本スライドでは、
主にここに記述

（２．１）document.write()

■body要素内に記述することで、ブラウザ画面上に文字列を表示します。

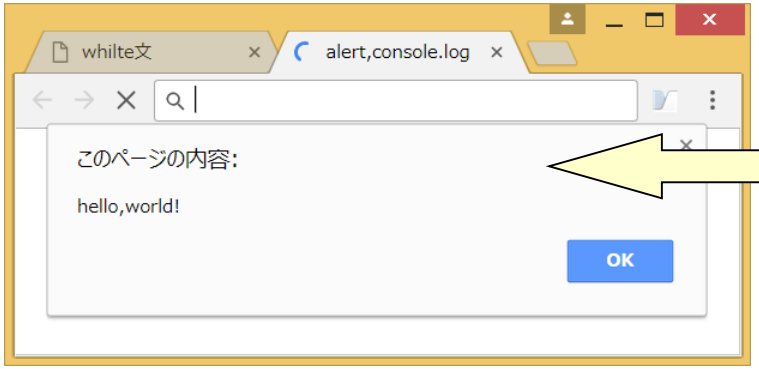
- 実用的なwebページではあまり使われませんが、本スライドではプログラムの動作確認に主に使います。



（2．2）その他の出力方法

■alert(“…”)

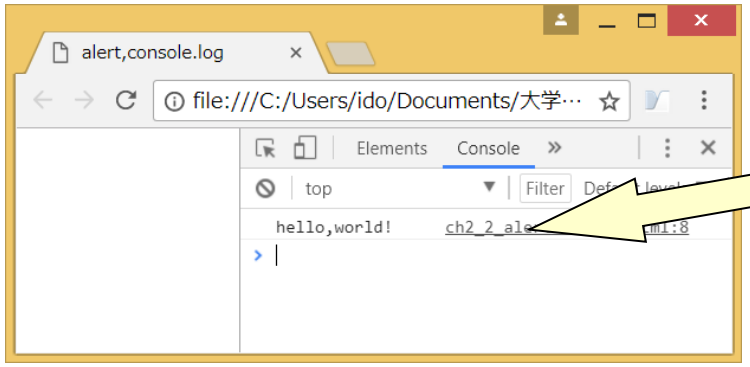
- ポップアップするアラート（警告）が表示されます。




```
<body>
<script type="text/javascript">
alert('hello,world!');
</script>
</body>
```

■console.log(“…”)

- 開発ツールのコンソールに表示されます。




```
<body>
<script type="text/javascript">
console.log('hello,world!');
</script>
</body>
```

（2．3）document.write()の文字列表示

```
<script type="text/javascript">
  document.write('abcd<br />');
  document.write('abcd'+'efg<br />');
  document.write('x='+514+'<br />');
</script>
```

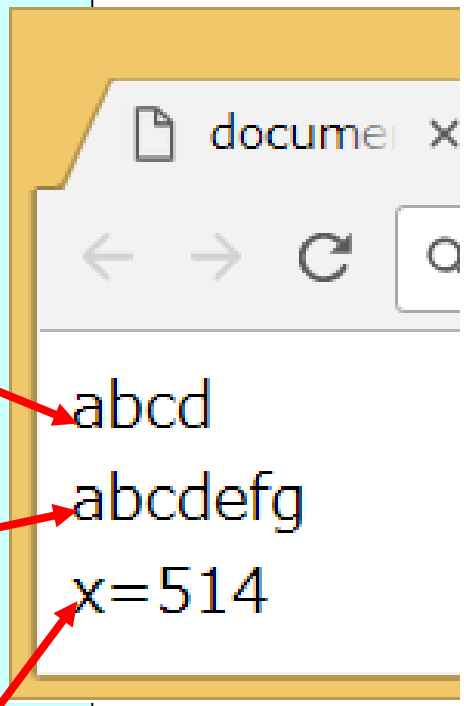
実行文の末尾には、セミコロン“;”。
（改行すれば省略可だが、必ず付けること）

文字列は、’’もしくは、
“”で囲う（違いは別途）

HTMLでの改行

「 + 」は、2つの文字列を連結する。

数値もそのままプリント出来る。



（2．4）¥nと

■用法

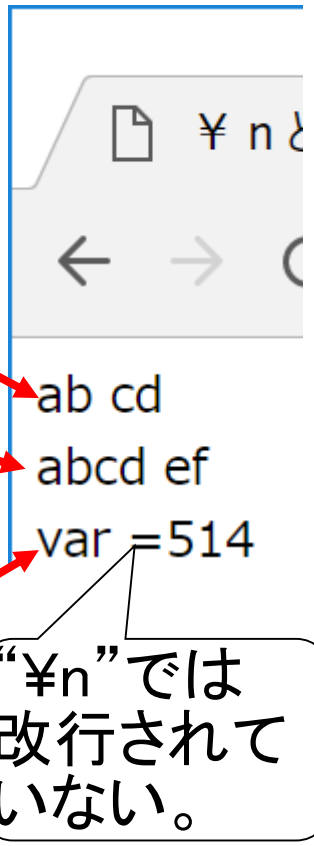
- webページ上の改行は、“
 ”を書き出すこと行います
- 改行文字“**¥n**”（Unixでは“**\n**”）を挿入しても改行されません。

■例

＜プログラム＞

```
document.write("ab¥ncd<br />");

document.write("abc"+"d¥nef<br />");

document.write("var¥n="+514+"<br />");
```

＜出力＞



（2．5）コメント

■形式

- （その1） // これ以降、この行はコメント
- （その2） /* この範囲はコメント、複数行でもOK */

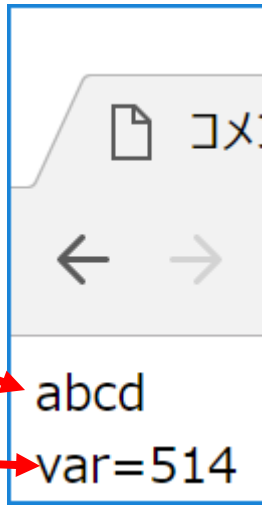
■用法

- プログラム中に、メモを残す場合に使います。プログラムの実行には影響を与えません。

＜プログラム＞

```
<script type="text/javascript">
/* 本プログラムは、
データの出力例を示すためのものである。*/
document.write("abcd<br />"); // OK
//次の行は実行されない（コメントアウトされている）
//document.write("abc"+"def");
//次の行は実行される
document.write("var="+514);
</script>
<!--  sciript要素の外側ではHTMLのコメントを用いる。 -->
```

＜出力＞



（2．6）例題：文字による図形

■右下のような出力を行うプログラムを作成してください（幅をそろえるために“＊”と“＃”とは、日本語で入力してください”）。

■回答例：

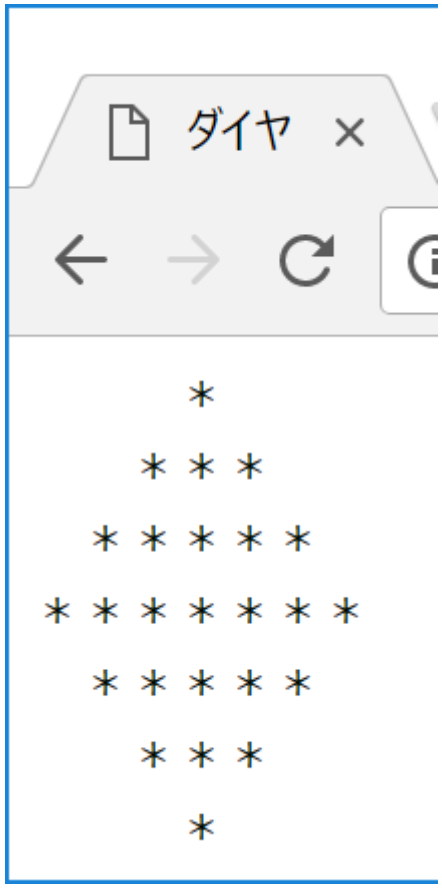
```
<script type="text/javascript">
document.write("#####<br />");
document.write("＊####<br />");
document.write("＊＊###<br />");
document.write("＊＊＊##<br />");
document.write("＊＊＊＊#<br />");
document.write("＊＊＊＊＊<br />");
</script>
```

（２．８）課題

■右のような出力を行うプログラムを作成してください。

- for文を使う必要はありません。
- “＊”は、アスタリスクです。
- 空白（“　”）も、文字として扱われます。
- 幅をそろえるために“＊”、“＃”、および空白は、日本語で入力してください。

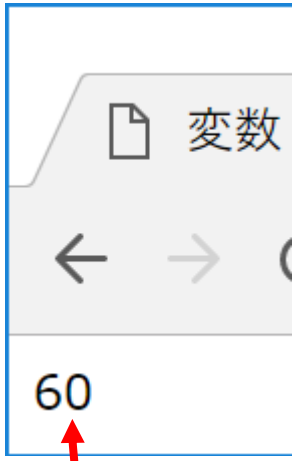
■この問題は簡単ですが、後に続編があります。



（3）変数

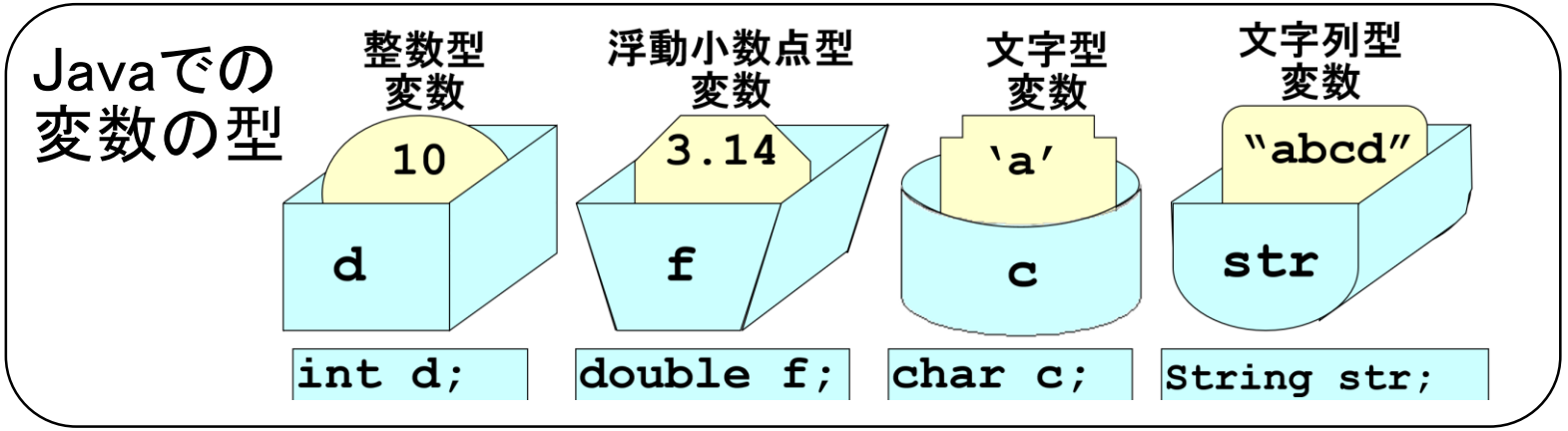
■変数とは、データを入れておく“箱”と考えます。

操作	記述例	操作後の変数の内容
変数x,y,zを用意する	`var x,y,z;`	x　y　z
x,y,zの内容を0にする	`x=0;y=0;z=0;`	0 x　0 y　0 z
xに10を入れる	`x=10;`	10 x　0 y　0 z
yに20を入れる	`y=20;`	10 x　20 y　0 z
xに30を加える	`x=x+30;`	40 x　20 y　0 z
xとyを足してzに入れる	`z=x+y;`	40 x　20 y　60 z
zの内容を出力する	`document.print(z);`	40 x　20 y　60 z



（3．1）変数の型

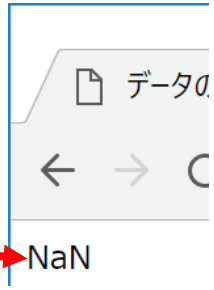
■JavaScriptには、Javaでの「変数の型」のようなものはなく、どのような型のデータでも同じ変数に入ります。



■しかしながら、データに型が無い訳ではありません。

- 次のように、文字型のデータに割り算をすると“NaN”（数値でない）と表示されるように、データの型を意識する必要はあります。

```
<script type="text/javascript">
var x='a';
var y=0.5;
var z=x/y;
document.write(z);
</script>
```



（3．2）変数の出力

■“document.write(…)”により、変数を画面上に出力することができます。

■「＋」を使って、出力をつなぎ合わせることも出来ます。

■例

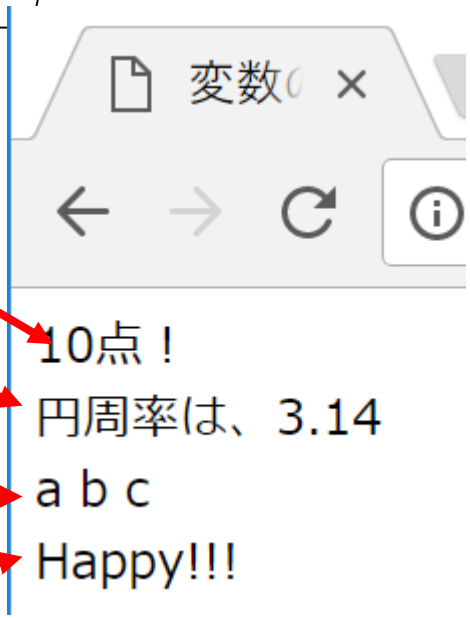
＜プログラム＞

セミコロン“;”で区切って、1行に複数の実行文を書くことも出来る（する必要はない）

```
<script type="text/ja
var d; d=10;
document.write(d+"点！<br />");

var f = 3.14;
document.write("円周率は、"+f+"<br />");

var c = 'a';
document.write(c+" b c<br />");

var str ="Happy";
document.write(str+"!!!<br />");
</script>
```

このような書き方もOK（初期化と言います）

＜出力＞

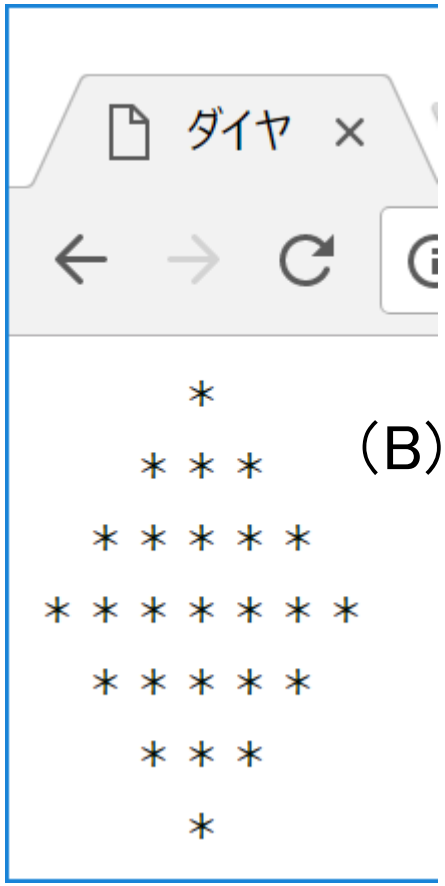


（3．3）課題

■（A)のような変数の宣言を行い、これを用いて（B)の出力を得るプログラムを作成してください。

（A）

```
var stars0="     *<br />";
var stars1="   * * *<br />";
var stars2=" * * * * *<br />";
var stars3="* * * * * * *<br />";
```



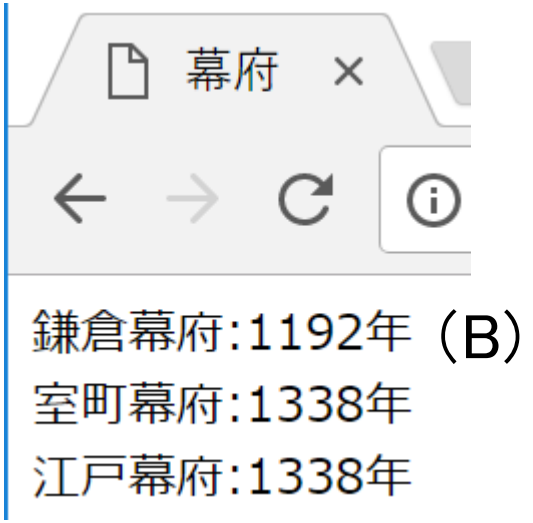
（B）

（3．4）課題

■（A)のような変数の宣言を行い、これを用いて（B)の出力を得るプログラムを作成してください。

（A）

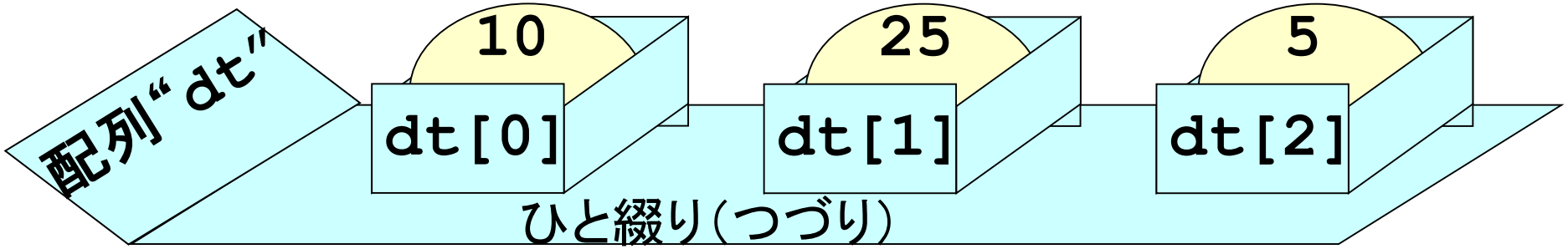
```
var kamakura=1192;
var muromachi=1338;
var edo=1338;
var bakufu="幕府";
var nen="年";
```



（B）

（4）配列

■配列は、番号のついた箱（変数）のひと綴りと考えます。

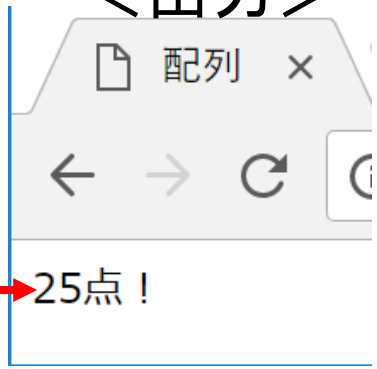


■配列の各々の要素は、変数と同じように扱うことが出来ます。

＜プログラム＞

```
<script type="text/javascript">
var dt=new Array();
dt[1]=25;
document.write(dt[1]+"点！");
</script>
```

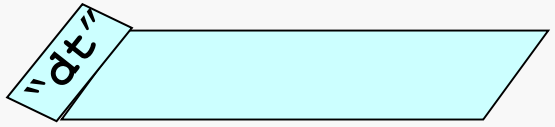
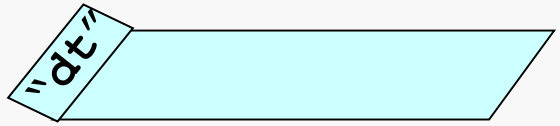
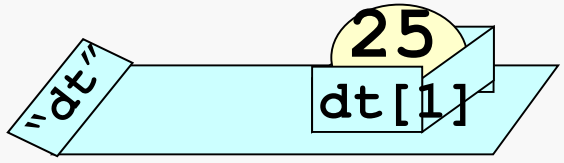
＜出力＞



■ひとつの配列の各要素は、みな同じデータ型となります。番号は、“0”から始まります。

（４．１）配列の宣言

- ①配列変数の名前の宣言と、②データが格納される実体の確保とは、別になります。
- 先頭（dt[0]）から作る必要はありません。

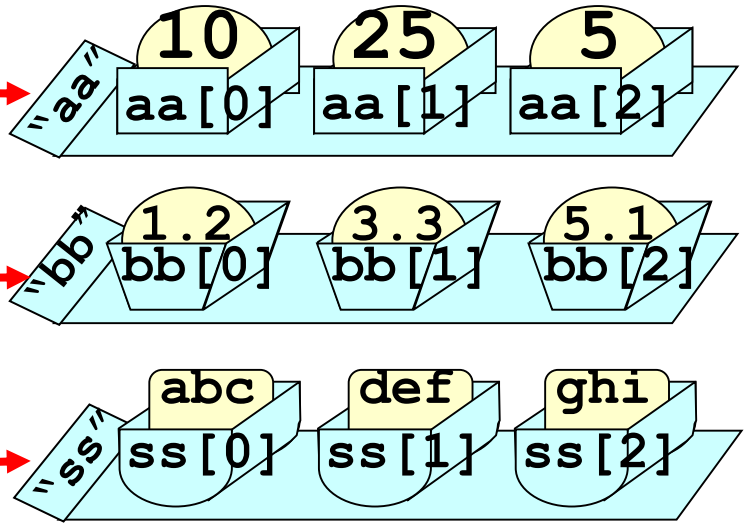
操作	記述例	変数の内容
①配列変数dtを宣言する	`var dt=new Array();`	
このように書いても同じ。	`var dt=[];`	
②番号1の要素（dt[1]）に25を入れる	`dt[1]=25;`	

（4．2）配列の初期化

■変数を宣言した際に値を入れておくことを、初期化と言います。（スライド(3.2): **`var f = 3.14;`** ）

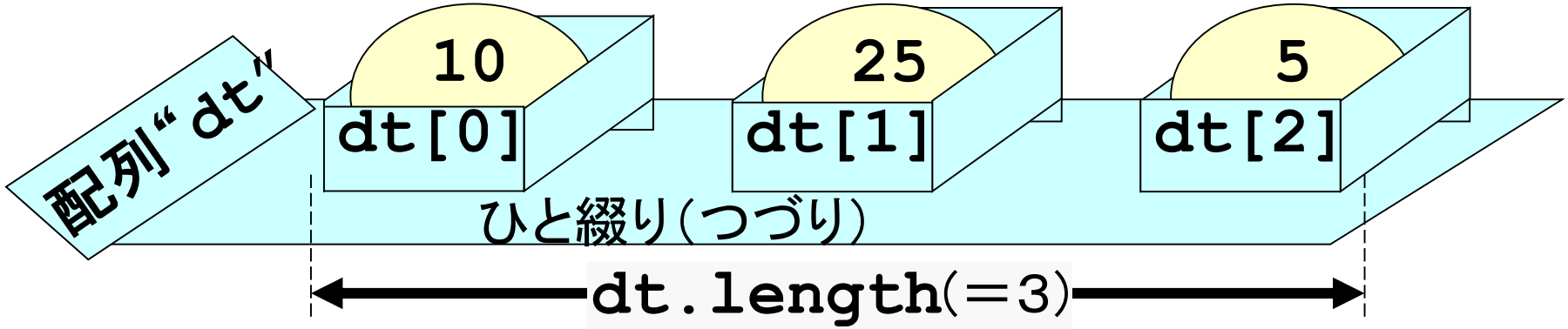
■配列も、初期化を行うことが出来ます。その際、配列の長さは、値が入れられた数に設定されます。

```
var aa = [10,25,5];

var bb = [1.2, 3.3, 5.1];

var ss = ["abc","def","ghi"];

document.write(aa[0]+"点！、長さは"+
                    bb[1]+"cm、"+ss[2]);
```

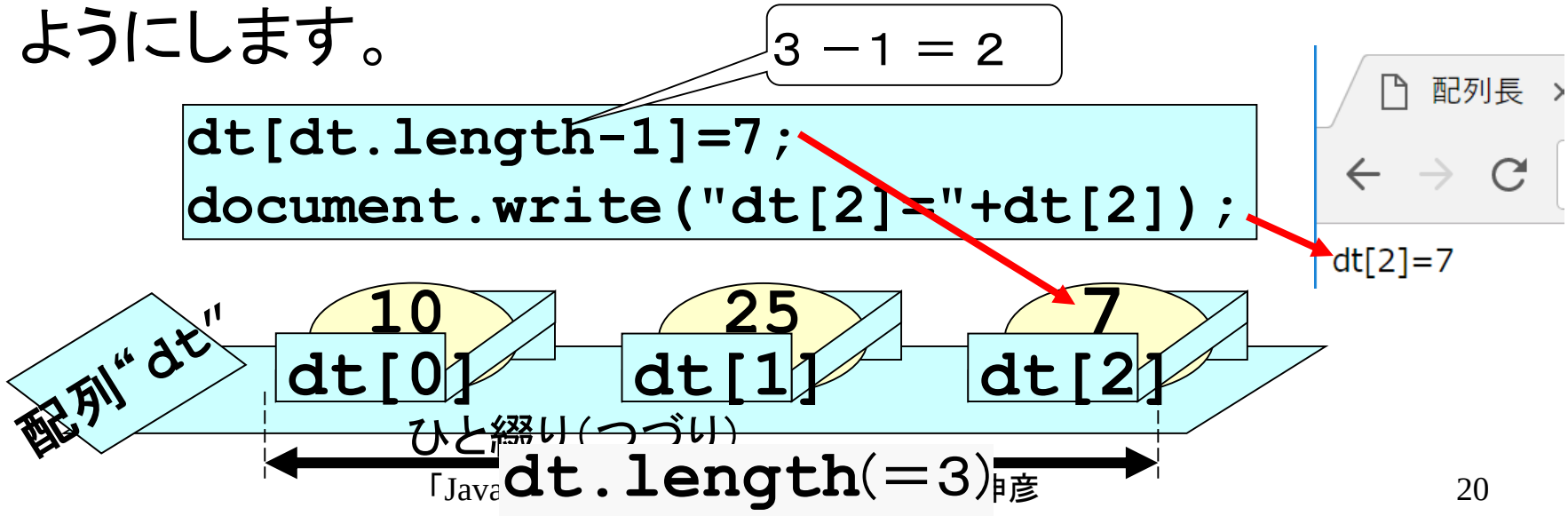


（4．3）配列長

■配列“dt”の長さ（要素の数）は、“dt.length”と表すことが出来ます。

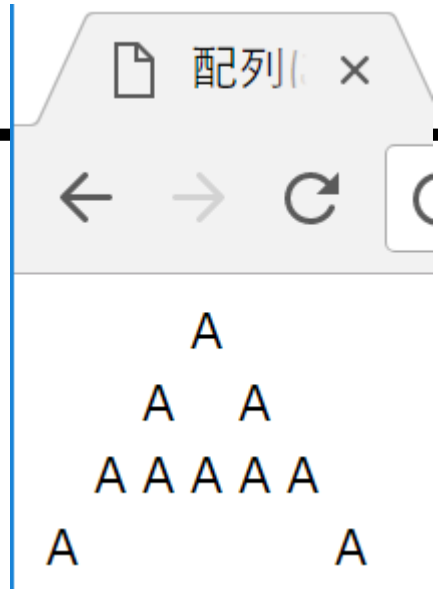


■配列の一番最後の要素に“5”を設定したいとき、次のようにします。



（4．4）例題

■各行を配列に入れて、右のような文字を出力してください。幅をそろえるために“A”、および空白は、日本語で入力してください。



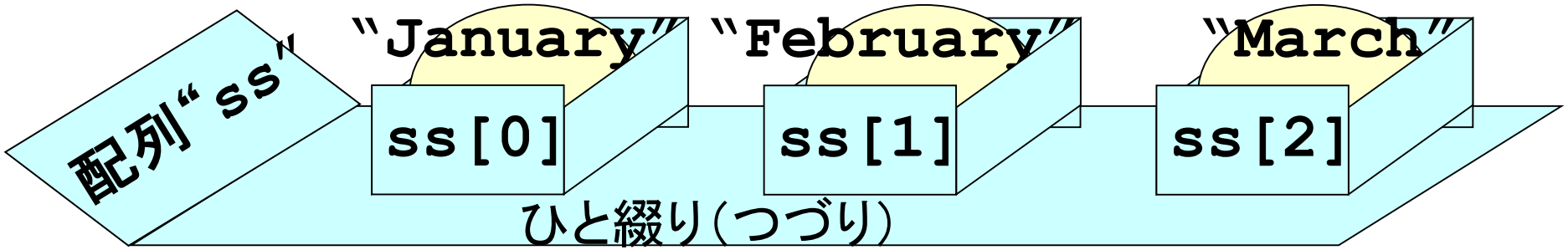
■解答例

```
<script type="text/javascript">
var aa=[
      '    A',
      '  A A',
      ' AAAAA',
      'A     A'];
document.write(aa[0]+"<br />");
document.write(aa[1]+"<br />");
document.write(aa[2]+"<br />");
document.write(aa[3]+"<br />");
</script>
```

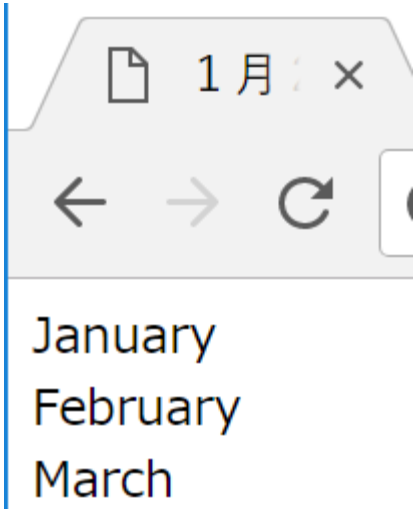
（4．5）課題

■課題４－１

- 次のような値を持つ配列を持ち、順に出力するプログラムを作成してください。



＜出力＞

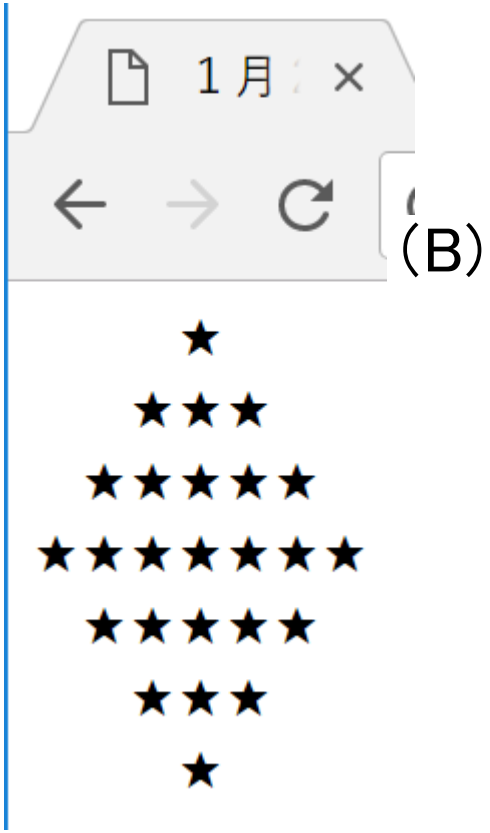


（４．６）課題

■（A)のような変数の宣言を行い、これを用いて（B)の出力を得るプログラムを作成してください。

（A）

```
var stars=[
      '    ★',
      '   ★★★',
      '  ★★★★★',
      ' ★★★★★★★'];
```

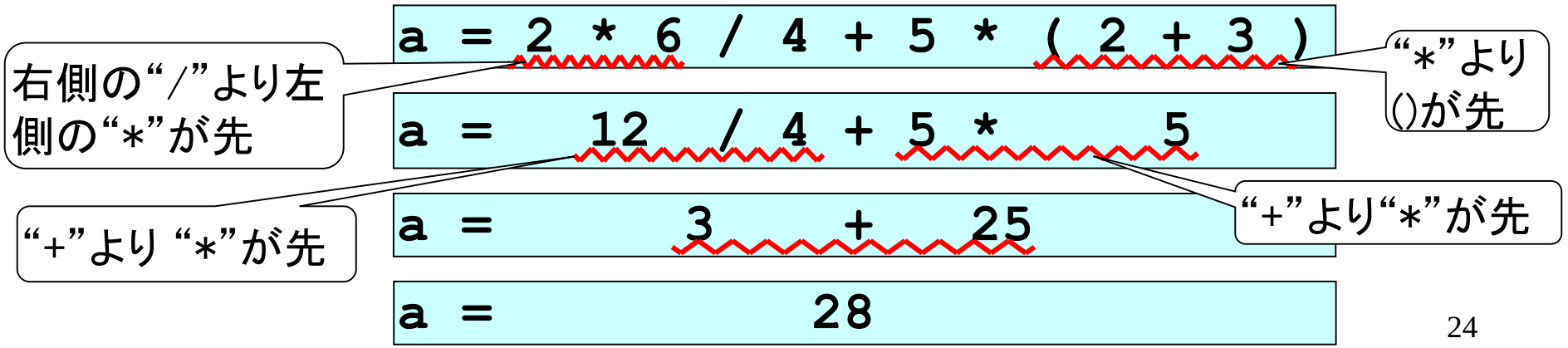


(5)演算

■算術演算子

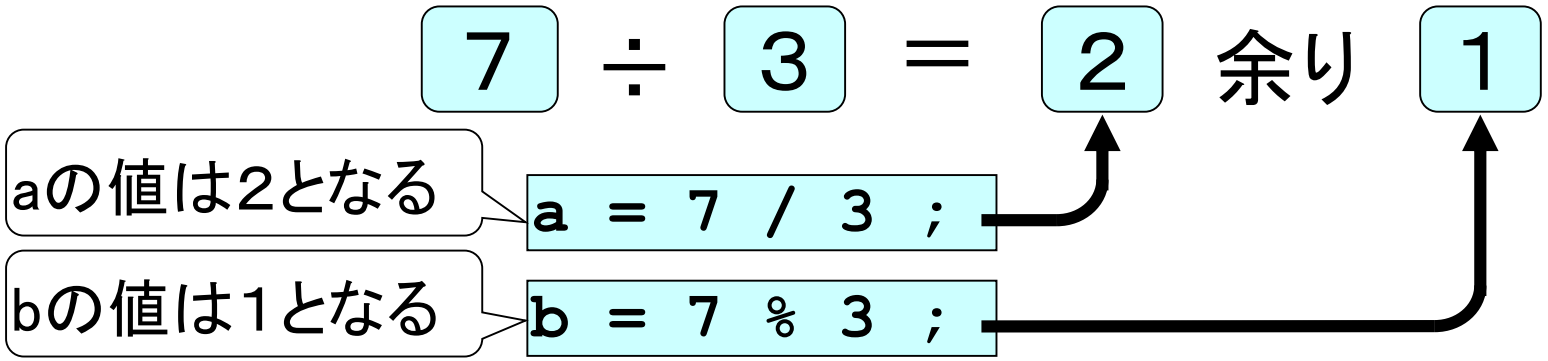
演算子	意味	使用例	優先度
*	掛け算	a = b * c ;	高
/	割り算	a = b / c ;	
%	あまり	a = b % c ;	
+	足し算	a = b + c ;	低
-	引き算	a = b - c ;	

- 優先度が高いものは、低いものより先に計算します。同じ優先度のもの同士は、左から計算します。
- ()があれば、その中身を先に計算します。

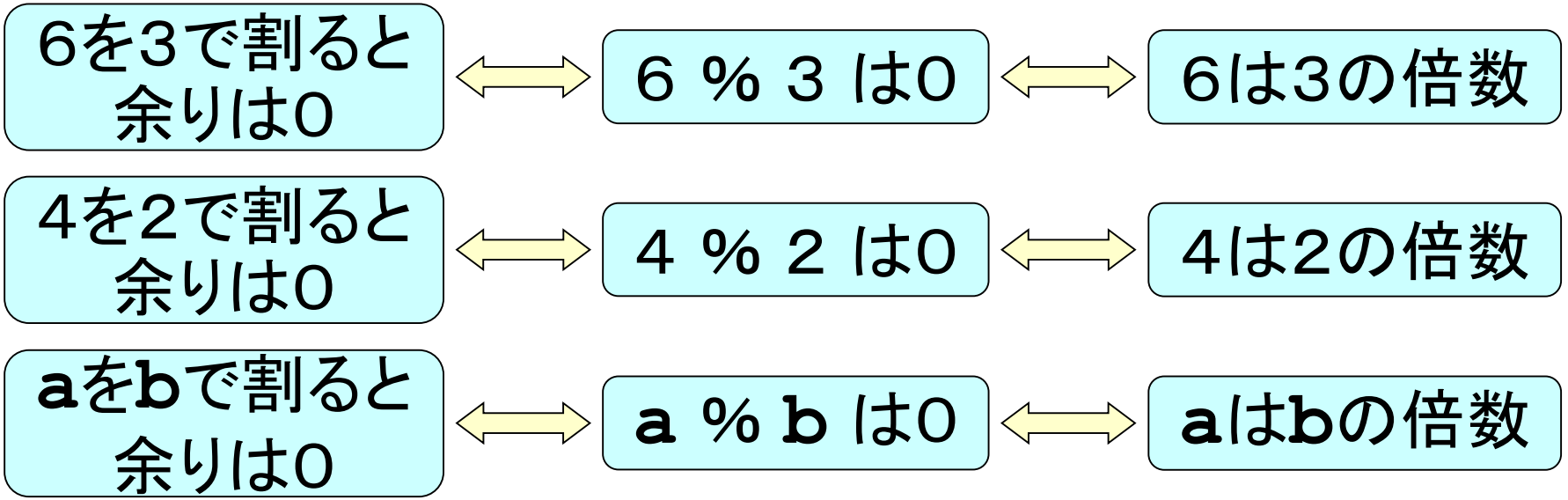


（５．１）“％”（余り、剰余）

■整数の割り算（／）と余り（％）



■余りと倍数



（５．２）インクリメント演算子、その他

■インクリメント演算子

`++a;` または `a++;` ⇒ `a=a+1;` と同じ。

■デクリメント演算子

`--a;` または `a--;` ⇒ `a=a-1;` と同じ。

■インクリメント演算子の代入（使用しなくてもOK）

`b = a++;` ⇒ `b = a; a = a+1;` と同じ。

`b = ++a;` ⇒ `a = a+1; b = a;` と同じ。

■代入演算子

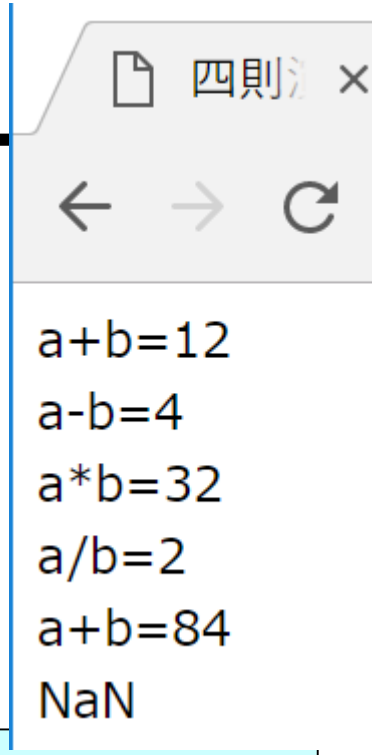
`a += 3;` ⇒ `a = a + 3;` と同じ。

`a *= 3;` ⇒ `a = a * 3;` と同じ。

- この他、“-=”、“/=”、“%=”なども同じ。

（5．3）例題

■ふたつの変数”a”,”b”を宣言し、それぞれの値を8と4として、その四則演算（加減乗除：＋、－、＊、／）の結果を次のように出力するプログラムを作成してください。



■解答例

なぜ括弧“(...)”が必要なのか？

```
<script type="text/javasc  pt">
var a=8; var b=4;
document.write("a+b="+(a+b)+"<br />");
document.write("a-b="+(a-b)+"<br />");
document.write("a*b="+a*b+"<br />");
document.write("a/b="+a/b+"<br />");
document.write("a+b="+a+b+"<br />");
document.write("a-b="+a-b+"<br />");
</script>
```

括弧“(...)”が無いとどうなるか？

（５．３．１）足し算で括弧が無い場合

■次のプログラムは、意図と違う出力をしてしまいます。

```
        :
document.write("a+b="+a+b+"<br />");
        :
```

8　4

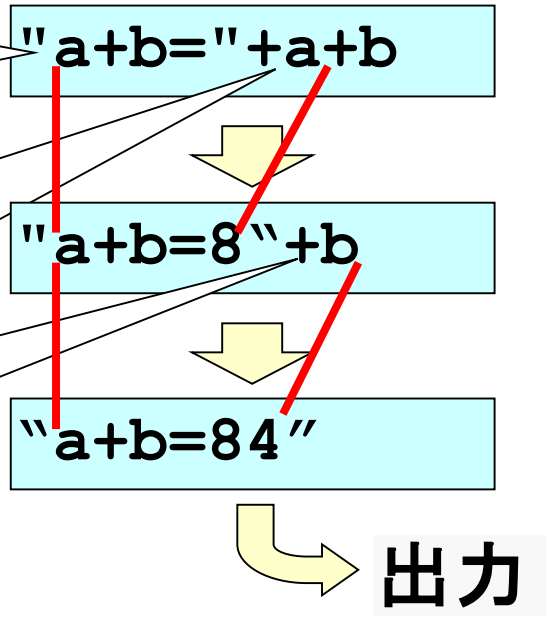
a+b=84

■コンパイラは、次のように解釈します。

①document.writeは文字列を引数とする。“a+b=”は文字列。

②文字列のあとに+だから、文字列を連結することにしよう。次は数値のaだけど、数値はその文字列と解釈する約束だから、aは文字の“8”として連結しよう

③文字列のあとに+だから、文字列を連結することにしよう。次は数値のbだけど、数値はその文字列と解釈する約束だから、bは文字の“4”として連結しよう



（５．３．２）括弧があれば。。。

■次のプログラムは、意図通りの動作をします。

```
        :
document.write("a+b="+(a+b)+"<br />");
        :
```

8　4

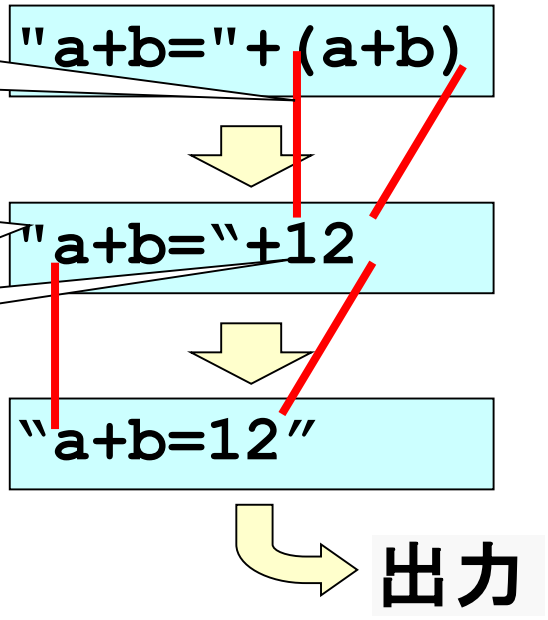
a+b=12

■コンパイラは、次のように解釈します。

①括弧“(…)”は最初に解釈する、
引き算だから、8+4＝12だ！

②document.write()は文字列を引数とする。
“a+b=”は文字列。

③文字列のあとに+だから、文字列を連結することにしよう。次は数値の12だけど、数値はその文字列と解釈する約束だから、文字列の“12”として連結しよう



（５．３．３）引き算で括弧が無い場合

■引き算の場合は、文字列でとなります。

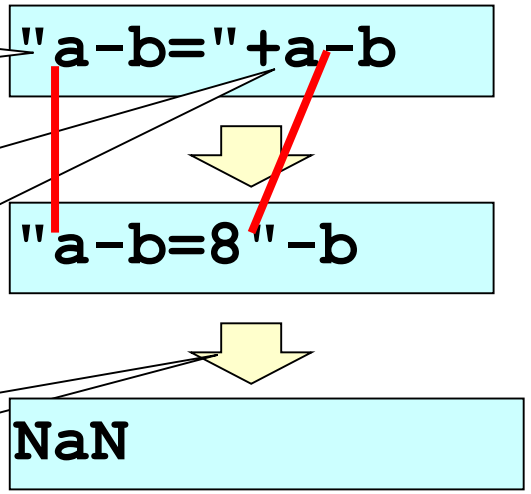
```
        :
document.write("a-b="+a-b+"<br />");
        :
```

■コンパイラは、次のように解釈してエラーとします。

①System.out.printlnは文字列を引数とする。“a+b=”は文字列。

②文字列のあとに+だから、文字列を連結することにしよう。次は数値のaだけど、数値はその文字列と解釈する約束だから、aは文字の“８”として連結しよう

③文字列“a-b=8”に対して、
-というのは、数にならない。
NaN（Not a Number）だ！



（５．３．４）括弧があれば。。。

■次のプログラムは、意図通りの動作をします。

```
        :
    document.write("a-b="+(a-b)+"<br />");
        :
```

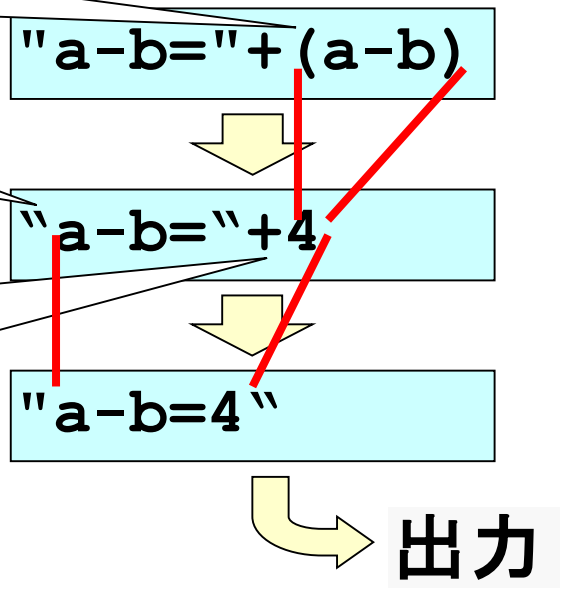
8　4

■コンパイラは、次のように解釈します。

①括弧“(…)”は最初に解釈する、
引き算だから、８－４＝4だ！

②System.out.printlnは文字列を引数とするから、まず“a-b=”は文字列。

③文字列“a-b=”に対して、+というのは、文字列を連結すること。数値は文字列と解釈するから、数値の4は、文字列の“4”として連結しよう！



（5．3．5）掛け算はOK！

■次のプログラムは、意図通りの動作をします。

```
        :
document.write("a*b="+a*b+"<br />");
        :
```

8　4

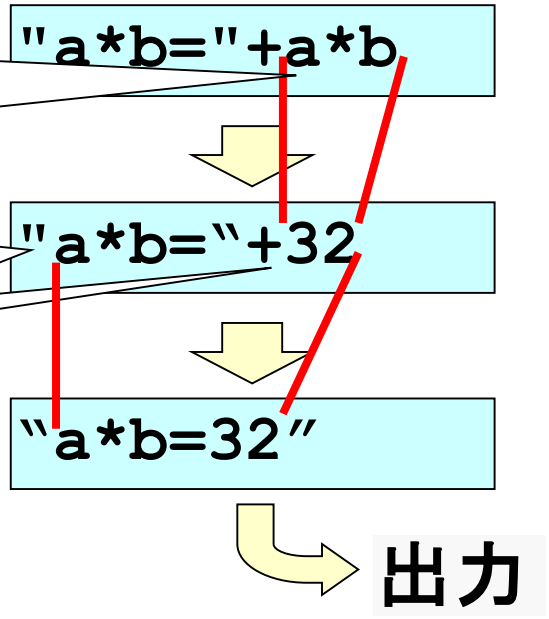
a*b=32

■コンパイラは、次のように解釈します。

①掛け算＊は、連結+よりも、優先して解釈する、
掛け算だから、8*4＝32だ！

②System.out.printlnは文字列を引数とするから、まず“a*b=”は文字列。

③文字列のあとに+だから、文字列を連結することにしよう。次は数値の32だけど、数値はその文字列と解釈する約束だから、文字列の“32”として連結しよう



（５．３．６）割り算もOK！

■次のプログラムは、意図通りの動作をします。

```
      :
document.write("a/b="+a/b+"<br />");
      :
```

8　4

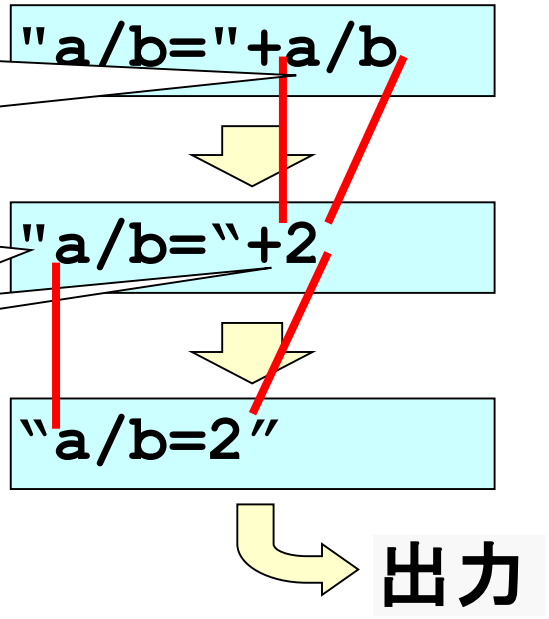
a/b=2

■コンパイラは、次のように解釈します。

①掛け算/は、連結+よりも、
優先して解釈する、
掛け算だから、８/４＝2だ！

②System.out.println は文字列を引数とするから、まず“a/b=”は文字列。

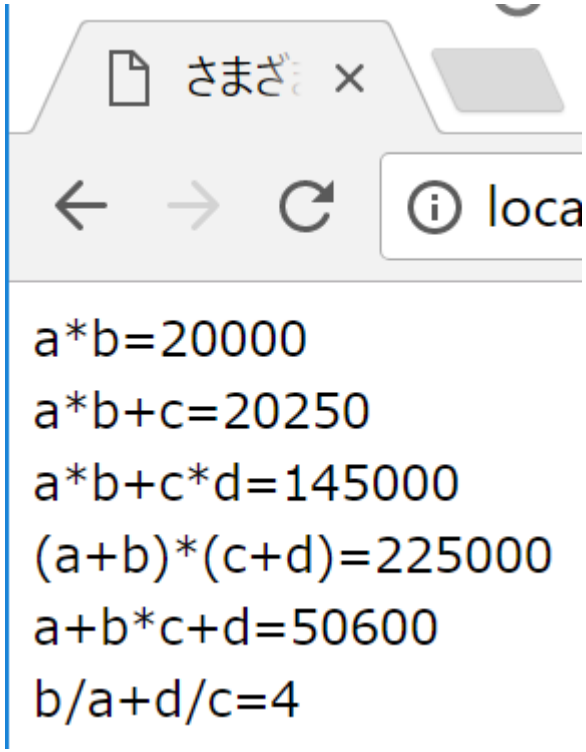
③文字列のあとに+だから、文字列を連結することにしよう。次は数値の2だけど、数値はその文字列と解釈する約束だから、文字列の“2”として連結しよう



（５．４）課題

■課題（例題を参考にしてください）

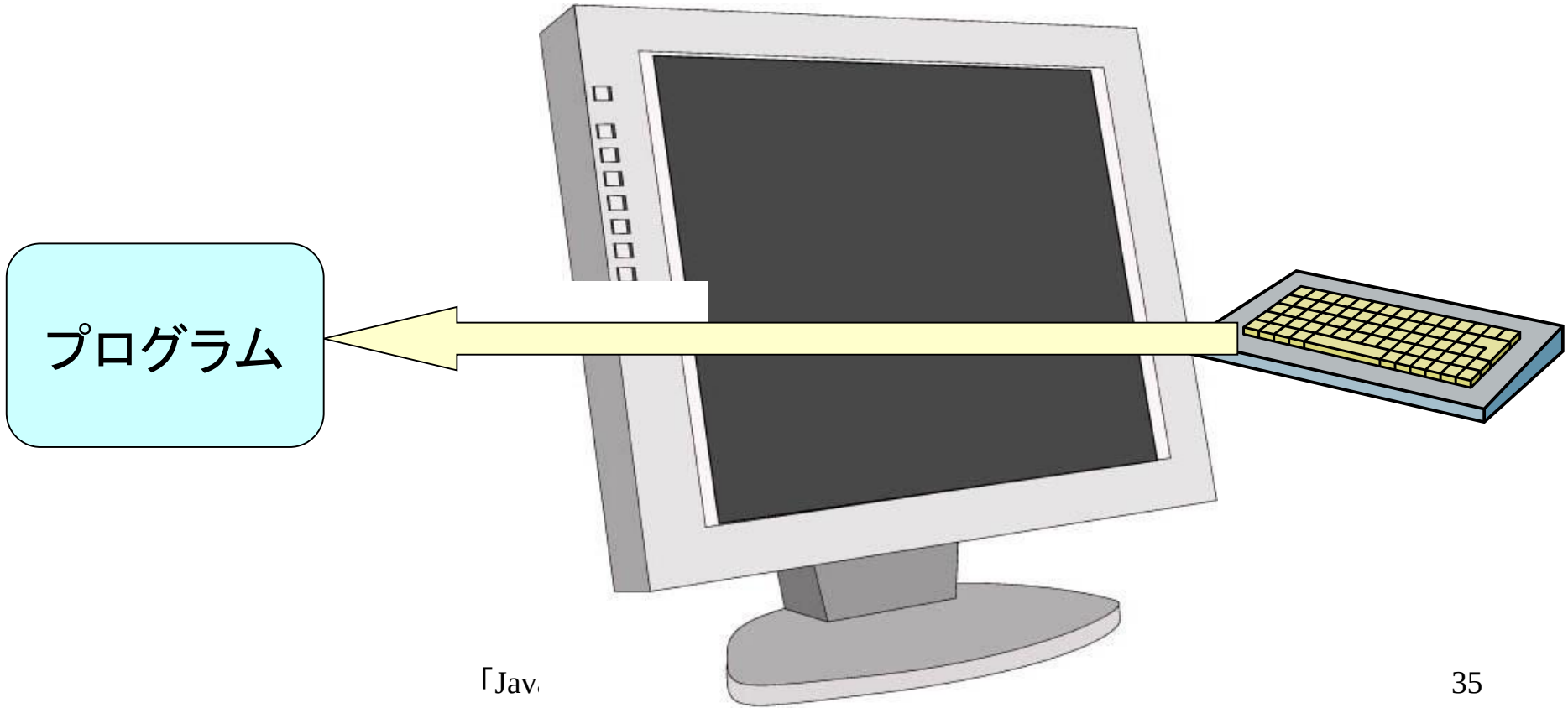
- 変数”a”,”b”,”c”,”d”（それぞれの値は、100,200,250,500）を宣言し、次の表示結果を得るように、計算を行うプログラムを作成してください。



（6）データの入力

■何をしたいのか？

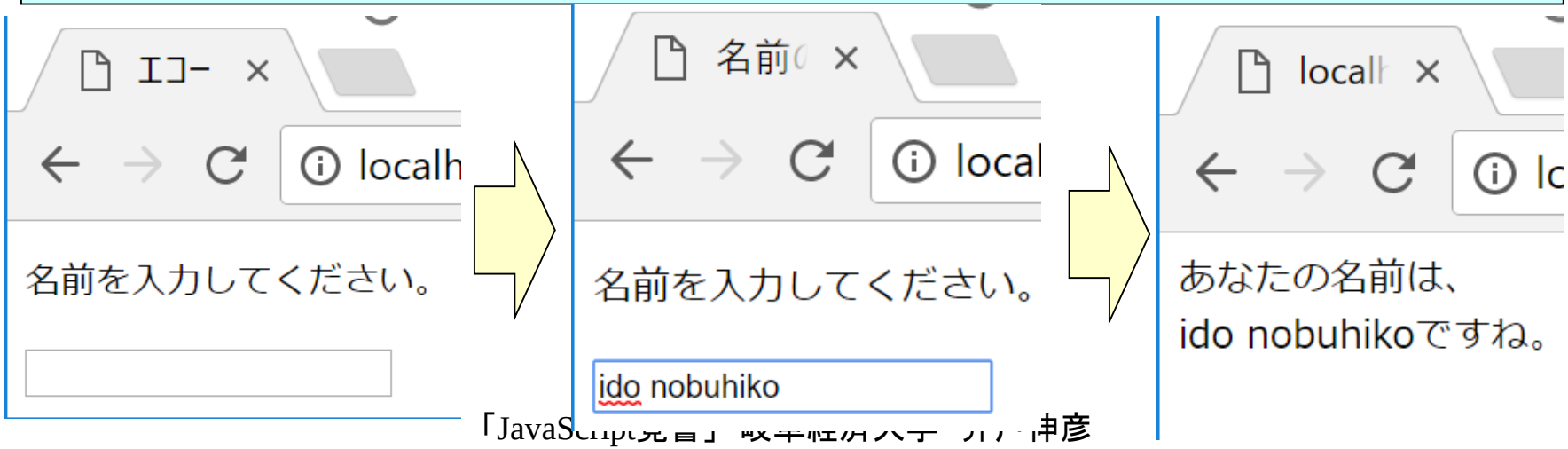
- 作成するプログラムへ、使用している端末のキーボード（標準入力という言い方もします）から、文字列を入力します。
- ここでは、input要素を使って文字列を入力する方法を説明します。



（6．1）入力を行うプログラム

■次のようなプログラムを動かしてみます。

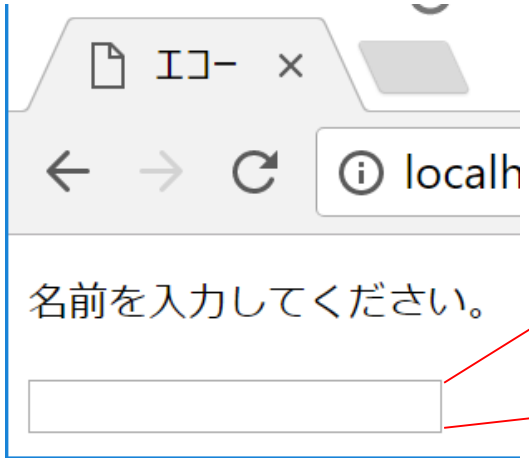
```
<body>
<script type="text/javascript">
function echo(){
  var name = document.form1.name_in.value;
  document.write('あなたの名前は、<br />'+name+'ですね。');
}
</script><p>名前を入力してください。</p>
<form name="form1">
<input type="text" name="name_in" onchange="echo();"/><br />
</form>
</body>
```



（6．2）input要素

■前のスライド(6.1)のプログラムでは、form要素の中にinput要素を記述することにより、ブラウザ上の入力するエリアを作成しています。

■Input要素は属性に「type=“text”」と設定することで、このように文字列を入力するエリアになります。

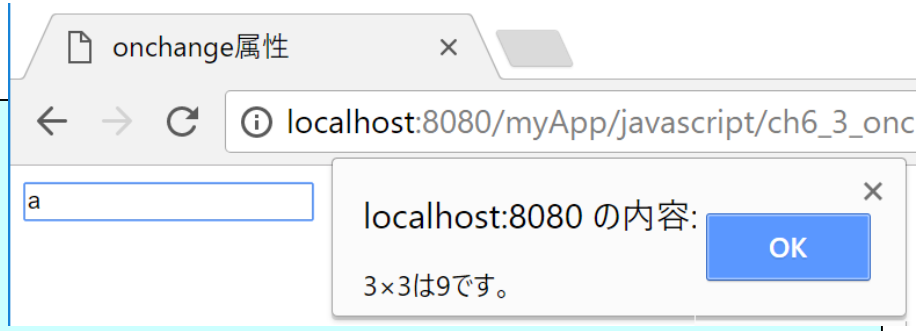



```
</script><p>名前を入力してください。</p>
<form name="form1">
<input type="text"
       name="name_in"
       onchange="echo();"/><br />
</form>
</body>
```

（6．3）onchange属性

■Input要素中のonchange属性には、プログラムを文字列（すなわち、ダブルコーテーション“”で囲む）として記述することが出来ます。そのプログラムは、inputの内容が変化した際に起動されて動きます。

■例えば、次のように計算をしてその結果をalert（スライド(2.2)参照）で表示することも出来ます。




```
<body>
<form name="form1">
<input type="text"
       name="name_in"
       onchange="var x=3*3;
                 alert('3×3は'+x+'です。');"/><br />
</form>
</body>
```

（６．４）関数(function)

■前のスライド(6.3)の例のように、onchange属性にプログラムをたくさん書くと見づらくなるので、前のスライド(6.1)では、関数を利用しています。

■Script要素中に関数を定義して、それをonchangeから呼び出しています。

■関数についての少し詳しい説明は、スライド(12)にて改めて行います。

```
<body>
<script type="text/javascript">
function echo(){
  // ここのプログラムが実行される
}
</script>
<p>名前を入力してください。</p>
<form name="form1">
<input type="text" name="name_in"
       onchange="echo();"/><br />
</form>
</body>
```

"echo"と言う名前の関数の定義

ジャンプ！

（6．5）input要素の値の取得

■Input要素の値は、form要素、input要素の名前（name属性により設定）を指定して、value属性により取り出すことが出来ます。

①全体が”document”で、

②”form1”の中の、

③”name_in”の中の、

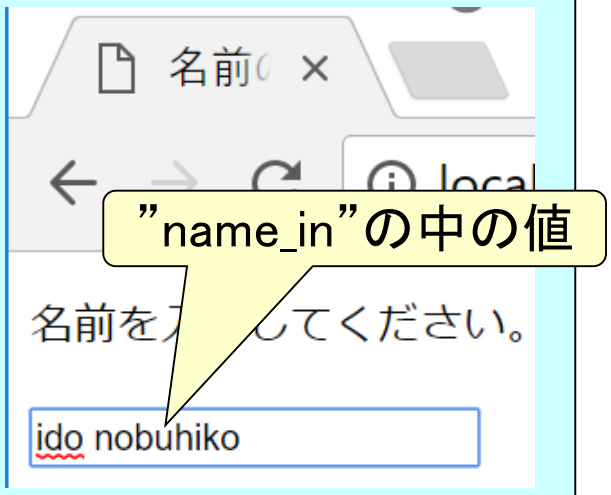
④値を、

⑤”name”に代入。

```
<body>
<script type="text/javascript">
function echo(){
  var name = document.form1.name_in.value;
  document.write('あなたの名前は、<br />'+name+'ですね。');
}
</script><p>名前を入力してください。</p>

<form name="form1">

<input type="text"
       name="name_in"
       onchange="echo();"/><br />
</form>
</body>
```

form1

name_in



”name_in”の中の値

（6．6）id属性の利用

■スライド(6.1)と同様な処理は、次のようにid属性を用いて行う方が一般的です。

■id属性は、ドキュメント内でユニークである必要があります。

①全体の”document”の中で、

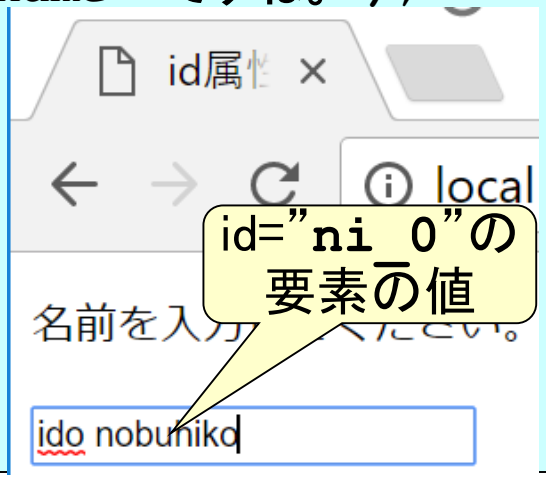
④”name”に代入。

②id属性が”ni_0”である要素(=element)の

③値を、

```
<body>
<script type="text/javascript">
function echo(){
  var name = document.getElementById("ni_0").value;
  document.write('あなたの名前は、<br />'+name+'ですね。');
}
</script><p>名前を入力してください。</p>
<form name="form1">
<input type="text"
       id="ni_0"
       onchange="echo();"/><br />
</form>
</body>
```

id=“ni_0”の要素

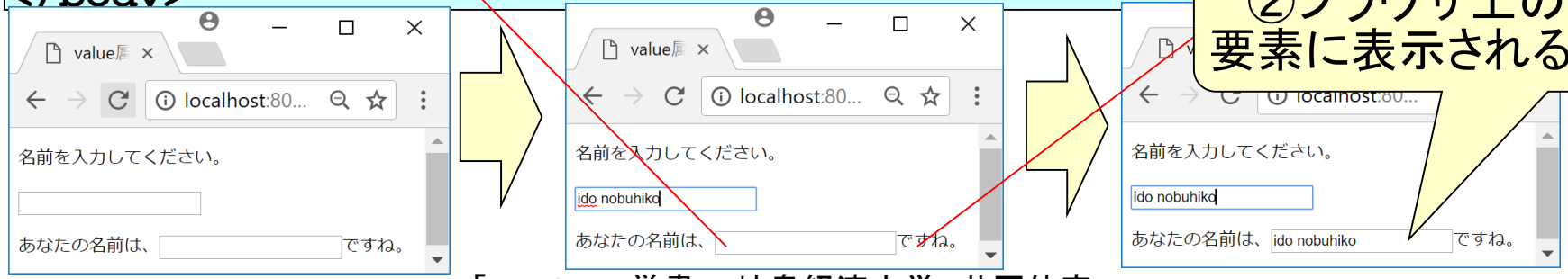


（6．7）value属性の設定

■input要素のvalue属性は、読み取るだけでなく、設定することも出来ます。設定すれば、ブラウザ上で表示されることになります。

```
<body>
<script type="text/javascript">
function echo(){
        var name = document.form1.name_in.value;
        document.form1.name_out.value=name;
}
</script><p>名前を入力してください。</p>
<form name="form1">
<input type="text" name="name_in" onchange="echo();"/><br />
<p>あなたの名前は、<input type="text" name="name_out" />ですね。</p>
</form>
</body>
```

①input要素のvalue属性に、値を設定すると、

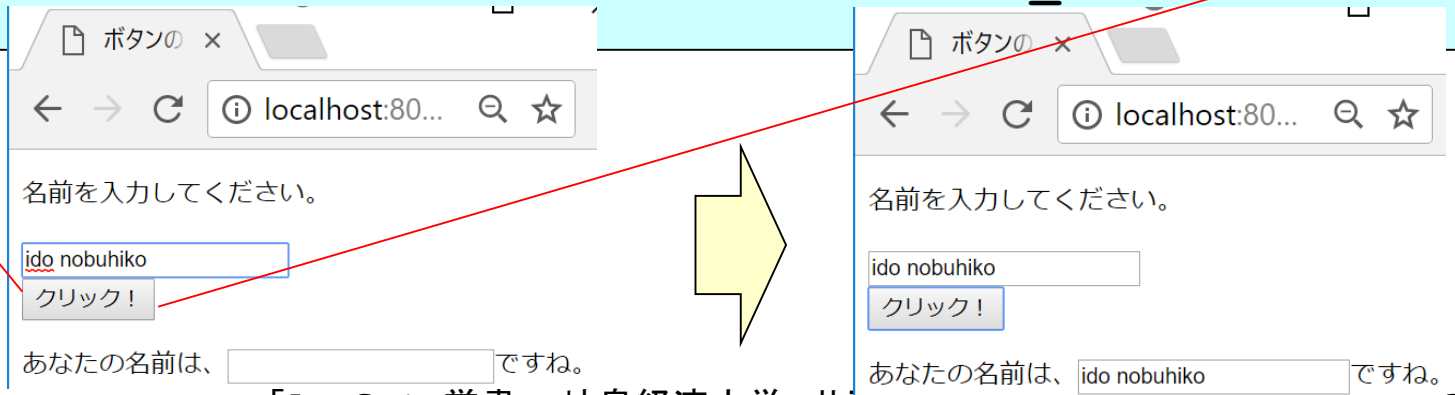


②ブラウザ上の要素に表示される、

（6．8）ボタンの利用

■ここまではinput要素（text）のonchange属性で関数を起動してきましたが、多くの場合はボタンのクリックで起動することが普通です。

```
<script type="text/javascript">
function echo(){
        var name = document.form1.name_in.value;
        document.form1.name_out.value=name;
}
</script>
<p>名前を入力してください。</p>
<form name="form1">
<input type="text" name="name_in"/><br />
<input type="button" value="クリック！" onclick="echo();"/><br />
<p>あなたの名前は、<input type="text" name="name_out" />ですね。</p>
</form>
```

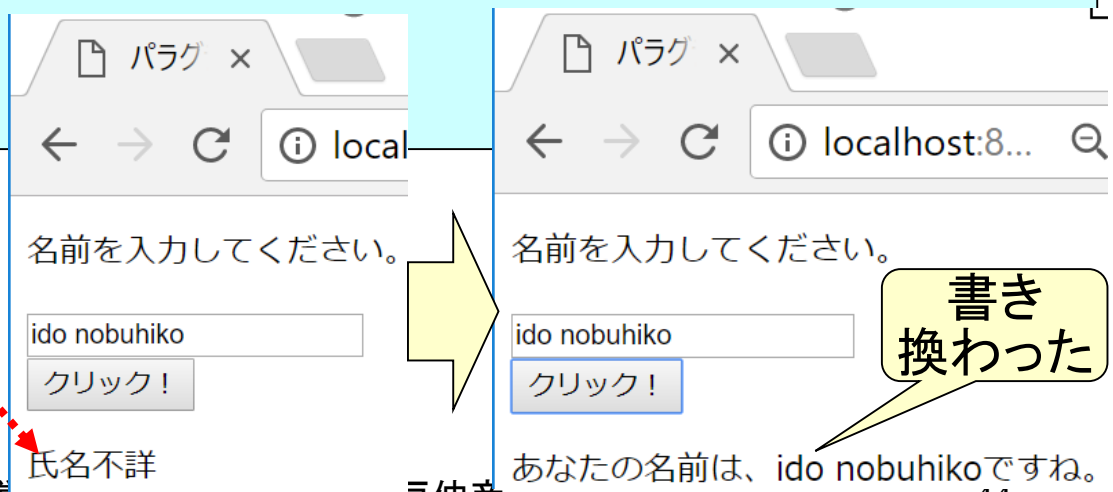


（6．9）その他の要素への設定

■input要素に限らず、さまざまな要素に値の設定は可能です(onchangeではうまく行かない場合があります)。

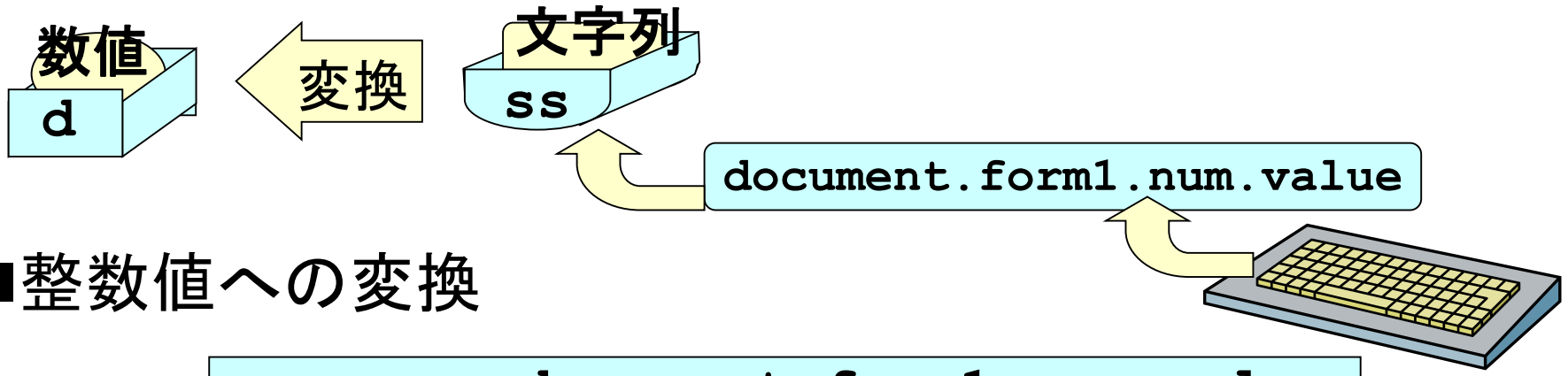
```
<body>
<script type="text/javascript">
function echo(){
        var name = document.getElementById("ni_0").value;
        document.getElementById("no_0").innerText
        ="あなたの名前は、"+name+"ですね。";
}
</script><p>名前を入力してください。</p>
<form name="form1">
<input type="text" id="ni_0" /><br />
<input type="button" value='クリック！' onclick="echo();"/><br />
</form>
<p id="no_0">氏名不詳</p>
</body>
```

valueではなくinnerTextに設定



（6．10）数値への変換

■キーボードから数値を入力したい際には、まず、文字列として入力してから、数値へ変換します。



■整数値への変換

```
var ss = document.form1.num.value;
var d = parseInt(ss);
```

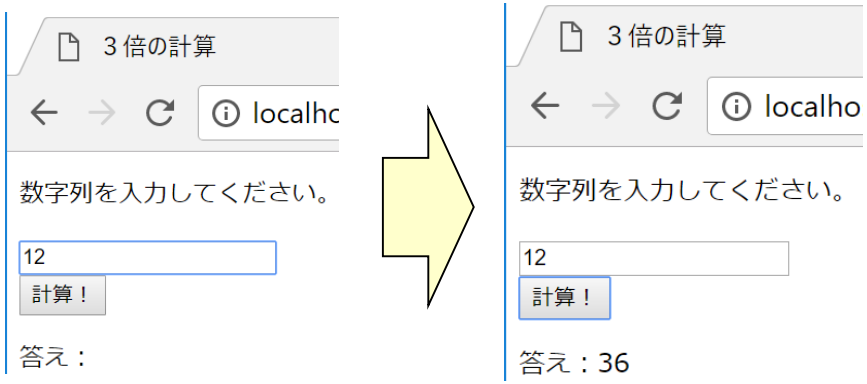
■浮動小数点数値への変換

```
var ss = document.form1.num.value;
var f = parseFloat(ss);
```

（parseInt()は関数ですが、これについては後述します。）

（6．11）例題

■数値の入力して、それの3倍の値を出力するプログラムを作成してください。



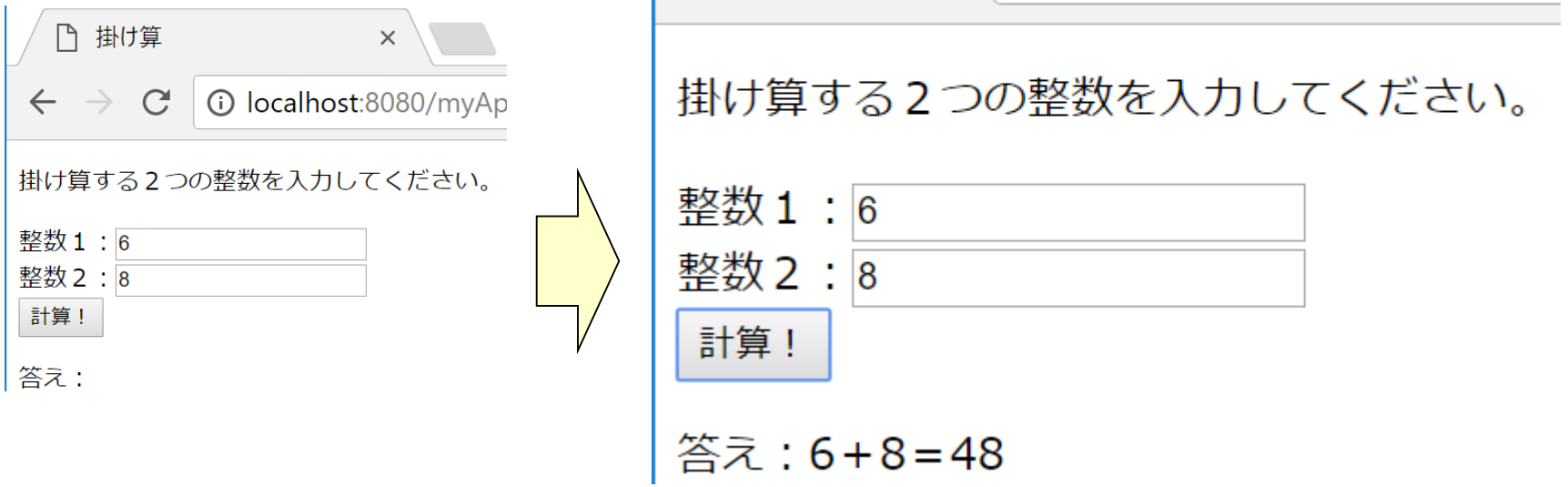
■解答例

```
<body>
<script type="text/javascript">
function echo(){
  var n_in = parseInt(document.getElementById("in_1").value);
  document.getElementById("ans_1").innerText="答え:"+n_in*3;
}
</script><p>数字列を入力してください。</p>
<form name="form1">
<input type="text" id="in_1" /><br />
<input type="button" value='計算！' onclick="echo();"/><br />
</form>
<p id="ans_1">答え:</p>
</body>
```

（6．12）課題

■課題（例題を参考にしてください）

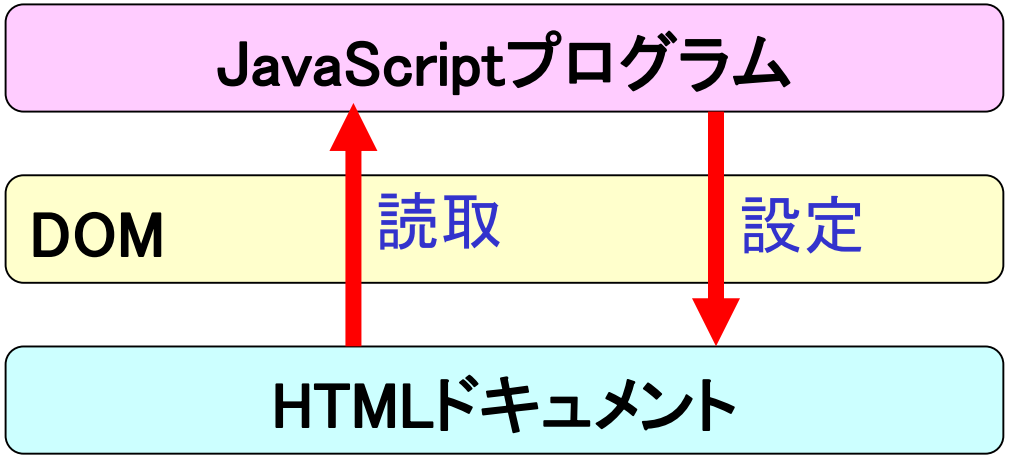
- 2つの数値を入力して、その積を出力するプログラムを作成してください。



（6．13）DOM(Document Object Model)

■DOMとは、HTMLドキュメントやXMLドキュメントをプログラムから利用するためのアプリケーションインターフェース（API）のことです。

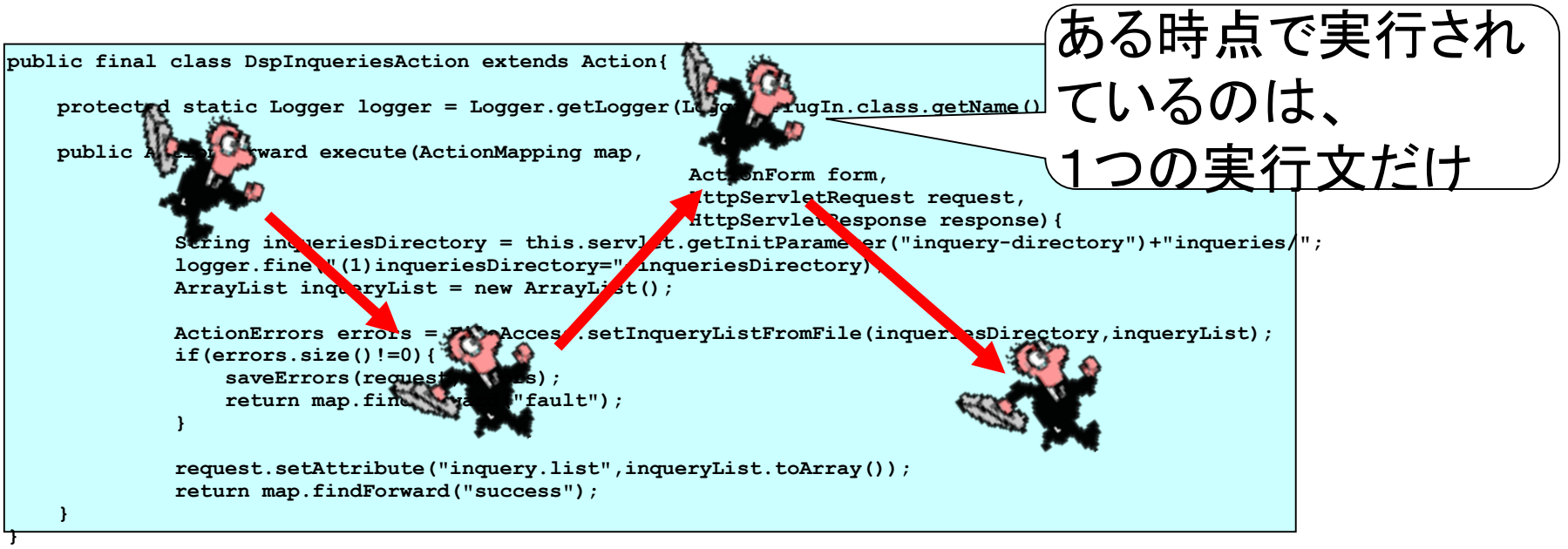
■ここまでJavaScriptプログラムからHTMLドキュメントの要素を操作してきましたが、これはDOMを利用しているということです。



(7)プログラムの実行

■プログラムの実行は複雑なようにも見えますが、「ある時点で実行している実行文はひとつだけ」という意味では単純とも言えます。

■すなわち、プログラムの実行は、プログラム中の実行箇所1箇所があちらこちらに走り回っているように考えることが出来ます。



（7．1）制御文

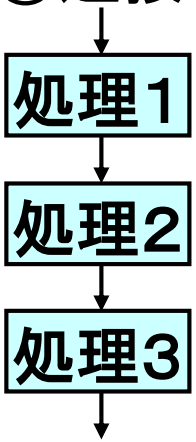
■実行箇所がどのように動くのかを決めるプログラムの要素を、制御文と言います。

■Javaには、次のような制御文があります。

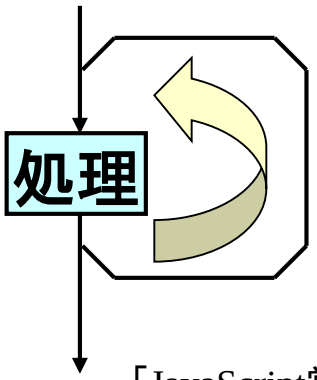
- `if while for do-while switch break continue return`

■プログラムの実行は複雑なようにも見えますが、制御文により実現される動きは、次の3つの組み合わせと考えることが出来ます（“構造化プログラミング”参照）。

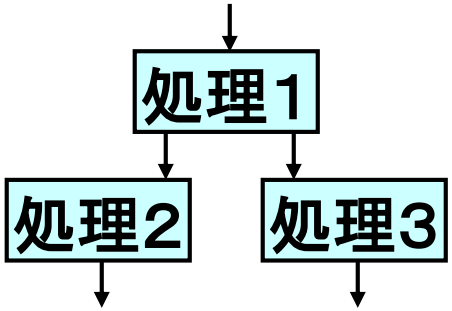
①連接：

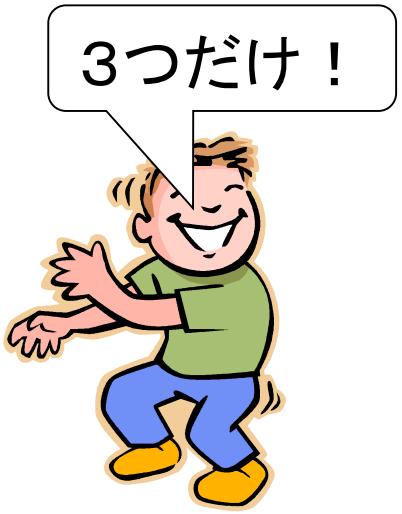


②繰り返し：



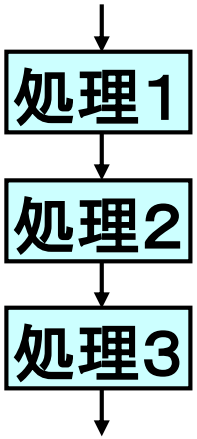
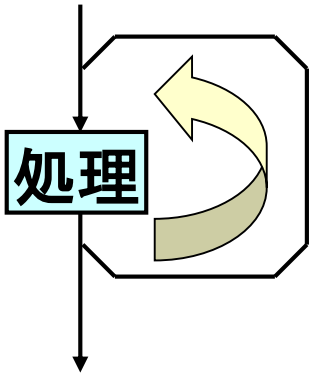
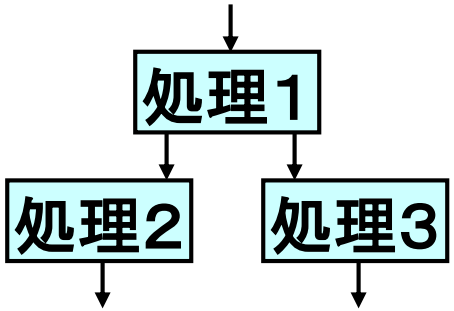
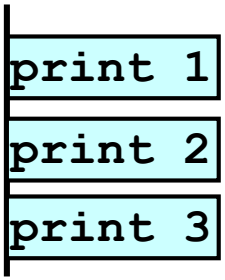
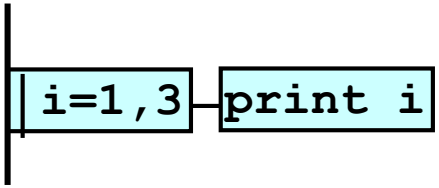
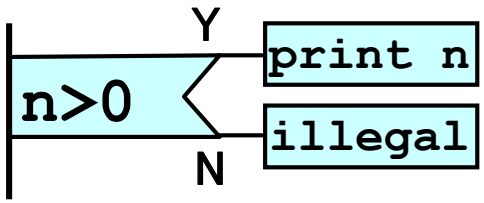
③分岐：





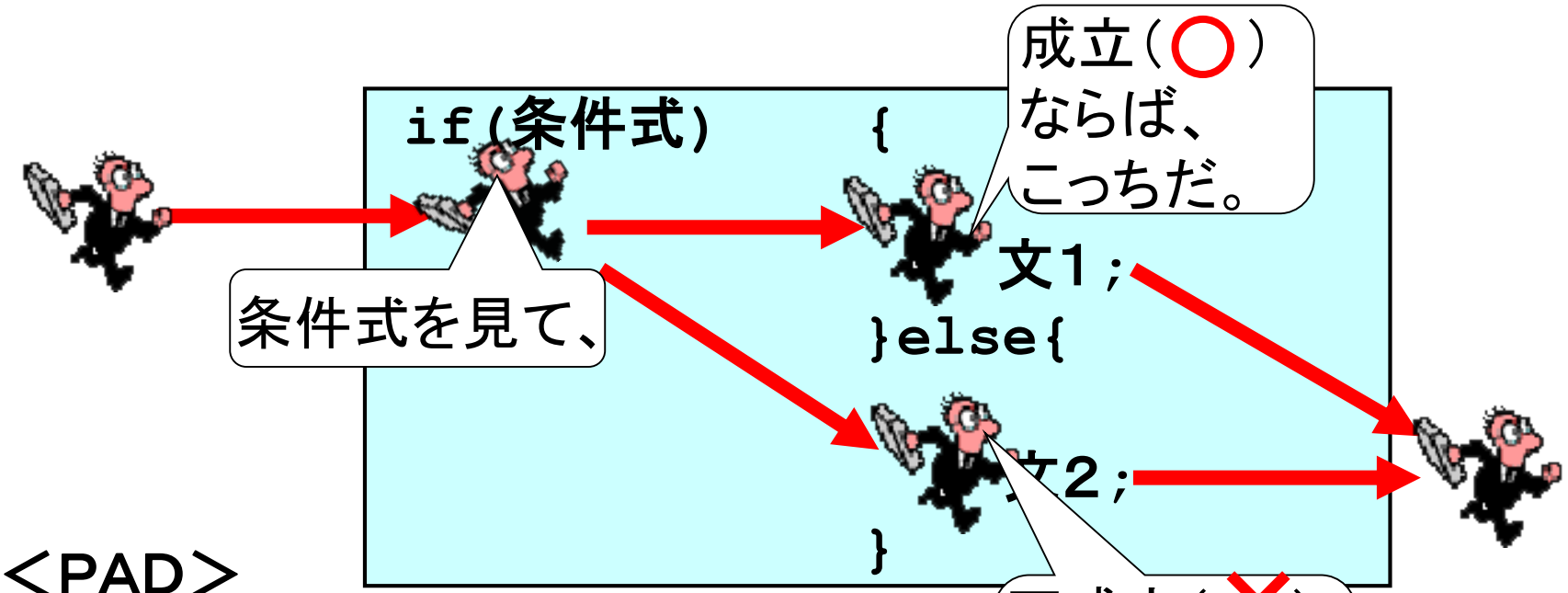
（7．2）PAD

■本資料では、PADと呼ばれる記法を制御文の説明時に併記しています。PADについて勉強されたい方は、別資料「山のあなたの空とおく―構造化プログラミングー」を参照してください。

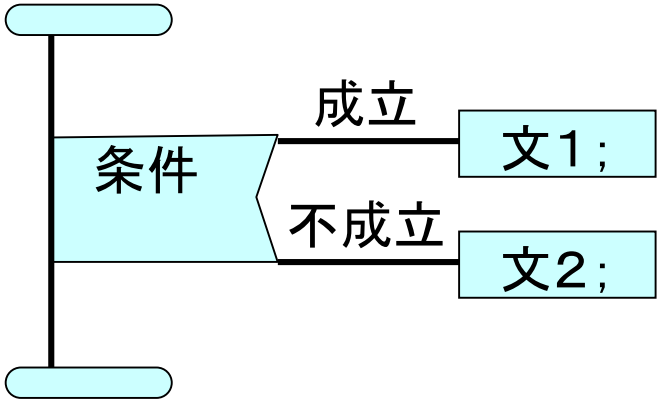
	①連接：	②繰り返し：	③分岐：
			
PAD			

（8）分岐、if文

■分岐では、”if”文がもっともよく用いられます。



＜PAD＞



(8．1)“if文”

■形式

```
if(条件式){
    文1;
}else{
    文2;
}
```

“{”と”}”に囲まれた範囲(＝ブロック)に、いくつ実行文があっても良い。

■例

＜プログラム＞

```
var a=0;
var b=0;
if (a==0){
  b=1;
}else{
  b=2;
}
donument.write(“b=”+b);
```

条件式「“a”は“0”である」

条件が成立(○)した場合、こちらを実行

条件が不成立(×)した場合、こちらを実行

＜出力＞

```
b=1
```

（8．2）条件

■関係演算子、等値演算子による条件

- 条件式は、関係演算子、等値演算子を用いて記述します。

関係演算子	比較	使用例	意味	xの値				
				3	4	5	6	7
<	より小さい	`if(x<5){`	xが5より小さいなら	○	○	×	×	×
<=	より小さいか等しい	`if(x<=5){`	xが5以下なら	○	○	○	×	×
>	より大きい	`if(x>5){`	xが5より大きいなら	×	×	×	○	○
>=	より大きいか等しい	`if(x>=5){`	xが5以上なら	×	×	○	○	○

等値演算子	比較	使用例	意味	xの値				
				3	4	5	6	7
==	等しい	`if(x==5){`	xが5なら	×	×	○	×	×
!=	等しくない	`if(x!=5){`	xが5でないなら	○	○	×	○	○

※　○：成立　　×：不成立

（8．3）論理型（boolean）

■条件が成立しているか、不成立であるかを表すデータのことを、論理型データ(boolean：ブーリアン)といいます。

- 成立していること（○）を、真（true）といいます。
- 不成立であること（×）を、偽（false）といいます。

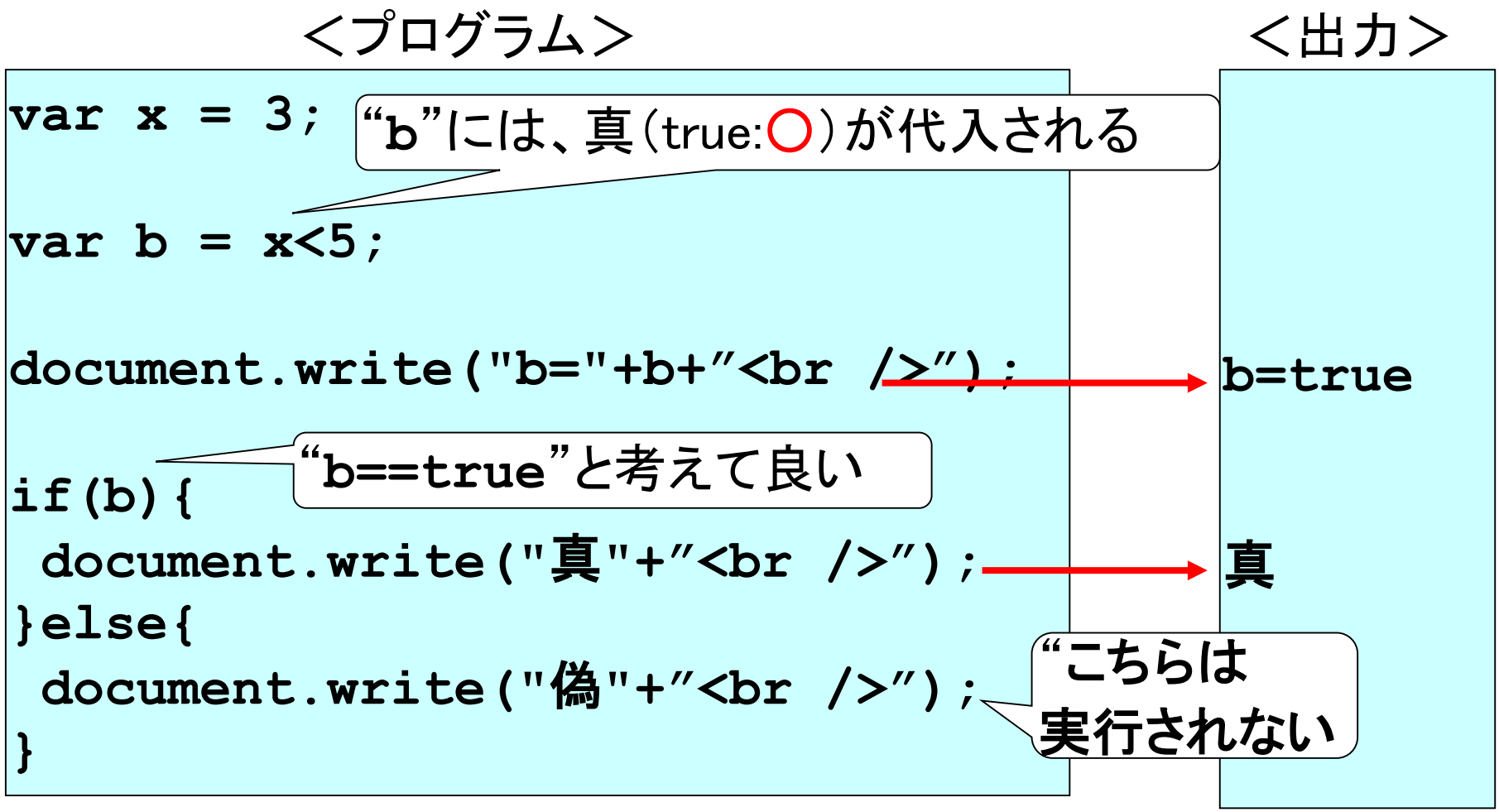
■論理型データを、整数型データと比較してみます。

	論理型(boolean)	整数型
変数の取る値	`true,false`	…,-3,-2,-1,0,1,2,3, …
可能な演算	論理演算 &&（かつ）、||（もしくは）	算術演算 ＋（足す）、－（引く）

■条件式は、論理型データの値を取るということになります。

`x=3;` ⇒ **“x<5”の値は“true”**

（8．4）論理型（boolean）の使用例



（8．5）論理演算子

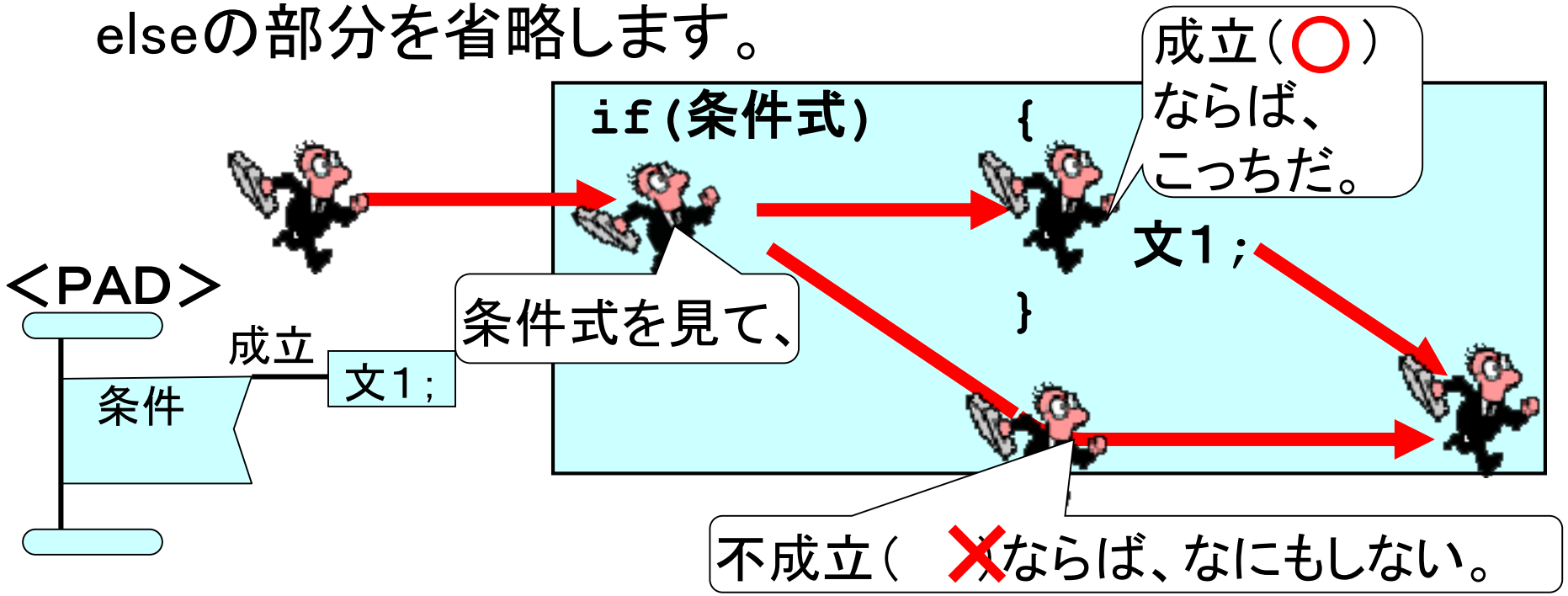
■2つ以上の条件を合わせて記述する場合などに、論理演算子を用います。

- 「今日は、13日の金曜日です」
  ⇒「今日は13日」、かつ(and)、「今日は金曜日」
- 「今日は、休日です。」
  ⇒「今日は日曜日」、または(or)、「今日は祝祭日」

論理演算子	演算	使用例 意味	xの値 3	4	5	6	7
&&	論理積 （かつ、and）	if(3<x && x<6){ xが3より大きい、かつ、xが6より小さい	×	○	○	×	×
||	論理和 （または、or）	if(x<5 || 6<x){ xが5より小さい、または、xが6より大きい	○	○	×	×	○
!	否定 （ではない、not）	if(!(x<5 || 6<x)){ xが5より小さい、または、xが6より大きいではない	×	×	○	○	×

(8.6)“else”の省略

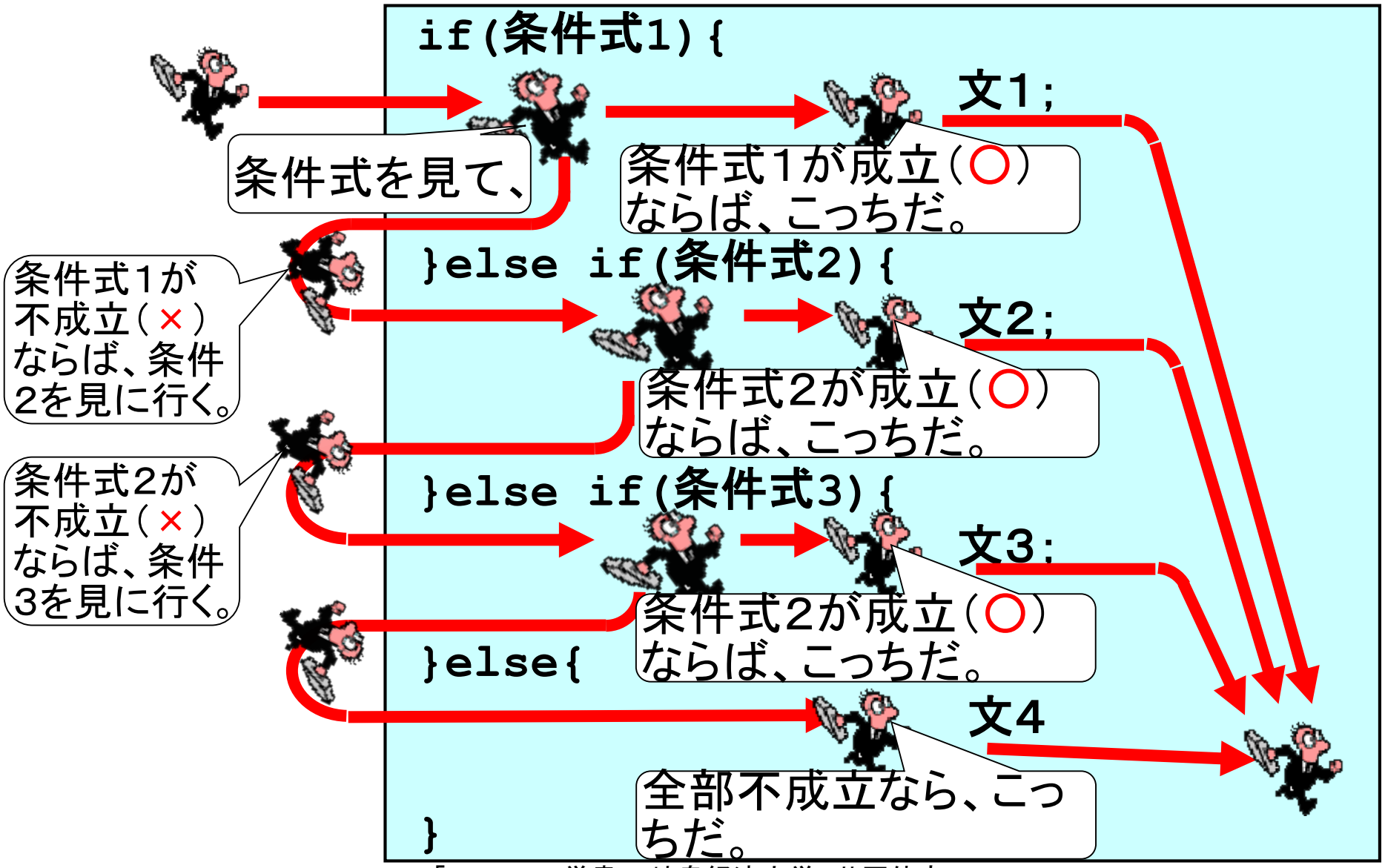
■条件式が成立しないときに何もしないのであれば、elseの部分を省略します。



■例：1000円以上ならば、100円引き

```
if(x>=1000){
   x = x - 100;
}
```

(8.7.1)“else if”


if(条件式1){
文1;
条件式を見て、
条件式1が成立(○)ならば、こっちだ。
条件式1が不成立(×)ならば、条件2を見に行く。
}else if(条件式2){
文2;
条件式2が成立(○)ならば、こっちだ。
条件式2が不成立(×)ならば、条件3を見に行く。
}else if(条件式3){
文3;
条件式2が成立(○)ならば、こっちだ。
}else{
文4
全部不成立なら、こっちだ。
}

（8．7．2）“else if”の例

■例：

- 10,000円以上ならば、1,000円値引き、
- 5,000円以上ならば、400円円値引き、
- 3,000円以上ならば、100円値引き、
- 1,000円以上ならば、10円値引き

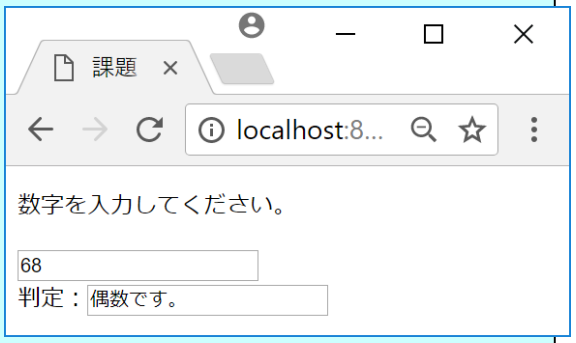
```
if(x>=10000){
  x = x - 1000;
}else if(x>=5000){
  x = x - 400;
}else if(x>=3000){
  x = x - 100;
}else if(x>=1000){
  x = x - 10;
}
```

（8．8）例題：奇数・偶数

■数字の入力を読み込んで、偶数か奇数かを出力するプログラムを作成してください。

■解答例：

```
<script type="text/javascript">
function judge(){
   var d = parseInt(document.form1.number.value);
   if(d%2==0){
      document.form1.answer.value="偶数です。";
   }else{
      document.form1.answer.value="奇数です。";
   }
}
</script>

<p>数字を入力してください。</p>
<form name="form1">
<input type="text" name="number" onchange="judge();"/><br />
判定:<input type="text" name="answer" />
</form>
```



（８．９）例題:うるう年の条件の整理

■うるう（閏）年である条件

- （１）４で割れる年は、うるう年
  - ２００４年はうるう年。
  - ２００５年は平年（うるう年でない）。
- （２）上記（１）の例外として、１００で割り切れる年は平年（うるう年でない）。
  - １９００年は平年（うるう年でない）。
  - １９０４年はうるう年。」
- （３）上記（２）の例外として、４００で割り切れる年はうるう年。
  - ２０００年はうるう年。
  - １９００年は平年（うるう年でない）。

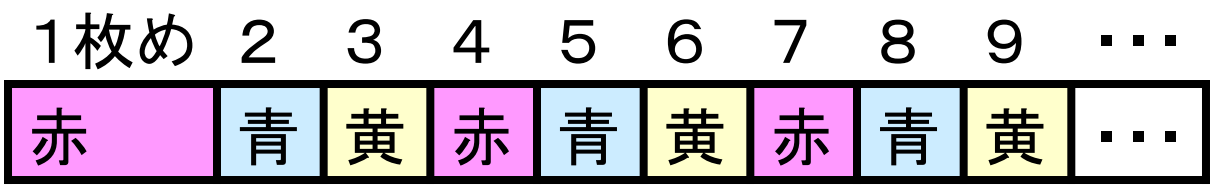
■同じ条件の別の言い方

- [1]４００で割れる年はうるう年。
- [2]上記[1] ではなくて、１００で割れる年は平年（うるう年でない）
- [3]上記[1][2]でなく、４で割れる年はうるう年。
- [4]上記[1][2][3]でない場合は平年（うるう年でない）。

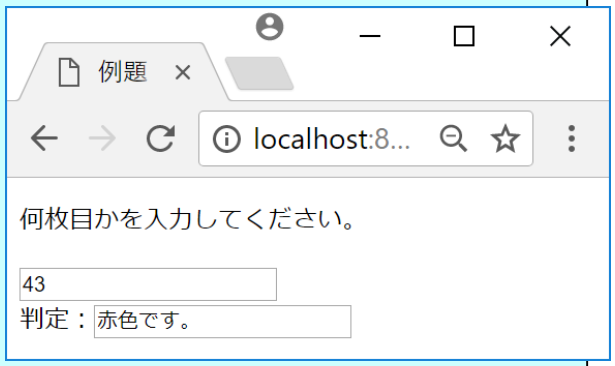
（8．10）例題：敷石

■次のように並んでいる舗道の敷石について、敷石が何枚目かを入力して、その色を出力するプログラムを作成してください。

■解答例：




```
<script type="text/javascript">
function judge(){
   var n = parseInt(document.form1.number.value);
   var color='';
   if(n%3==1){
      color='赤';
   }else if(n%3==2){
      color='青';
   }else{
      color='黄';
   }
   document.form1.answer.value=color+"色です。";
}
</script>
<p>何枚目かを入力してください。</p>
<form name="form1">
<input type="text" name="number" onchange="judge();"/><br />
判定:<input type="text" name="answer" />
</form>
```

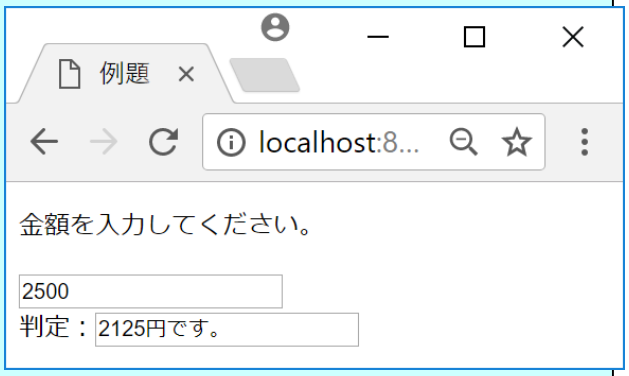


（8．11）例題：値引き

■金額（円）を入力して、3000円以上なら2割引き、2000円以上なら1割5部引き、1000円以上なら1割引きの額を出力するプログラムを作成してください。

■解答例：

```
<script type="text/javascript">
function judge(){
   var price = parseInt(document.form1.price.value);
   if(price>=3000){
      price*=0.8;
   }else if(price>=2000){
      price*=0.85;
   }else if(price>=1000){
      price*=0.9;
   }
   document.form1.answer.value=price+"円です。";
}

</script>
<p>金額を入力してください。</p>
<form name="form1">
<input type="text" name="price" onchange="judge();"/><br />
判定:<input type="text" name="answer" />
</form>
```

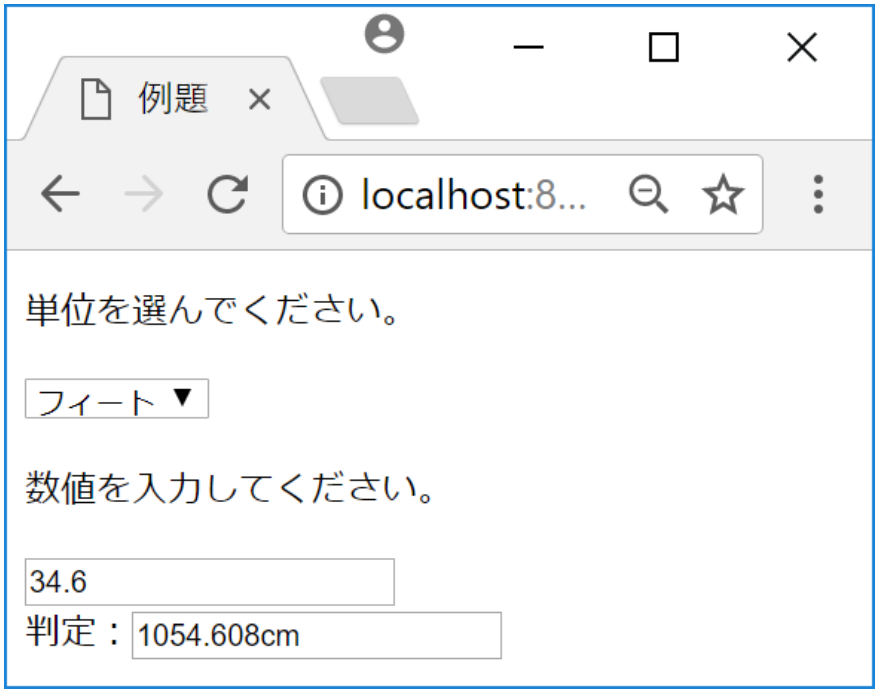


（8．12）例題：単位の換算

■長さの単位、インチ、フィート、ヤードは、cm(センチメール)に換算すると次のとおりとなります。

	1インチ（in）	1フィート(ft)	1ヤード（yd）
センチ（cm）換算	2．54	30．48	91．44

■単位（in,ft,yd）と数値とを入力して、そのセンチメートル換算の値を出力するプログラムを作成してください。



（８．１２）例題：続き

■解答例：

```
<script type="text/javascript">
function judge(){
  var unit = document.form1.unit.value;
  var number = parseFloat(document.form1.number.value);
  var mul=0;
  if(unit=='in'){
    mul=2.54;
  }else if(unit=='ft'){
    mul=30.48;
  }else if(unit=='yd'){
    mul=91.44;
  }
  document.form1.answer.value=number*mul+"cm";
}
</script>
<form name="form1">
<p>単位を選んでください。</p>
<select name='unit' onchange="judge();">
<option value='in'>インチ</option>
<option value='ft'>フィート</option>
<option value='yd'>ヤード</option>
</select>
<br />
<p>数値を入力してください。</p>
<input type="text" name="number" onchange="judge();" value="1"/><br />
判定:<input type="text" name="answer" />
</form>
```

（８．１３）課題

■（８．１３．１）課題

- 数字の入力を読み込んで、次の条件の“いずれか”を満たす場合に、“条件を満たす”と出力するプログラムを作成してください。
  - ◆入力した数字は、３で割り切れる。
  - ◆入力した数字を用いて１２８を割ると、あまりが０になる。

■（８．１３．２）課題

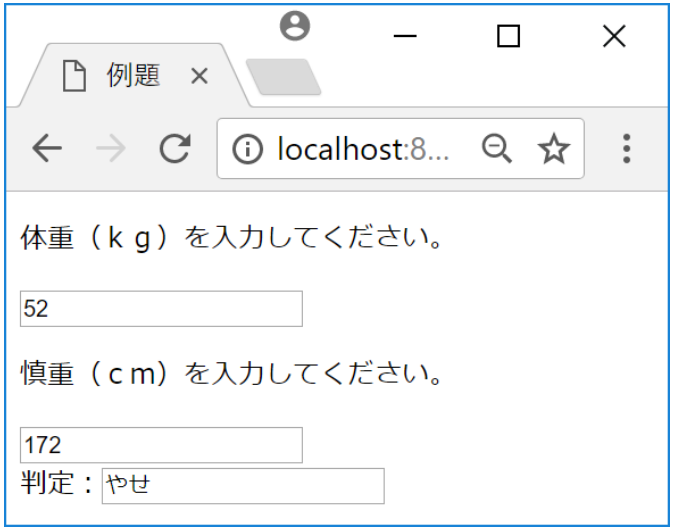
- 今月の日にち（ex. 3, 15, 27）の数字の入力を読み込んで、曜日を出力するプログラムを作成してください。
- 配列を用いる方法も考えてください。

（8．14）課題

■課題

- 肥満度の判定の1つに、BMI判定があります。
  - ◆指標＝体重(kg)／（身長(m)×身長（m））
  - ◆指標に対して、次の表にて判定する。

指標	判定
18.5未満	やせ
18.5～25未満	標準
25～30未満	肥満
30以上	高度肥満



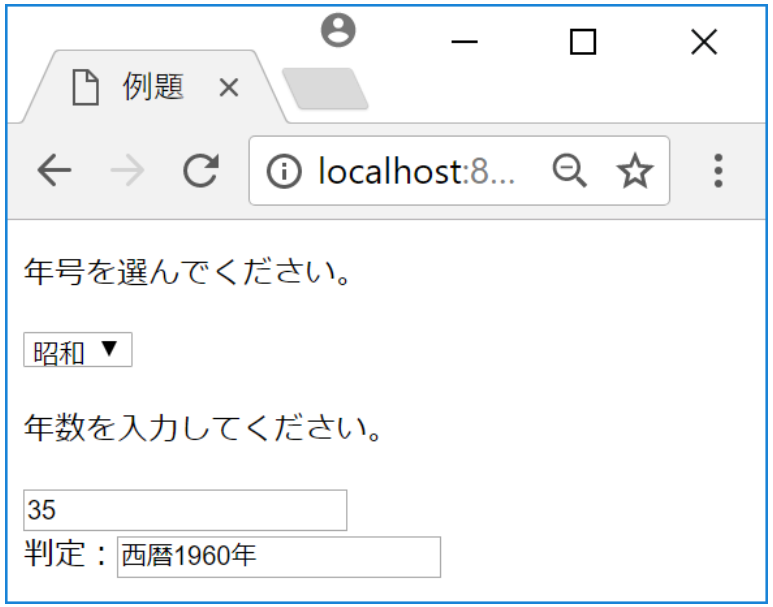
- 判定を出力するプログラムを作成してください。

（8．15）課題

■課題

- 元号（明治を、大正、昭和、平成）と年（1、2、...）とを別々に入力して、西暦を出力するプログラムを作成してください。

明治元年	1868年
大正元年	1912年
昭和元年	1926年
平成元年	1989年

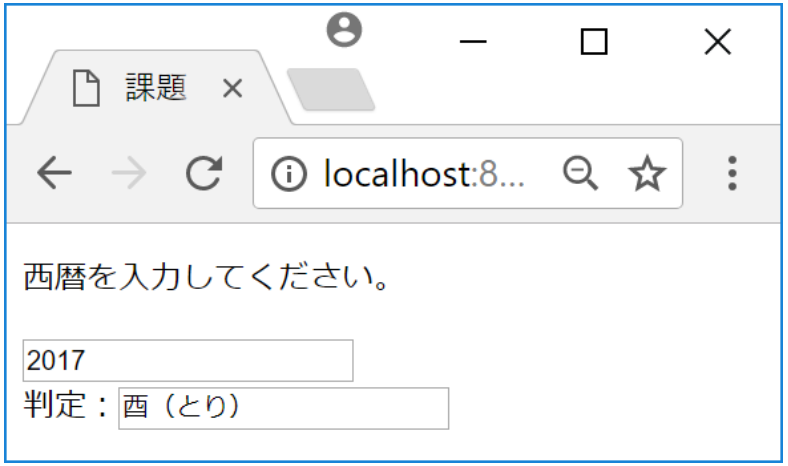


（8．16）課題

■課題

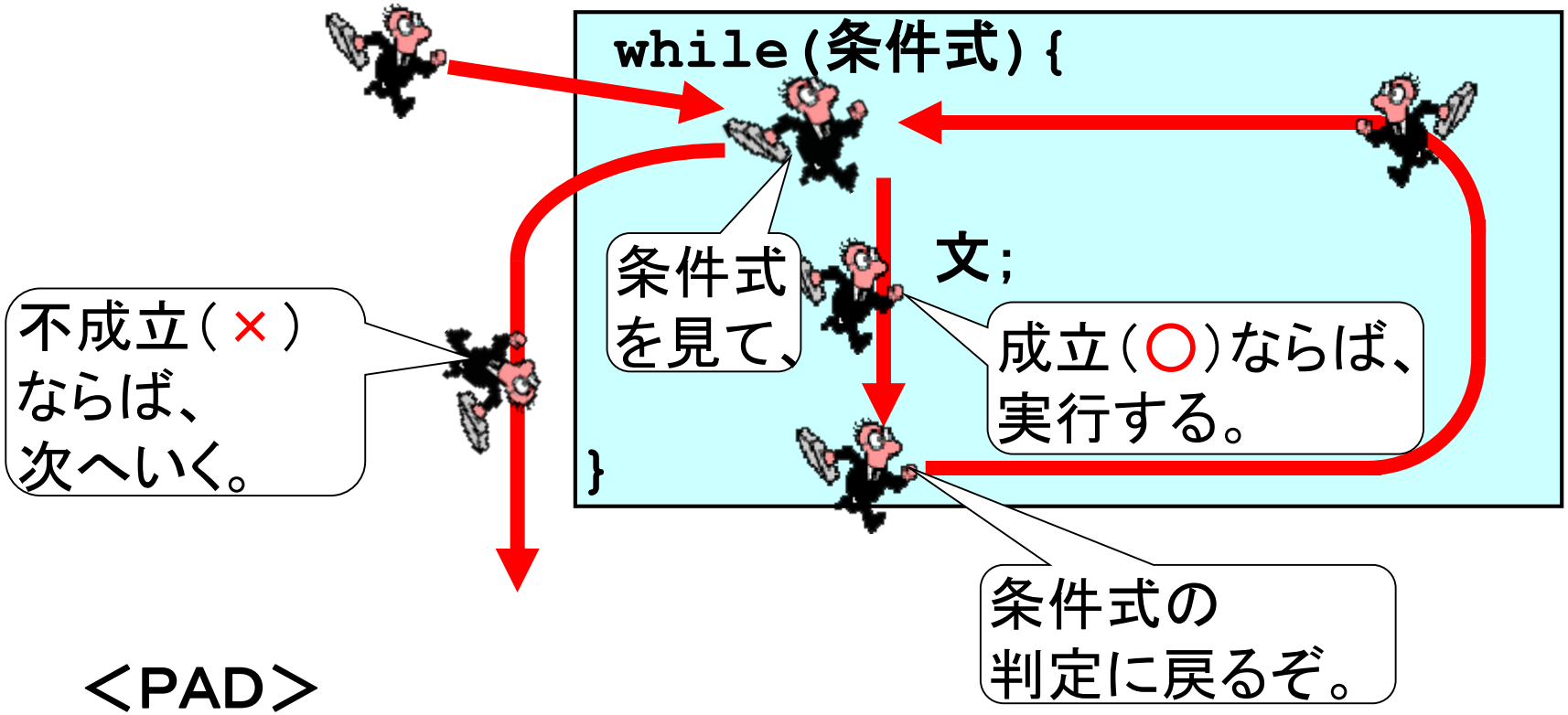
- 近年の干支は下の表のとおり。
- 西暦（1995年以降）を表す数字の入力を読み込んで、干支を出力するプログラムを作成してください。
- ちなみに、2004を12で割ると、あまりが0になります。
- 配列を使う方法も考えてください。

1995	亥（い）	2001	巳（み）
1996	子（ね）	2002	午（うま）
1997	丑（うし）	2003	未（ひつじ）
1998	寅（とら）	2004	申（さる）
1999	卯（う）	2005	酉（とり）
2000	辰（たつ）	2006	戌（いぬ）

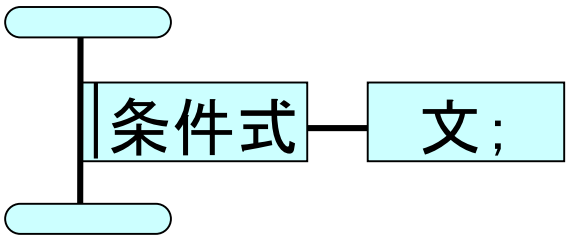


（9．1．1）while文

■最初に基本的な繰り返しwhile文を見ます。



＜PAD＞



（9．1．2）while文の形式・例

■形式

```
while(条件式){
      文;
}
```

■例

＜プログラム＞

```
var a=3;
while(a>0){

   document.write('a='+a+'<br>');
   a--;
}
```

“a”が“0”より大きい間、繰り返し

“a=a-1”と同じ。スライド(5.2)参照。

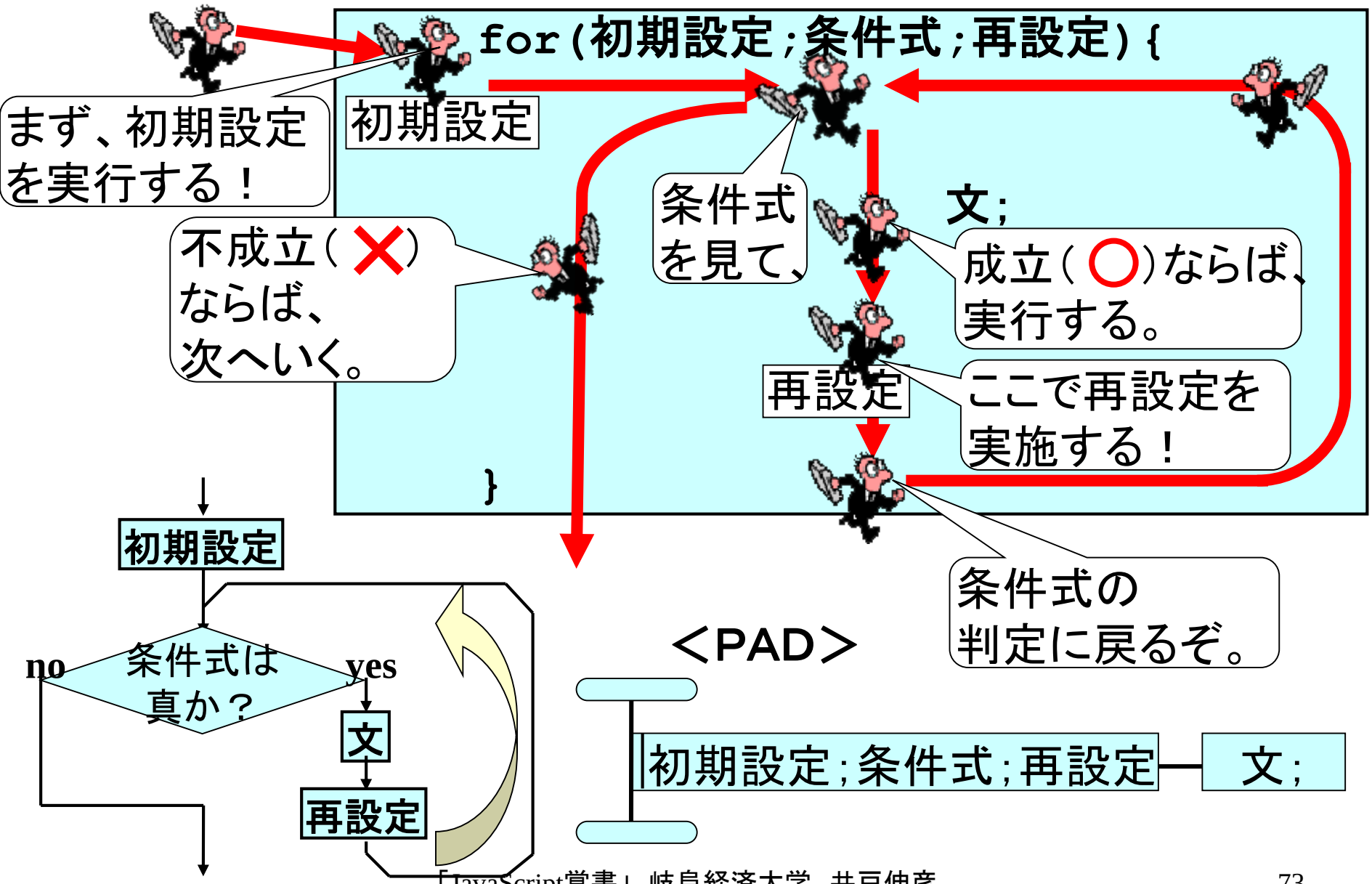
＜出力＞

```
a=3
a=2
a=1
```

■注意

- 繰り返される部分に、条件を変化させていく文（上記の例では、“a--;”）がないといつまでも繰り返しを続けるという事態（無限ループ）になる。

（9．2．1）for文


for(初期設定;条件式;再設定){
初期設定
まず、初期設定を実行する！
条件式を見て、
文;
成立（○）ならば、実行する。
不成立（×）ならば、次へいく。
再設定
ここで再設定を実施する！
}
条件式の判定に戻るぞ。
初期設定
条件式は真か？
no
yes
文
再設定
＜PAD＞
初期設定;条件式;再設定
文;

（9．2．2）なぜfor文を使うのか？

■while文との比較

- 次の2つのプログラムは、まったく同じ動きとなります。

```
for(文1;条件式;文3){
        文2;
}
```

同じ

```
文1;
while(条件式){
        文2;
        文3;
}
```

※　文1と文3は、複合文ではないこと。
　　文2は“continue”を含まないこと。

■なぜfor文を使うのか？

- 上記の例で、文1を「初期化」、文3を「再設定」と捉えることで、繰り返しの条件が明瞭になります。

条件が明瞭

```
for(a=3;a>0;a--){
    文;
}
```

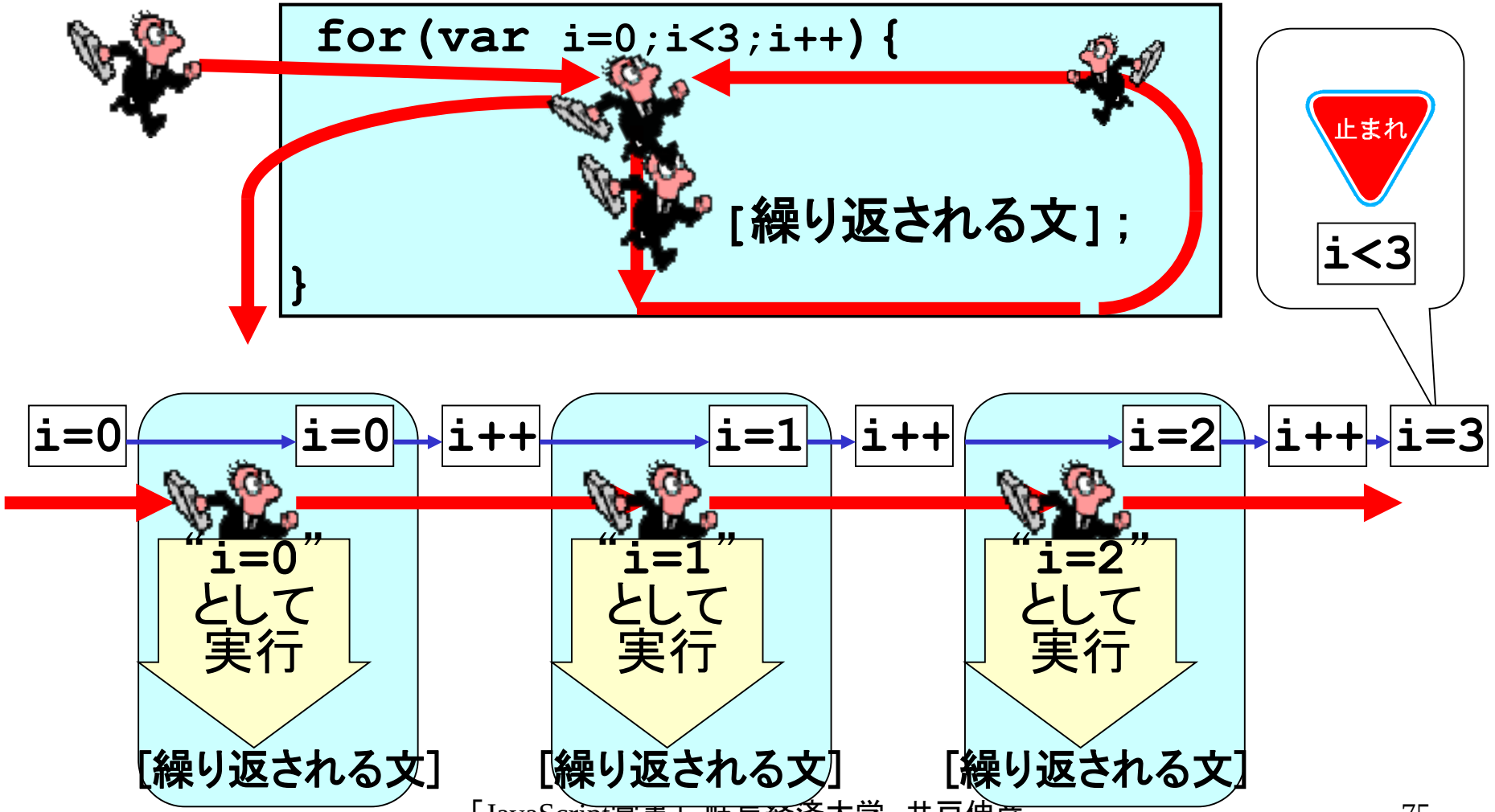
同じ

```
var a=3;
while(a>0){
   文;
   a--;
}
```

ここまでが長くなる。

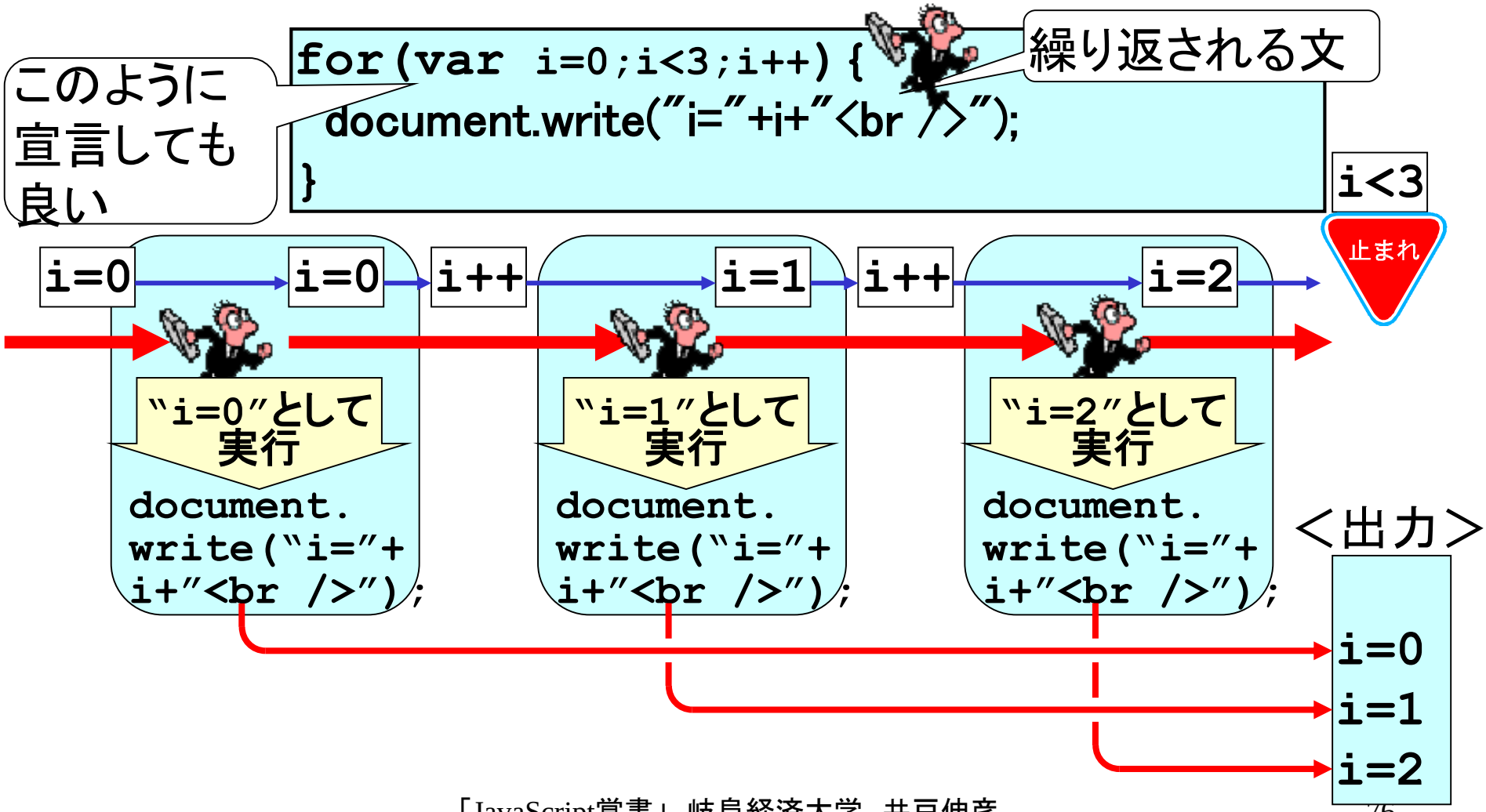
（9．2．3）for文による繰り返し

■for文の中で使われる変数は、繰り返し方を決めることになり、よく“i”（“iteration”の“i”）が使われます。



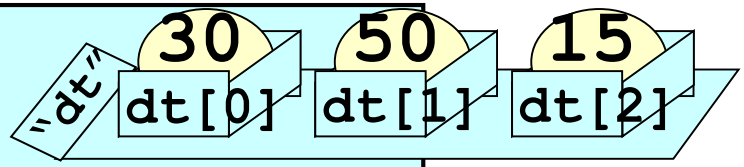
（9．3．1）for文による繰り返しの例

■for文の中で使われる変数は、繰り返し方を決めることになり、よく“i”（“iteration”の“i”）が使われます。

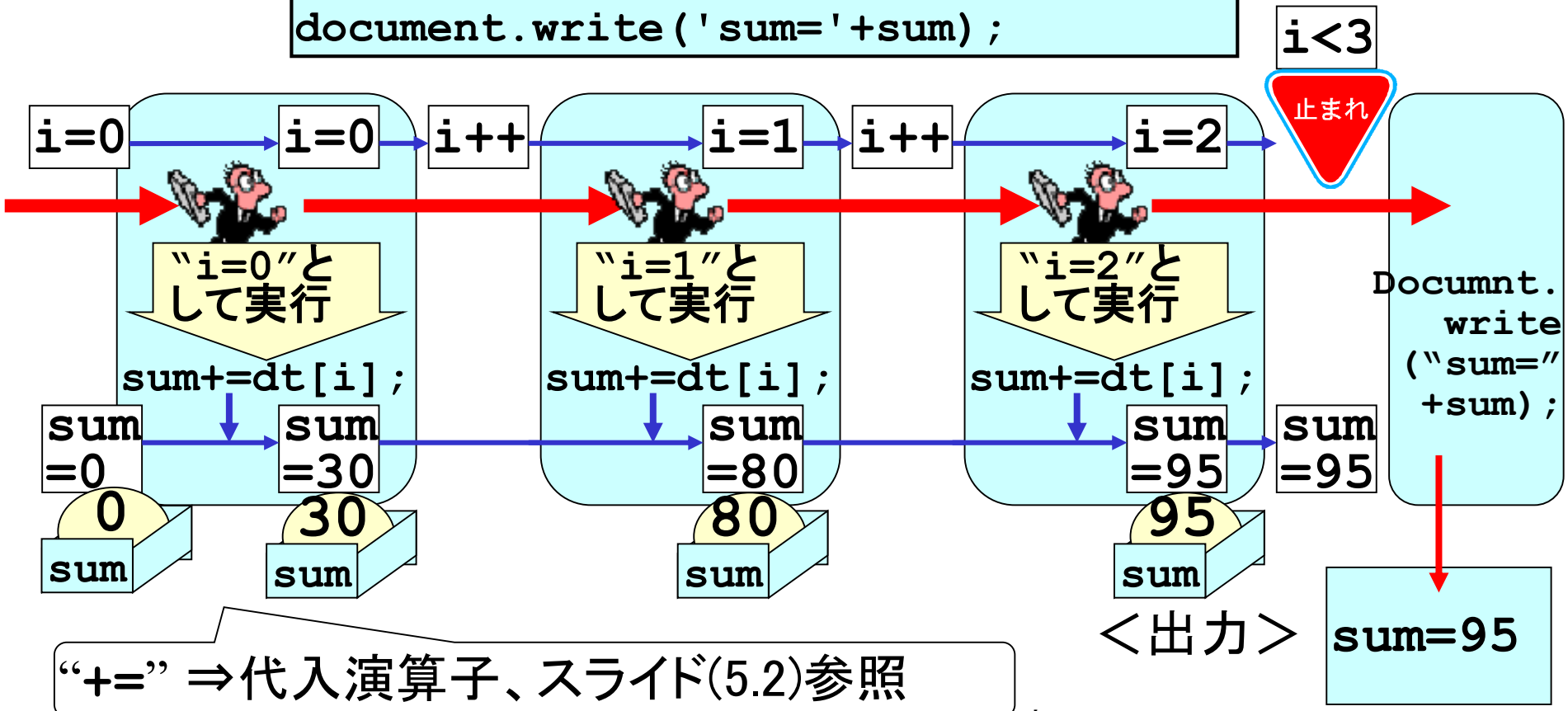


(9．3．2)配列の合計

```
var dt = [30,50,15];
var sum = 0;
for(var i=0;i<3;i++){
   sum += dt[i];
}
document.write('sum='+sum);
```



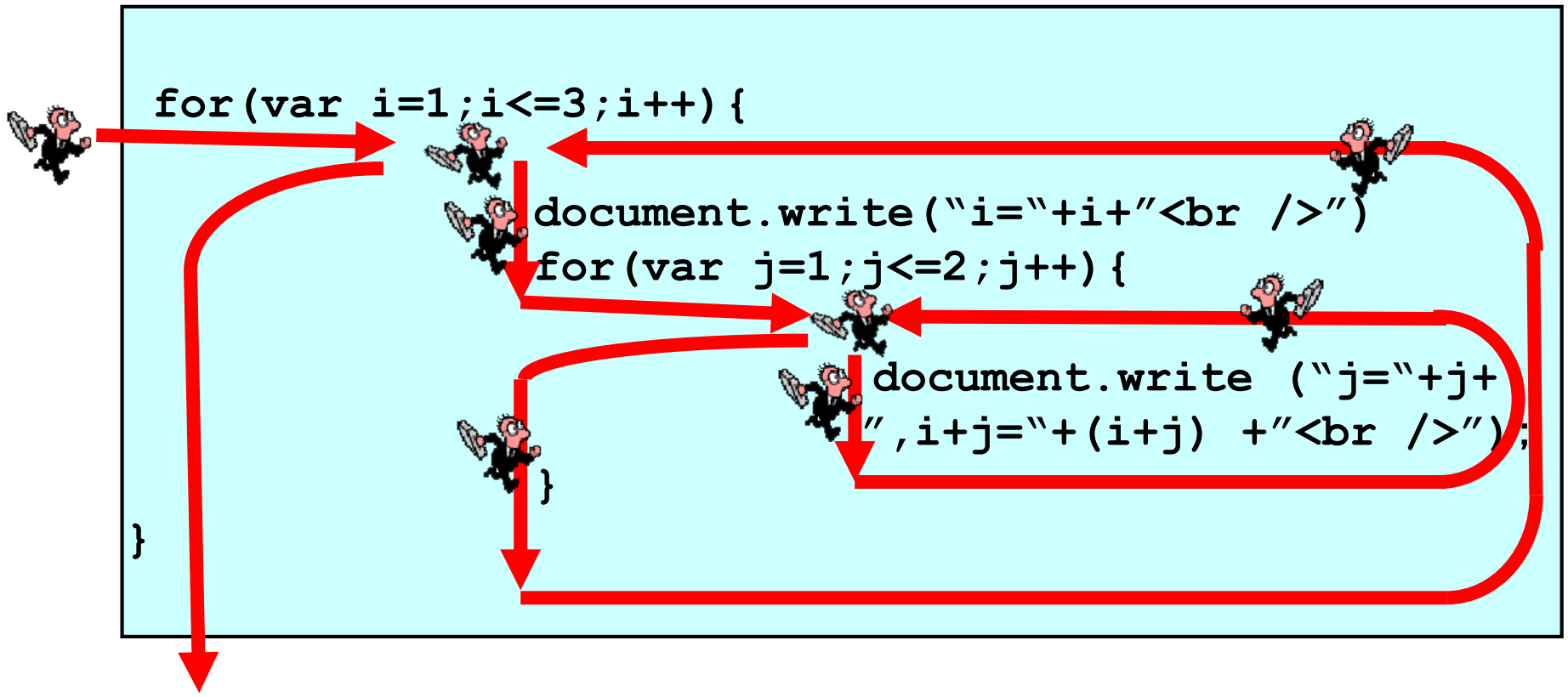
繰り返される文



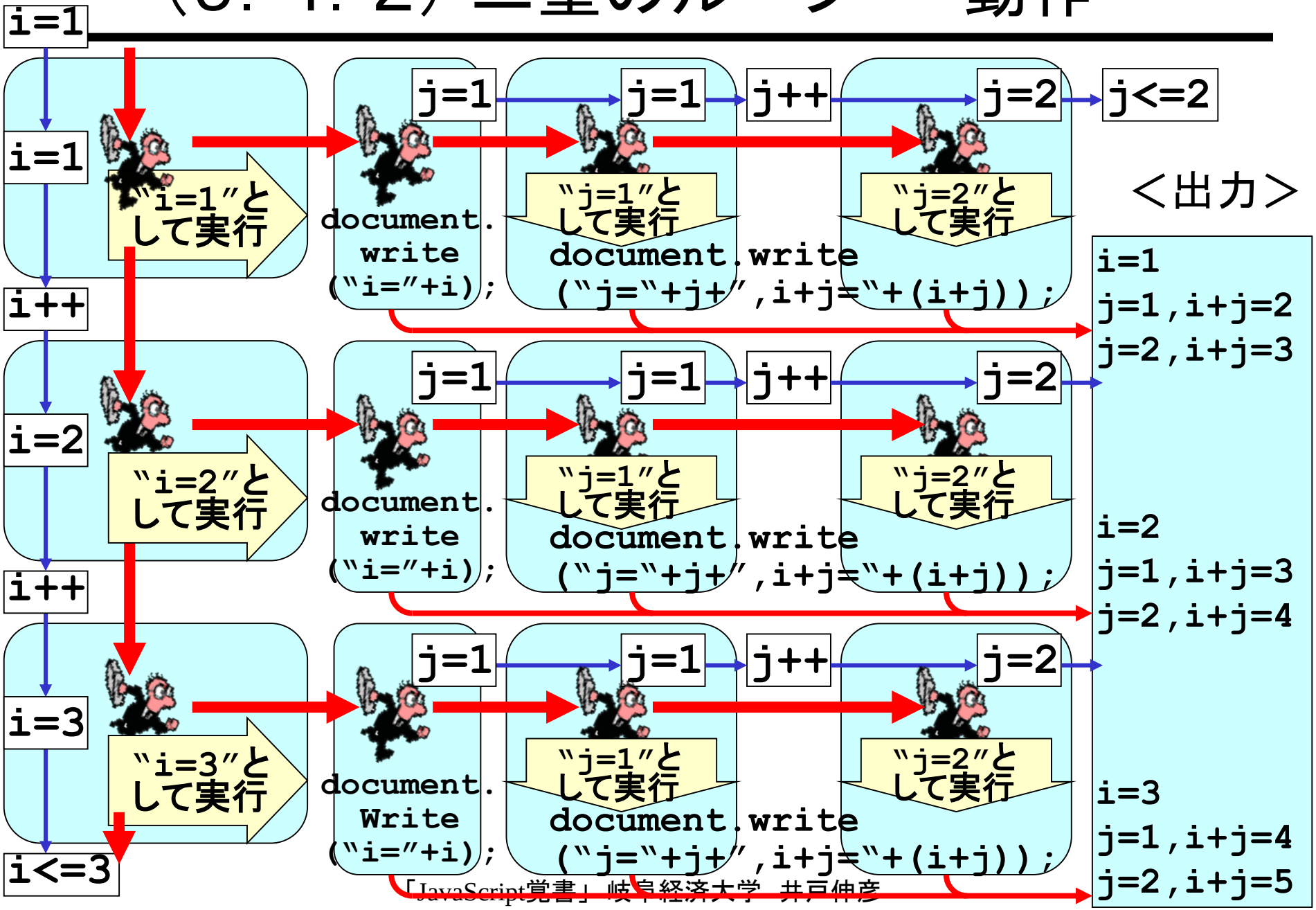
"+=" ⇒代入演算子、スライド(5.2)参照

（9．4．1）二重のループ

■for文の中にfor文が現れる形も、同じように理解出来ます。



（9．4．2）二重のループ　ー動作ー



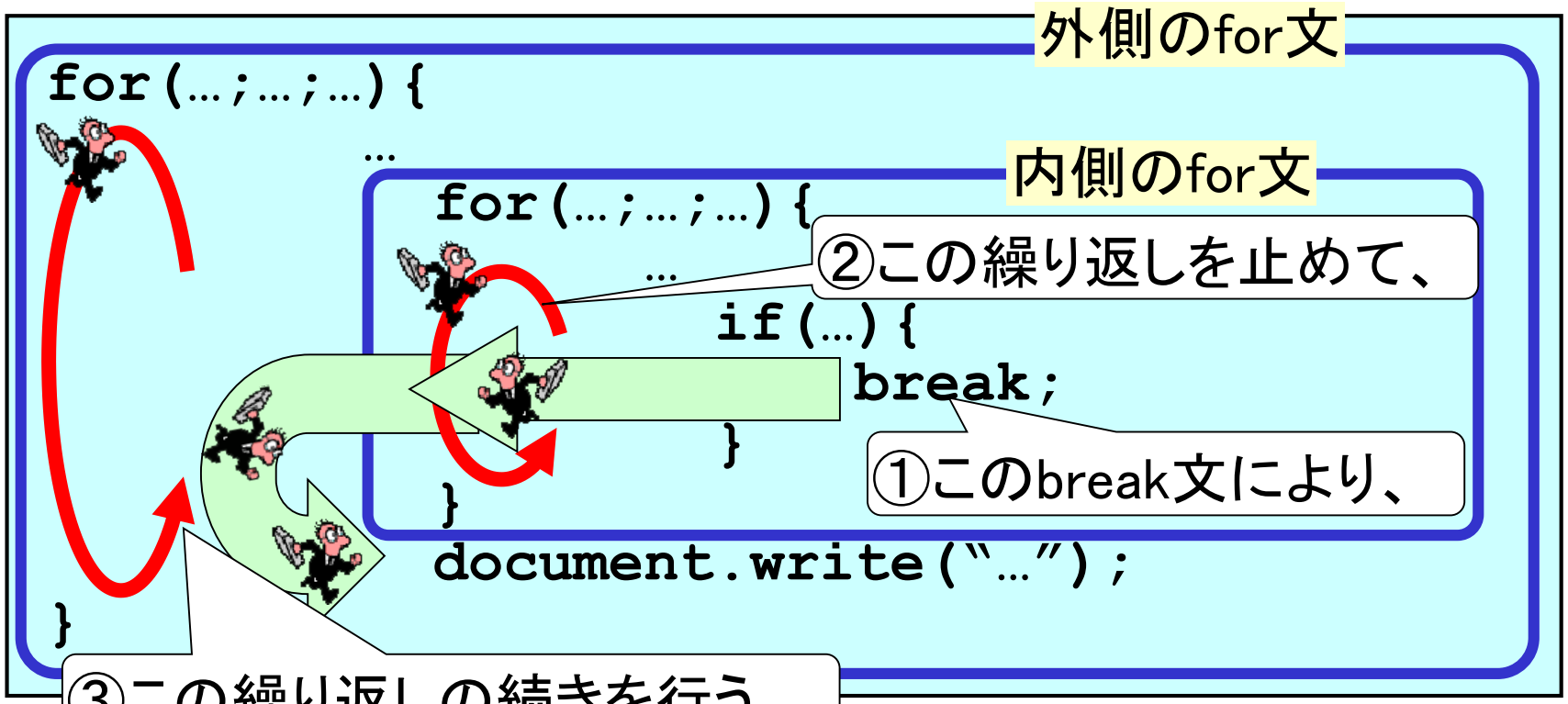
（９．４．３）while文の使い道

- 繰り返しの条件が、回数として明確な場合には、今まで見てきたとおり、for文を使うことが良いようです。
- 何回繰り返すか分からないような場合には使います。わざとらしい例ですが、次のようなプログラムではwhile文を使います。

```
<body onload="init();">
<script type="text/javascript">
function init(){
  a='';
  while(confirm('＊を増やしますか？')){
    a+='*';
  }
  var place=document.getElementById('place');
  place.textContent=a;
}
</script>
<p id='place'></p>
</body>
```

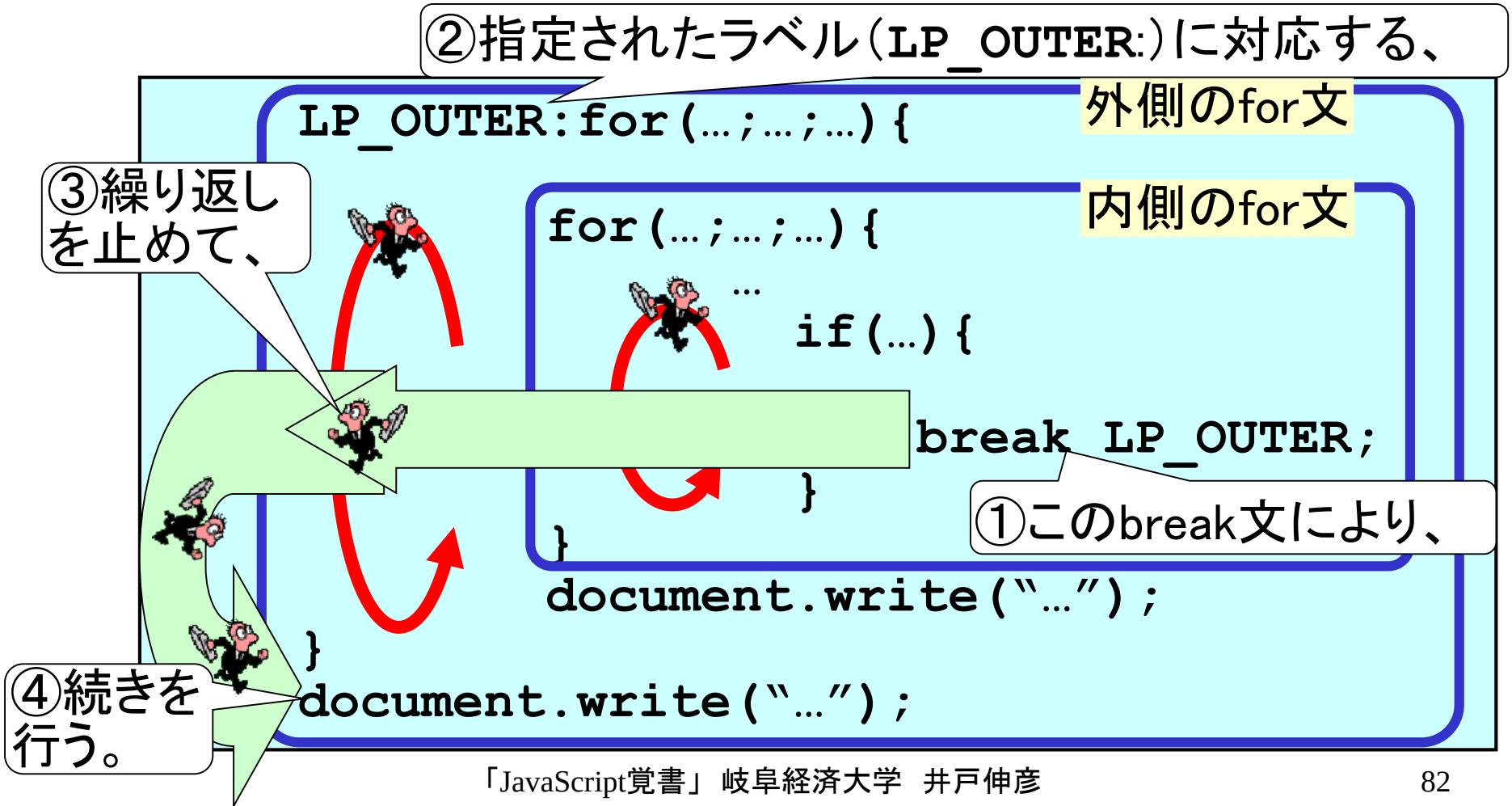
（9．5．1）break文で中止する繰り返し

■**break**文により終了される繰り返しは、その**break**文を含む一番小さい繰り返し（＝一番内側の**for**文、**while**文）となります。すなわち、2重ループなどでbreak文を実行すると、内側の繰り返しを止めて、外側の繰り返しを続けることになります。



（9．5．2）ラベルつきbreak

■一番内側の繰り返し以外の繰り返しを止めるには、繰り返しにラベルをつけ、そのラベルを指定したbreak文（ラベルつきbreak文）を用います。



（９．６）例題：最大値

■次のように初期化された配列のそれぞれの値の、最大を求めるプログラムを作成してください。

```
var dt[] = {5,4,12,23,7};
```

■解答例

```
1: var dt=[5,4,12,23,7];
2: var max=dt[0];
3: for(var i=1;i<dt.length;i++){
4:   if(max<dt[i]){
5:     max=dt[i];
6:   }
7: }
8: document.write("最大値＝"+max);
```

（９．６．１）解答例のプログラムの見方

```
1:  var dt = [5,4,12,23,7];

2:  var max=dt[0];

3:  for(var i=1;i<dt.length;i++){

4:     if(max<dt[i]){

5:        max=dt[i];

6:     }
7:  }
8:  document.write("最大値="+max);:
```

ひとまず、最初の要素を最大値とする。

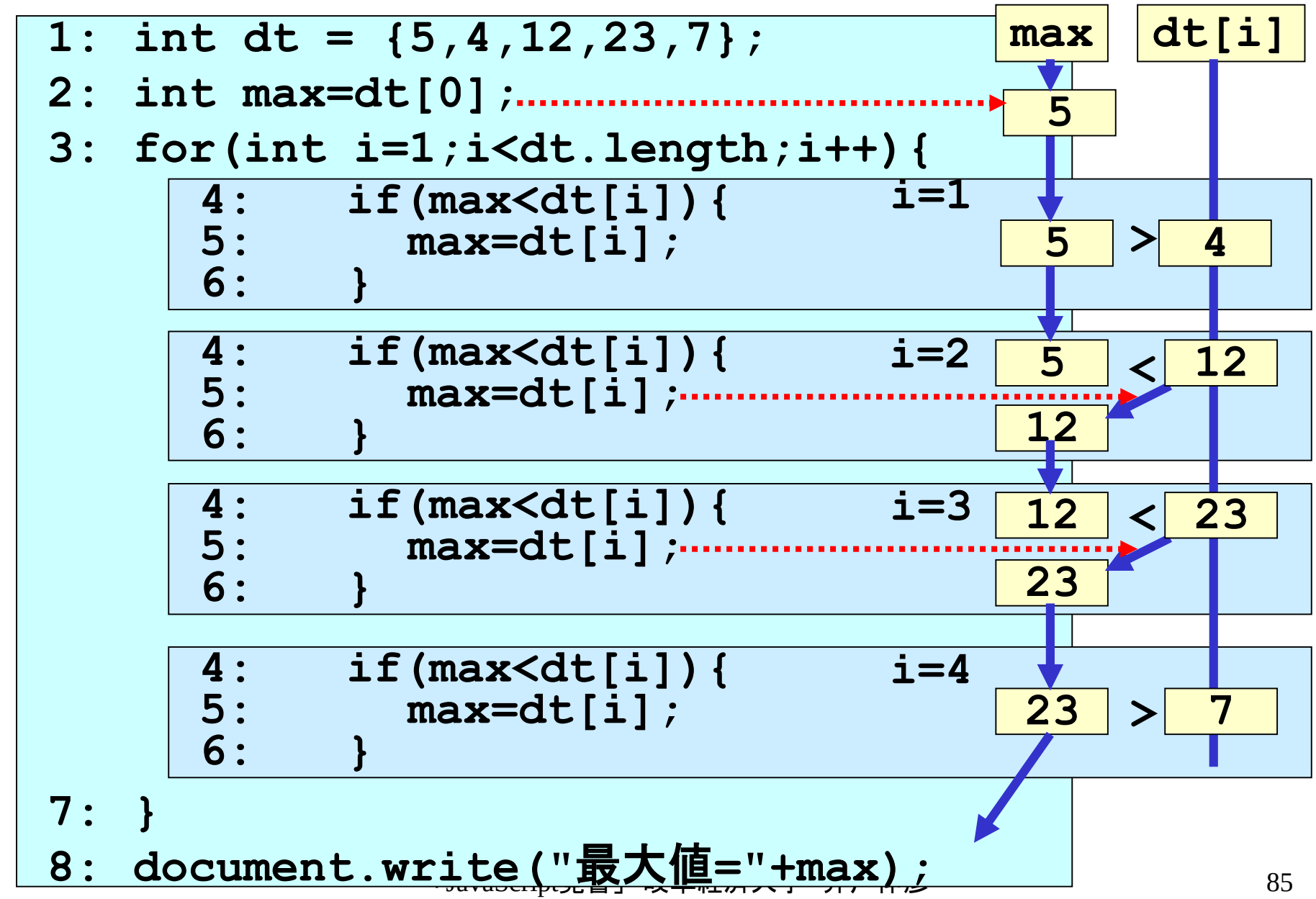
２つめ以降の要素について、順次調べる

もし、要素が最大値よりも大きければ

その要素を最大値とする

（9．6．2）動作

```
1:  int dt = {5,4,12,23,7};
2:  int max=dt[0];
3:  for(int i=1;i<dt.length;i++){
        4:    if(max<dt[i]){
        5:       max=dt[i];
        6:    }
        4:    if(max<dt[i]){
        5:       max=dt[i];
        6:    }
        4:    if(max<dt[i]){
        5:       max=dt[i];
        6:    }
        4:    if(max<dt[i]){
        5:       max=dt[i];
        6:    }
7:  }
8:  document.write("最大値="+max);
```



（９．７）例題：文字絵

＜出力＞

```
#####
*####
**###
***##
****#
*****
```

■右のような出力を行うプログラムを
作成してください（for文を使ってください）。

■解答例：

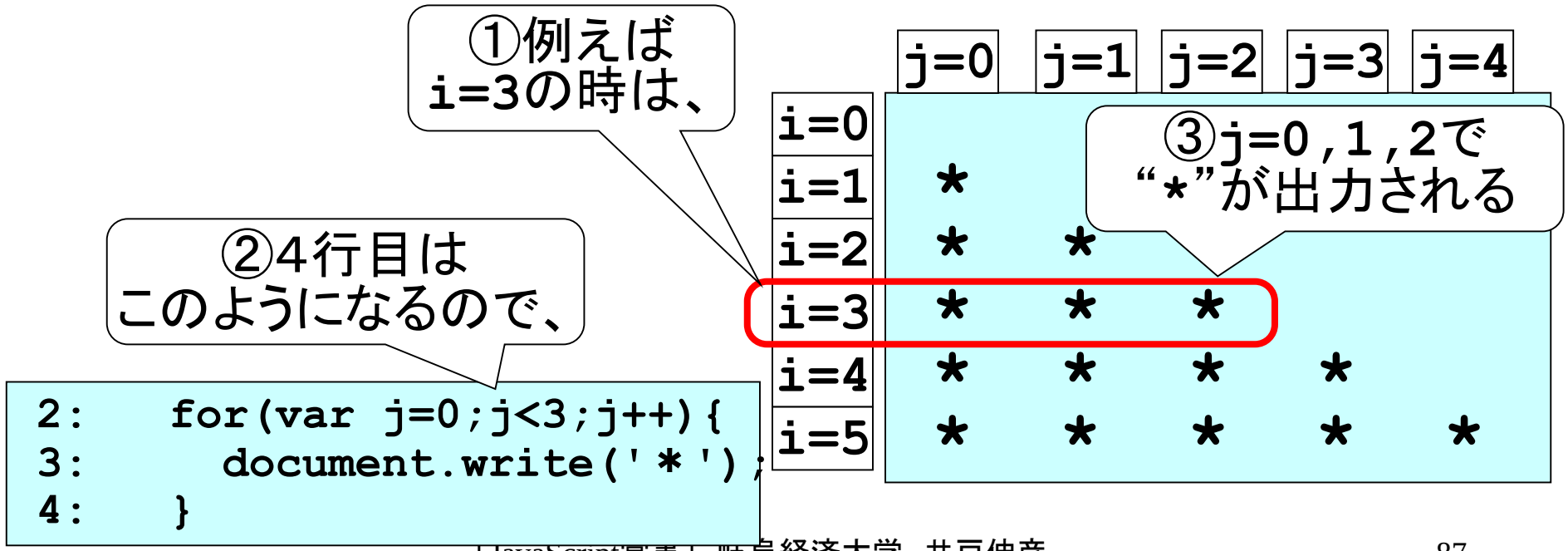
```
1: for(var i=0;i<6;i++){
2:    for(var j=0;j<i;j++){
3:      document.write('＊');
4:    }
5:    for(var j=0;j<5-i;j++){
6:      document.write('＃');
7:    }
8:    document.write('<br />');
9: }
```

- 各行の表示を揃えるために、“＊”と“＃”とは、日本語入力しています。

（9．7．1）“*”の部分

■アスタリスク“*”の部分だけをまず考えます。

```
1: for(var i=0;i<6;i++){
2:    for(var j=0;j<i;j++){
3:       document.write('*');
4:    }
8:    document.write('<br />');
9: }
```



（9．7．2）“#”の部分

■i行目で出力される“#”の数は、“5-i”と表すことが出来ます。

```
#####
*####
**###
***##
****#
*****
```

行	#の数	iで表すと…
0行目(i=0)	5	= 5-i = 5-0
1行目(i=1)	4	= 5-i = 5-1
2行目(i=2)	3	= 5-i = 5-2
3行目(i=3)	2	= 5-i = 5-3
4行目(i=4)	1	= 5-i = 5-4
5行目(i=5)	0	= 5-i = 5-5

■よって、“#”を出力は、“0”から“5-i”まで繰り返せばよい訳です。

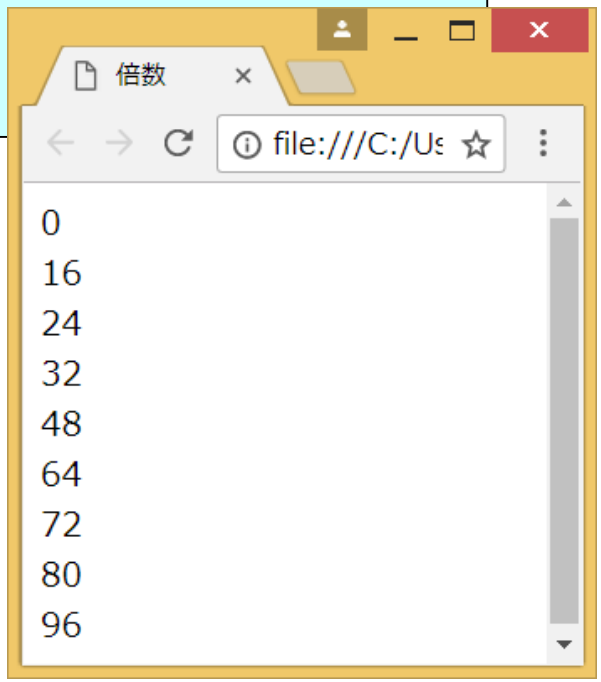
```
1: for(var i=0;i<6;i++){
5:    for(var j=0;j<5-i;j++){
6:       document.write('#');
7:    }
8:    document.write('<br />');
9: }
```

（９．８）例題：倍数

■0から100までの数の中で、16の倍数であるか、24の倍数である数をすべて打ち出してください。

■解答例：

```
for(var i=0;i<=100;i++){
  if((i%16==0)||(i%24==0)){
    document.write(i+'<br />');
  }
}
```



（９．９）課題

■（９．９．１）課題

- 次のように初期化された配列のそれぞれの値の、①平均、②最大、③最小を求めるプログラムを作成してください。

```
var dt = [172.3,168.5,177.8,181.5,174.8];
```

■（９．９．２）課題

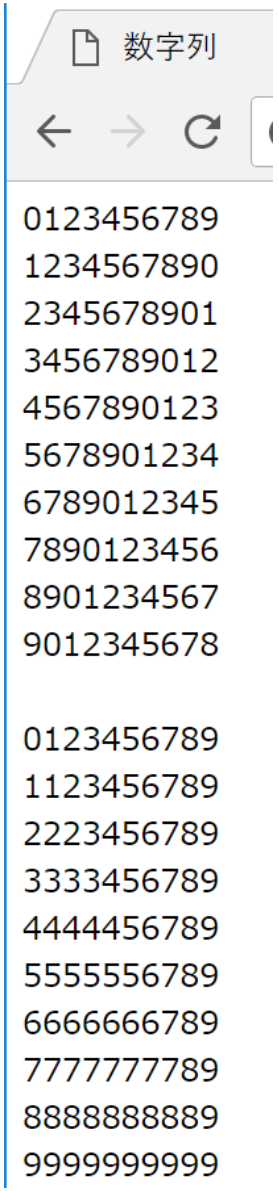
- 次のような出力を行うプログラムを作成してください（for文を使ってください）。

```
    *
   ***
  *****
*******
  *****
   ***
    *
```

（9．10）課題

■課題

- 右のような出力を得るプログラムを作成してください（二重ループを用います）。

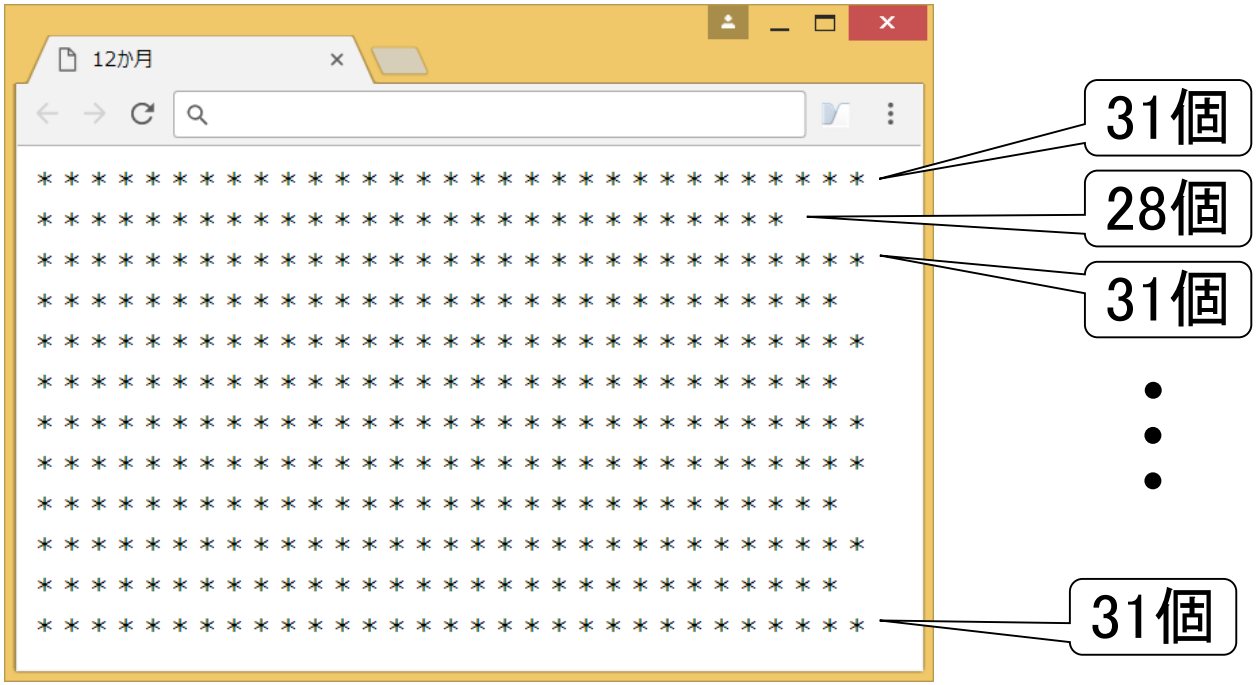


（9．11）課題

■課題

- 下のような配列“months”を宣言し、その下のような出力を得るプログラムを作成してください。

```
var months=[31,28,31,30,31,30,31,31,30,31,30,31];
```

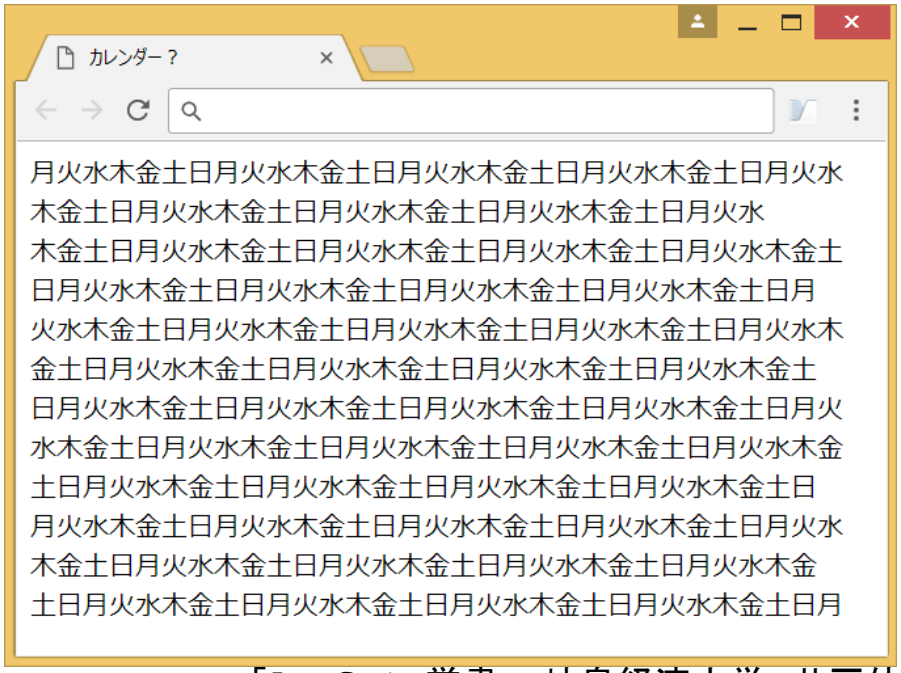


（9．12）課題

■課題

- 「（9．11）課題」のプログラムを改造して、1月1日が月曜日の場合の、各月の曜日の羅列を出力するプログラムを作成してください。ただし、次のような配列を宣言するものとします。

```
var days=['月','火','水','木','金','土','日'];
```



（10．1）プログラムの見栄え

■ここで説明することは、作成したプログラムの実行に直接関わることではありません。単にソースコードの“見た目”に関わるだけです。しかしながら、どうでも良いということではなく、重要なことです。

■次の2つのプログラムの断片を見て、どう感じますか？　どちらのプログラムの実行結果も同じですが、＜B＞のようなプログラムを書く人と共同でプログラムを開発する気は起こらないと思います。

＜A＞

```
var x=0
var y=3;
var z=5;
x = z-y;
document.write(“x=”+x);
```

＜B＞

```
    var x=0
  var y=3;
                    var z=5;
  x = z-y;
   document.write(“x=”+x);
```

（10．2）段下げ

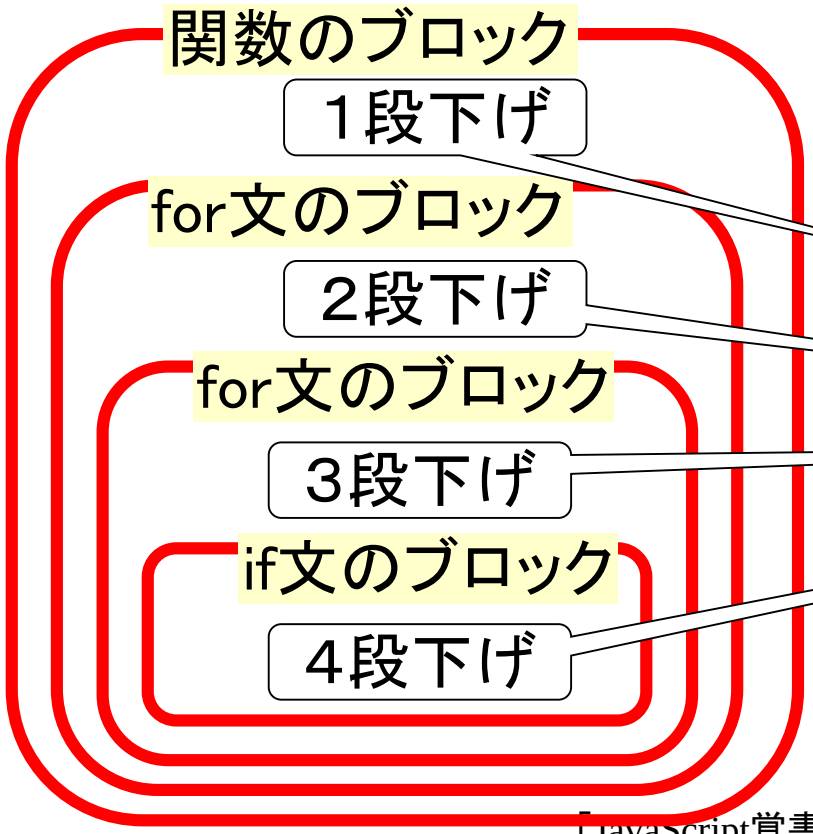
■クラス、メソッド、if文、while文、for文では、その中身の実行文について通常4文字段下げを行うことが、見栄えの上でのルールとなっています。これを守らないプログラムを書く人は、ある意味前ページの＜B＞より酷いと言えます。“{”と“}”の位置にも注意してください。

関数
for文
for文
if文

```
funtion calc_sum{
    for(var i=0;i<10;i++){
        int sum = 0;
        for(var j=0;j<10;j++){
            if(((i+j) % 2) ==0){
                sum += i;
            }
        }
        document.write("sum="+sum);
    }
}
```

（10．3）ブロック

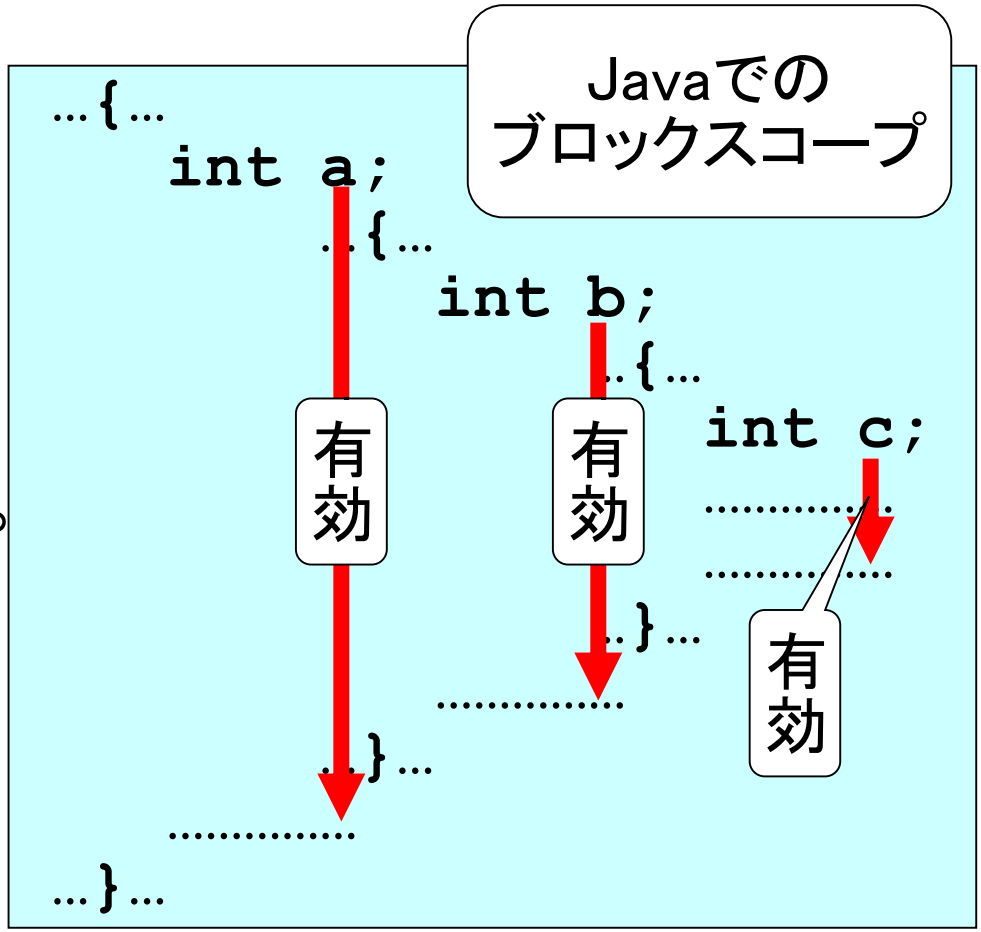
- “{”で始まり、これに対応する“}”で終わる範囲のことを、ブロックを言います。
- どれだけ段下げを行うかは、いくつのブロックに囲まれているかにより決まる訳です。




```
funtion calc_sum{
    for(var i=0;i<10;i++){
        int sum = 0;
        for(var j=0;j<10;j+
            if(((i+j) % 2)
                sum += i;
            }
        }
        document.write("sum
    }
}
```

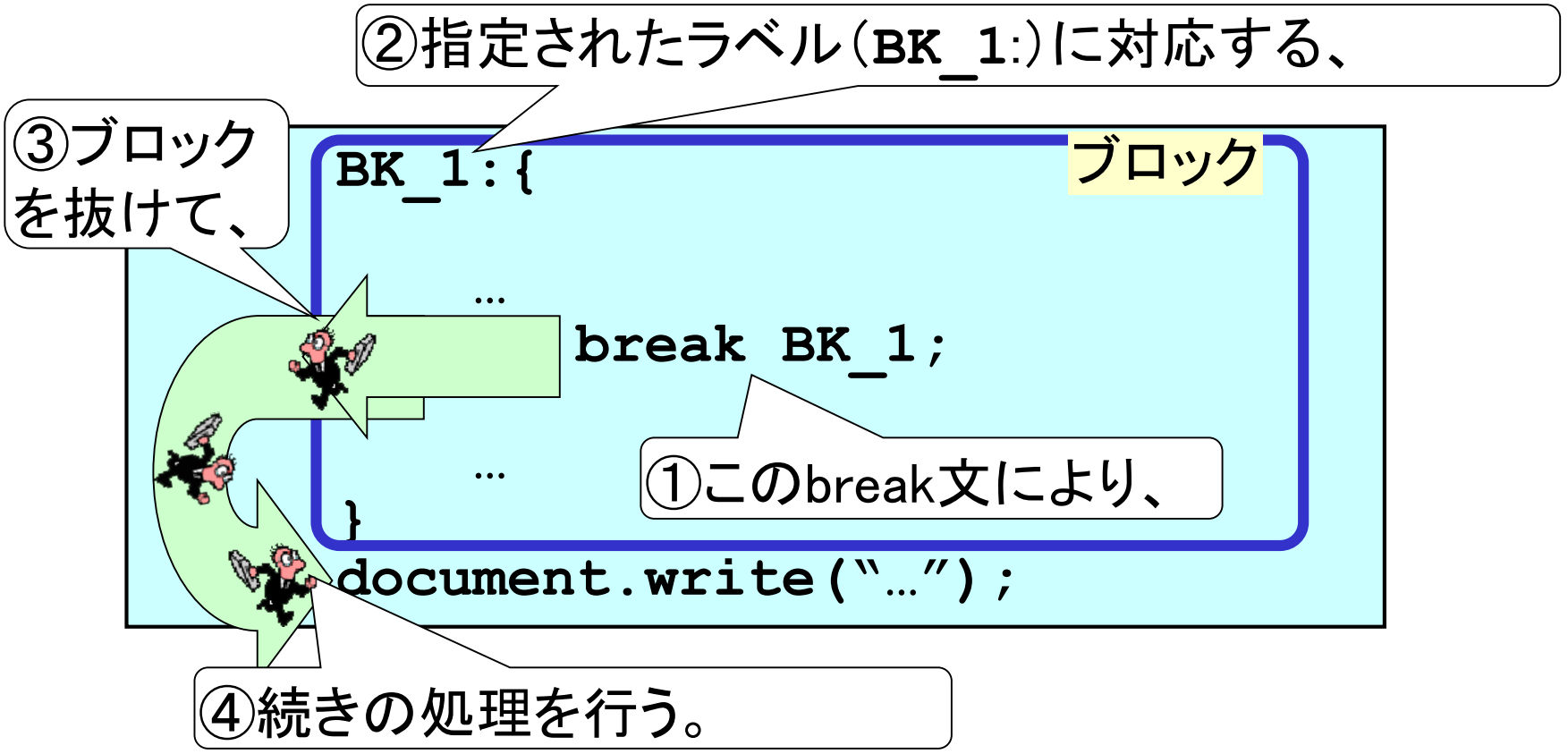
（10．4）スコープ（変数の有効範囲）

- スコープとは、宣言した変数が使えるか範囲のことを指すと理解しておいてください。
- Java等と異なり、JavaScriptではブロックはスコープになりません。
- 変数を定義したブロックの範囲外でもその変数にアクセス出来てしまします。この点に注意が必要な場合があります。



（10．5）ブロックに対するbreak文

■（9）章にて説明したbreak文は、for文やwhile文などの繰り返しに対して使うだけではなく、①if文のブロックや、②for文やwhile文無しのブロックだけにも使うことが出来ます。

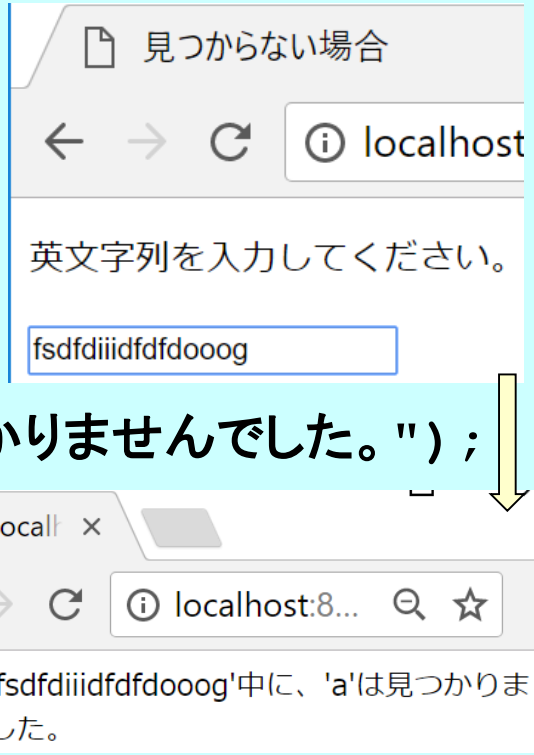


（10．8．1）“見つからない場合の処理”

■ブロックをbreak文で抜ける必要があるのかというと、必要な場合は時々あります。

```
<script type="text/javascript">
function notFound(){
    var str = document.form1.string.value;
    BK_1:{
        for(var i=0;i<str.length;i++){
            if(str.charAt(i)=='a'){
                break BK_1;
            }
        }
        document.write("文字列'"+str+
                       "'中に、'a'は見つかりませんでした。");
    }
}
</script>
<p>英文字列を入力してください。</p>
<form name="form1">
<input type="text" name="string" onchange="notFound();"/>
</form>
```

charAt等は、スライド(13)参照。

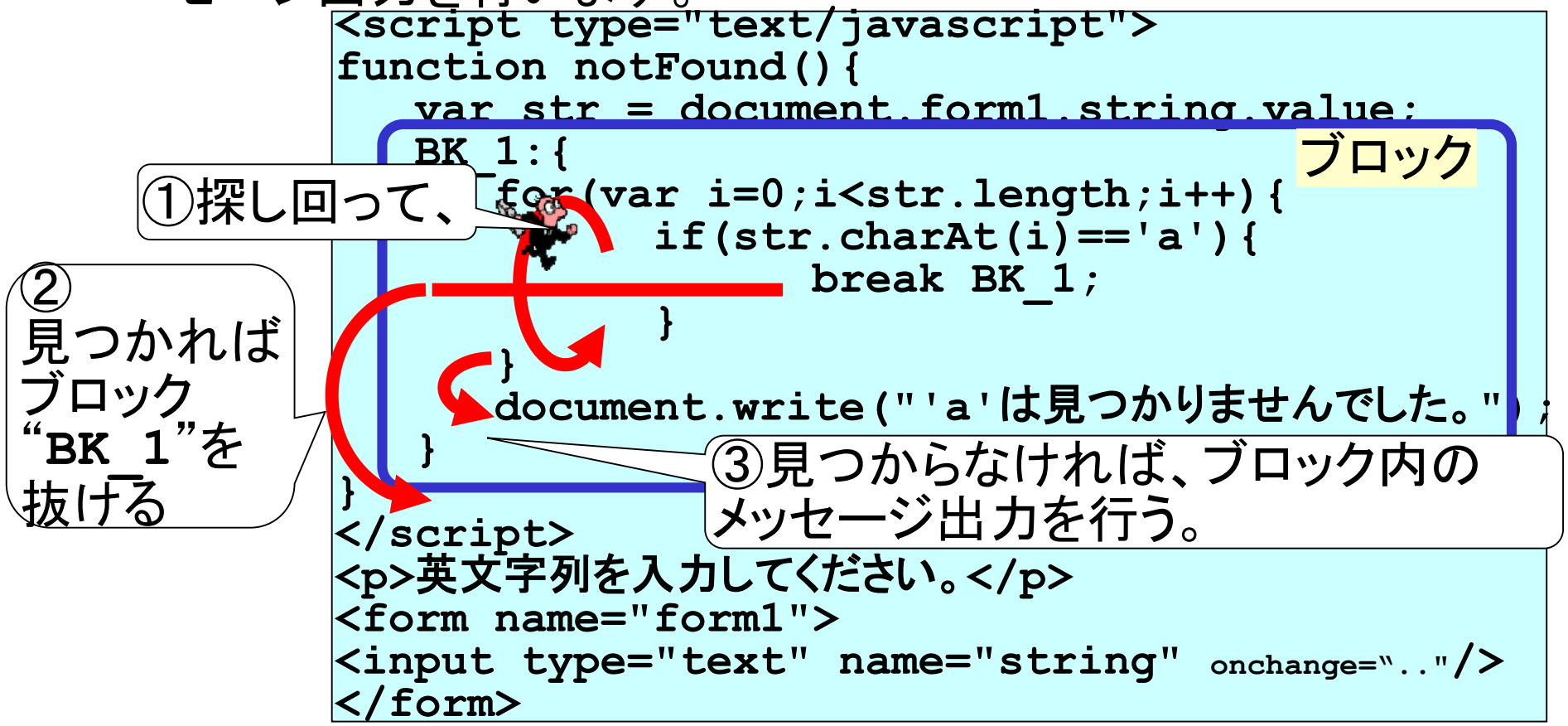


（10．8．2）処理の概要

■**for**文により文字‘a’を探し回って、

【見つかった場合】 **break**文によりブロックを抜けることで、ブロック内のメッセージ出力は行われません。

- 【見つからない場合】 **for**文の次にある、ブロック内のメッセージ出力を行います。




```
<script type="text/javascript">
function notFound(){
    var str = document.form1.string.value;
    BK_1:{
      for(var i=0;i<str.length;i++){
        if(str.charAt(i)=='a'){
          break BK_1;
        }
      }
      document.write("'a'は見つかりませんでした。");
    }
}
</script>
<p>英文字列を入力してください。</p>
<form name="form1">
<input type="text" name="string" onchange="..."/>
</form>
```

（10．8．3）変数の導入

■もちろん、「’a’を含むか否かを表す」変数を導入すれば、スライド(10.8.1)の処理は下のようにも書けます。

- これでも問題はありませんが、(10.8.1)の流儀で書きたくなる場面はよくあります。
- 「導入する必要のない変数は導入しない」という考えにより、筆者自身は(10.8.1)の処理を好みます。

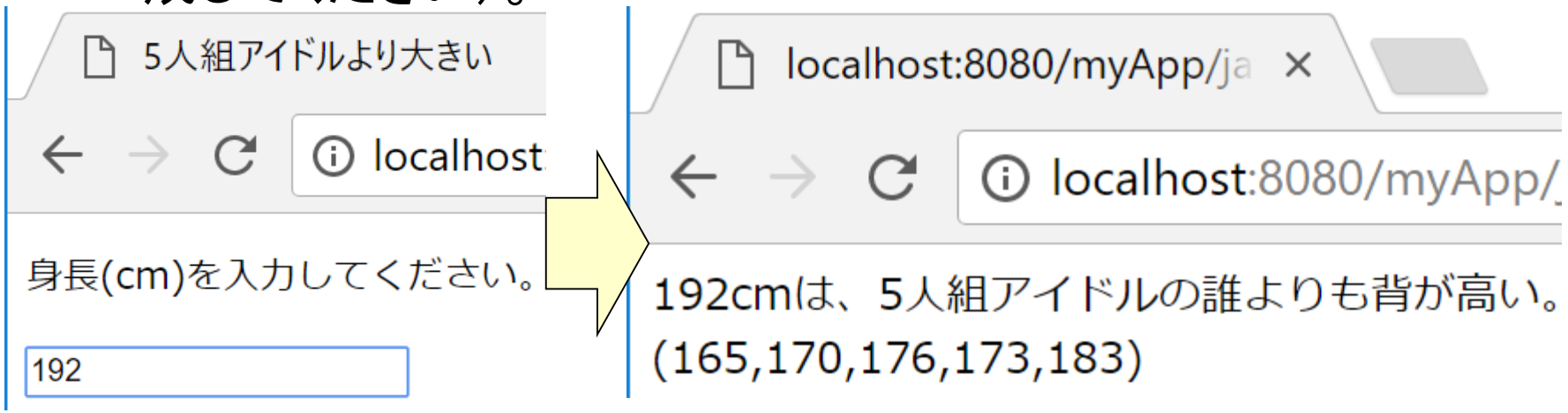
```
<script type="text/javascript">
function notFound(){
   var str = document.form1.string.value;
   var no_a=true;
   for(var i=0;i<str.length;i++){
      if(str.charAt(i)=='a'){
            no_a=false;
            break;
      }
   }
   if(no_a){
      document.write("文字列'"+str+"'中に、'a'は(略)でした。");
   }
}
</script>
```

（10．9）課題

- ここまでに作成したすべてのプログラムについて、段下げが正しくなされているか否かをチェックしてください。

■課題

- 下図のように初期化された配列を用意します。数値を入力して、配列内のすべての値が入力した値よりも大きい場合に、“配列の各値はすべて入力した値より大きい。”と出力するプログラムを作成してください（スライド(10.8.1)流のやり方で作成してください）。

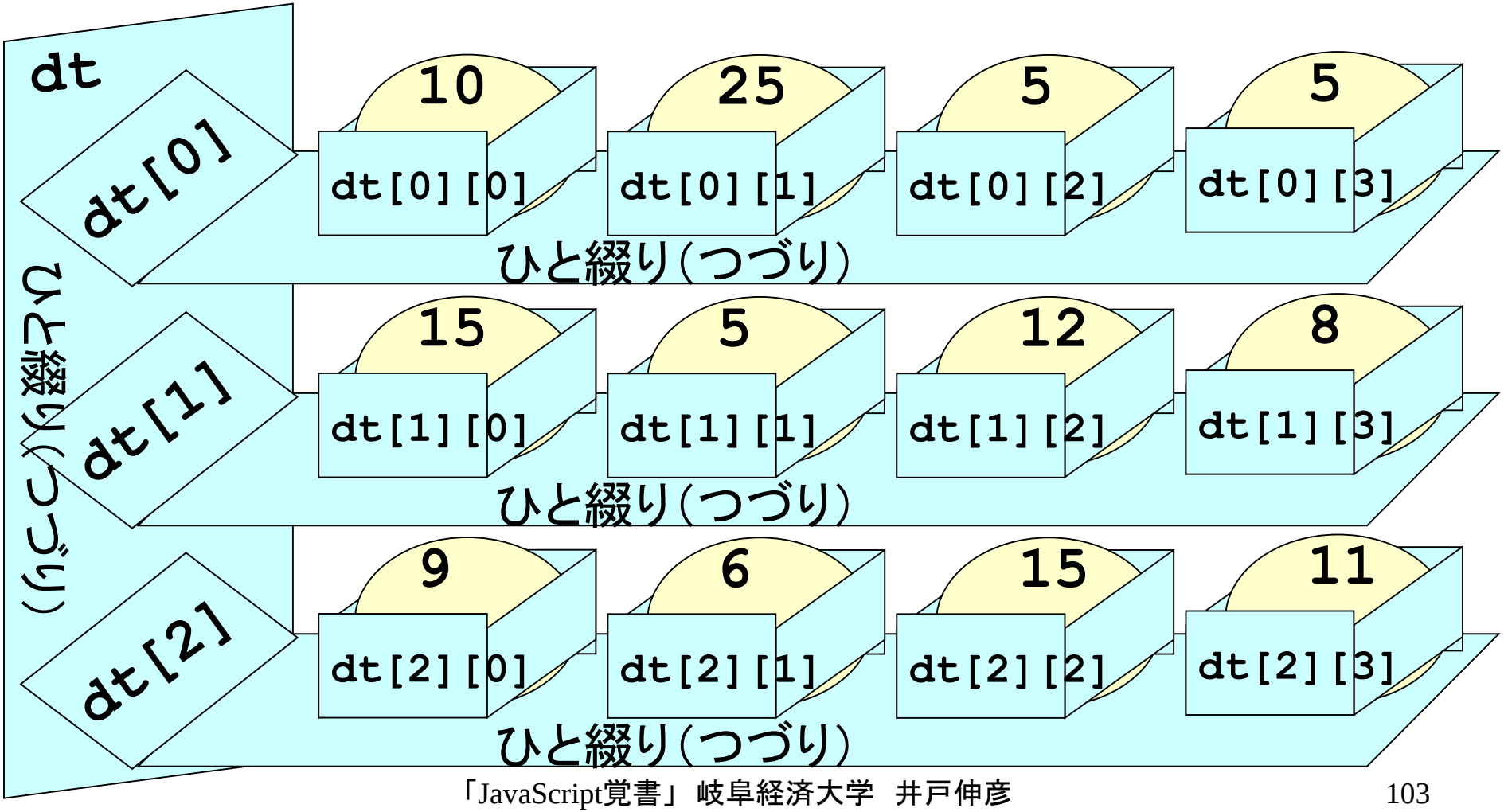



```
var hs=[165,170,176,173,183];
```

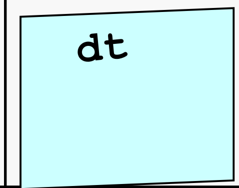
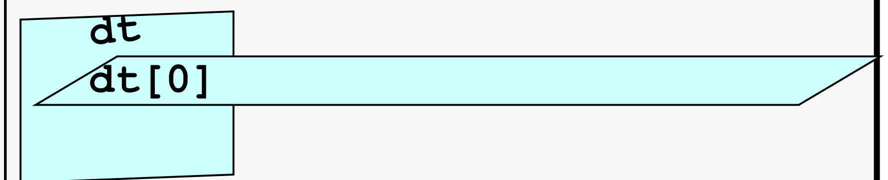
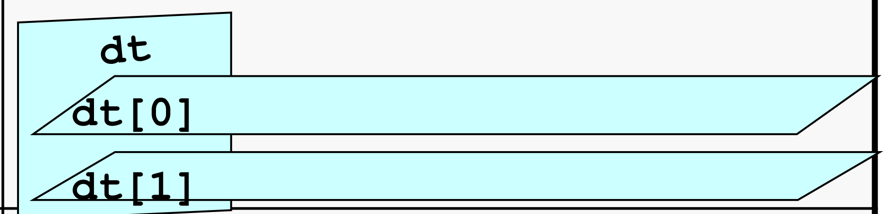
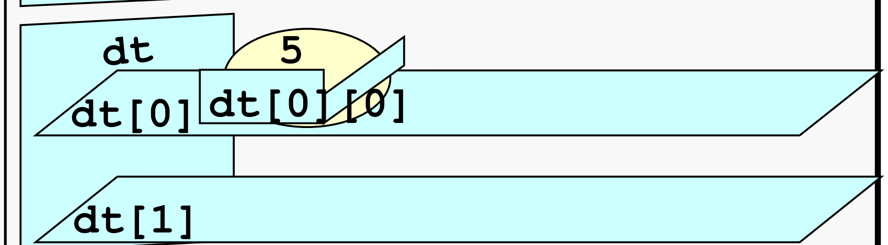
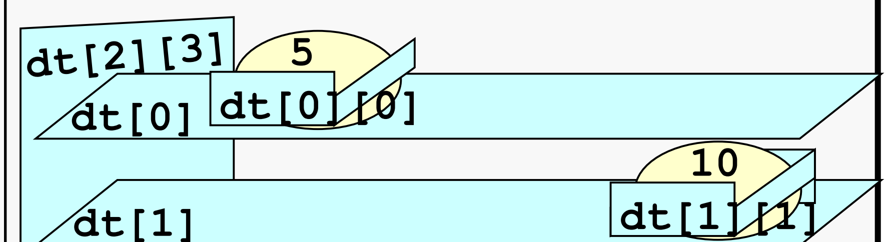
（１１）多次元配列

■配列は、多次元にすることも出来ます。

■３×４の2次元配列は、次のようなイメージとなります。

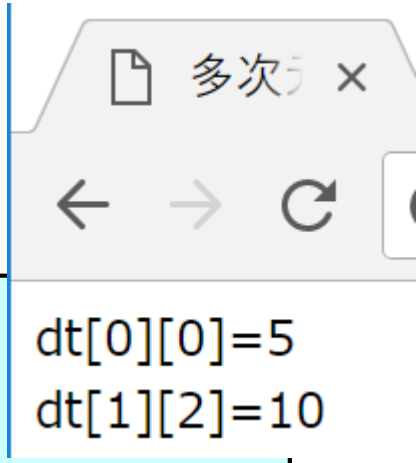


（11．1）多次元配列の宣言

操作	記述例	操作跡の変数の内容
①変数dtに 配列を設定する	**`var dt=new Array();`**	
②配列dtの0番目の要素に配列を設定する	**`dt[0]=new Array();`**	
③配列dtの1番目の要素に配列を設定する	**`dt[1]=new Array();`**	
④配列dtの0番目の要素の配列の、0番目の要素に“5”を設定する	**`dt[0][0]=5;`**	
⑤配列dtの1番目の要素の配列の、2番目の要素に“10”を設定する	**`dt[1][2]=10;`**	

（11．2）値の出力

■前スライドのプログラムの、設定した値を出力してみます。



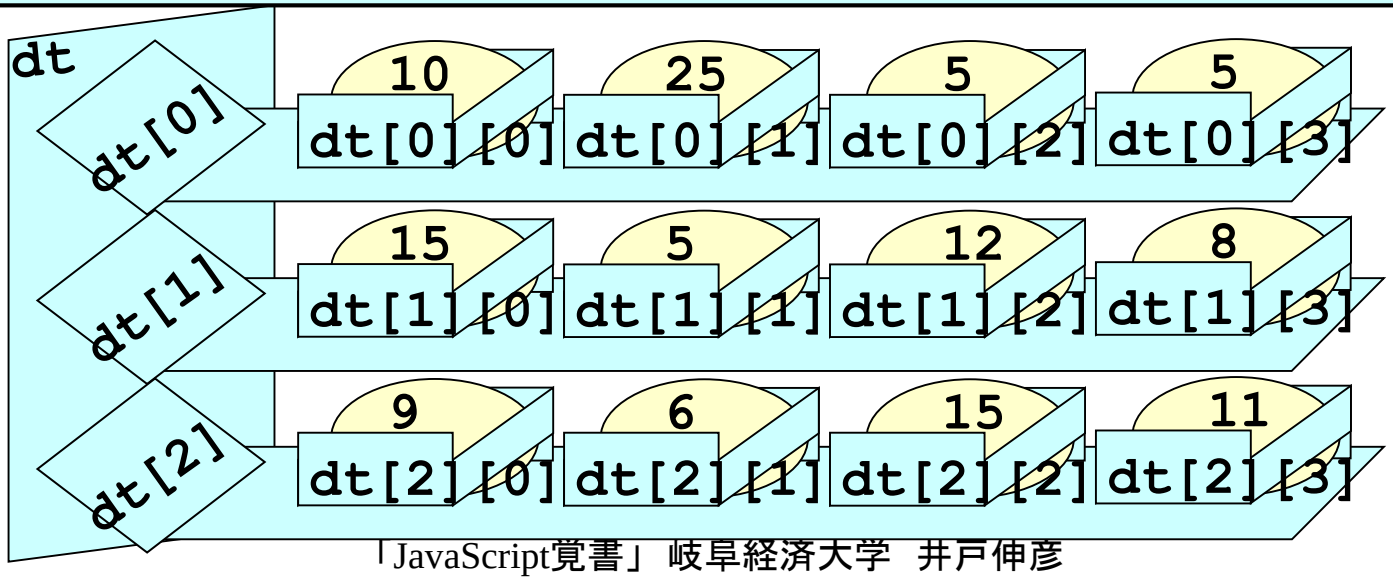

```
<script type="text/javascript">
var dt=new Array();
dt[0]=new Array();
dt[1]=new Array();
dt[0][0]=5;
dt[1][2]=10
document.write(“dt[0][0]=”+dt[0][0]+“<br />");
document.write("dt[1][2]="+dt[1][2]+"<br />");
</script>
```

（11．3）初期化

■多次元配列も、次のように初期化が可能です。

```
<script type="text/javascript">
var dt=[
       [10,25, 5, 5],
       [15, 5,12, 8],
       [ 9, 6,15,11],
    ];
document.write("dt[0][0]="+dt[0][0]+"<br />");
document.write("dt[1][2]="+dt[1][2]+"<br />");
</script>
```

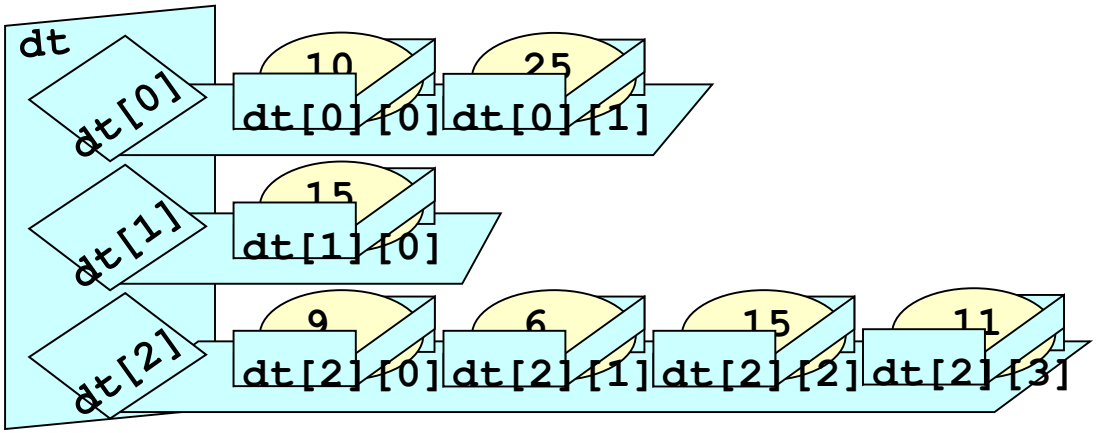
このカンマ（最終カンマ）はあってもなくてもOK。



（11．4）非矩形の配列

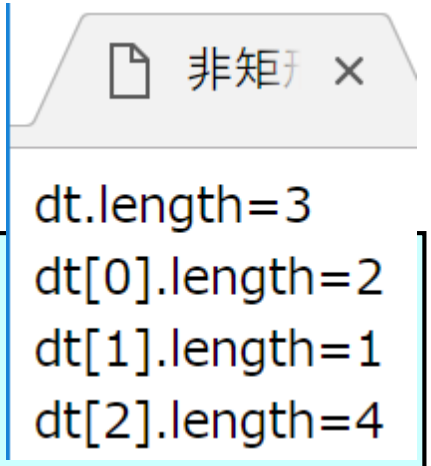
■次のように、2次元目の長さが異なる配列とすることもOKです。

```
var dt=[
 [10,25],
 [15],
 [ 9, 6,15,11],
];
```



■配列の長さを打ち出すと、次のようになります。

```
<script type="text/javascript">
var dt=[ [10,25],
         [15],
         [ 9, 6,15,11],];
document.write("dt.length="+dt.length+"<br />");
document.write("dt[0].length="+dt[0].length+"<br />");
document.write("dt[1].length="+dt[1].length+"<br />");
document.write("dt[2].length="+dt[2].length+"<br />");
</script>
```



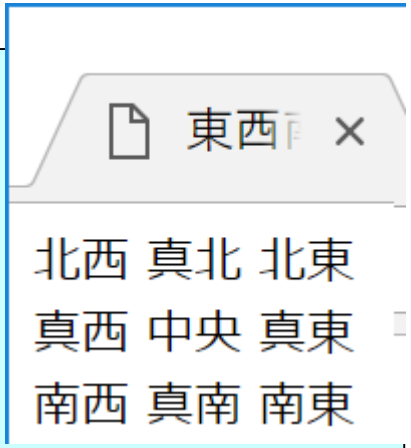
（11．5）例題：二次元配列の打出し

■右の配列を順次打出すプログラムを作成してください。

```
var directions=[
 ['北西','真北','北東',],
 ['真西','中央','真東',],
 ['南西','真南',’南東’,],
];
```

■解答例：

```
<script type="text/javascript">
var directions=[
 ['北西','真北','北東',],
 ['真西','中央','真東',],
 ['南西','真南','南東'],];
for(var i=0;i<directions.length;i++){
   for(var j=0;j<directions[i].length;j++){
      document.write(directions[i][j]+' ');
   }
   document.write('<br />');
}
</script>
```



（11．6）例題：花文字

- “A”、“B”、“C” のいずれかの文字を入力して、入力した文字の花文字を出力するプログラムを作成してください。
- 花文字とは、右図のように文字を連ねて描いた文字のことです。“B”、“C”についても、適当に作ってください。
- 多次元配列を使ってください。

（11．7．1）解答例1（1／2）

```
<script type="text/javascript">
function draw_character(){
   var hana=[
      [      "___A",
             "__A_A",
             "_AAAAA",
             "A_____A",
      ],
      [      "BBBBB",
             "B____B",
             "BBBBB",
             "B____B",
             "BBBBB",
      ],
      [      "_CCCC",
             "C____C",
             "C",
             "C____C",
             "_CCCC",
      ],
   ];
```

空白の代わりに
アンダーバー（“_”）を
使っています。

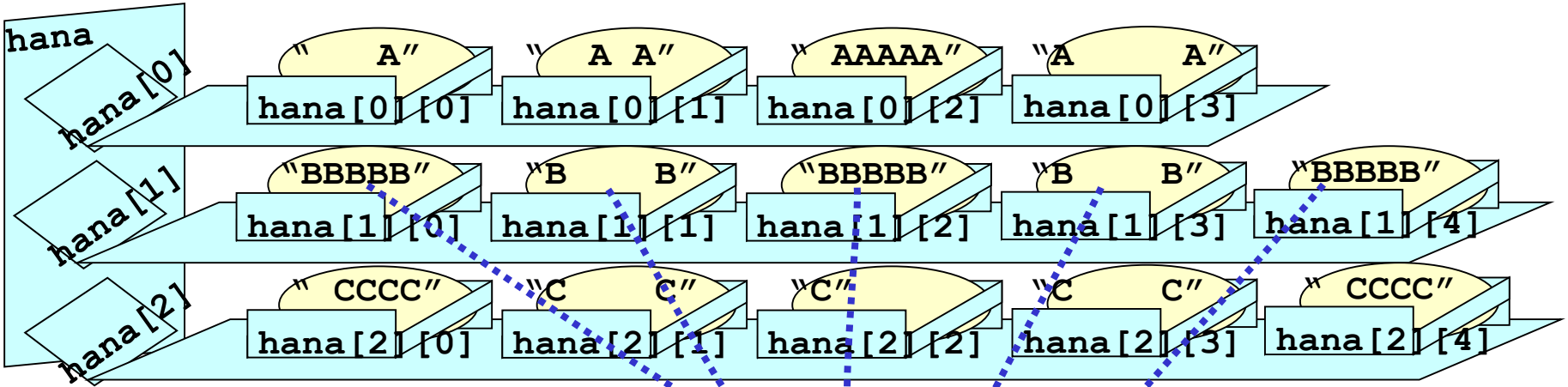
（11．7．2）解答例1（2／2）

```
    var c_input = document.form1.character.value;
    if(c_input=='A'){
        for(var i=0;i<hana[0].length;i++){
            document.write(hana[0][i]+'<br /> ');
        }
    }else if(c_input=='B'){
        for(var i=0;i<hana[1].length;i++){
            document.write(hana[1][i]+'<br />');
        }
    }else if(c_input=='C'){
        for(var i=0;i<hana[2].length;i++){
            document.write(hana[2][i]+'<br />');
        }
    }else{
        document.write('A,B,C以外は無効。');
    }
}
</script>
<p>A,BまたはCを入力してください。</p>
<form name="form1">
<input type="text" name="character"
        onchange="draw_character();"/><br />
</form>
```

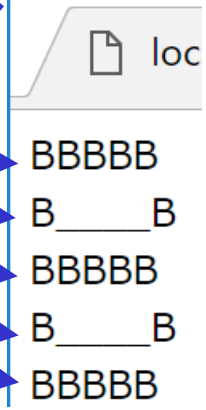
（11．7．3）解答例1の概要

■Hana[][]は、下図のような配列です。

■入力された文字によって、下の図でのいずれかの1行を打出せば、花文字が出力される訳です。




```
    :
}else if(c_input=='B'){
  for(var i=0;i<hana[1].length;i++){
    document.write(hana[1][i]+'<<br />);
  }
}.....
```



（11．7．4）解答例1の問題点

■解答例1のでは、次のような同じような記述が繰り返されています。

```
for(var i=0;i<hana[0].length;i++){
   document.write(hana[0][i]+'<br /> ');
}
```

```
for(var i=0;i<hana[1].length;i++){
   document.write(hana[1][i]+'<br />');
}
```

```
for(var i=0;i<hana[2].length;i++){
   document.write(hana[2][i]+'<br />');
}
```

■同じような記述があること自体無駄でもあり、また、この部分を変更する際には3箇所変更が必要になるなどの問題があります。

■上記の3つの部分の違いは 0 の部分だけなので、これを変数とすれば、1つにまとめることができます。

（11．8．1）解答例2（1／1）

```
<script type="text/javascript">
function draw_character(){
   // 文字の定義は解答例1と同じです。
   var c_input = document.form1.character.value;
   var i_char=0;
   if(c_input=='A'){
      i_char=0;
   }else if(c_input=='B'){
      i_char=1;
   }else if(c_input=='C'){
      i_char=2;
   }else{
      document.write('A,B,C以外は無効。');
      return;
   }
   for(var i=0;i<hana[i_char].length;i++){
      document.write(hana[i_char][i]+'<br /> ');
   }
}
</script>
<!-- フォームも解答例1と同じです。 -->
```

（11．8．2）解答例2の概要

■どの文字を打出すかを“ichar”という変数に設定しています。

```
var i_char=0;
if(c_input=='A'){
    i_char=0;
}else if(c_input=='B'){
    i_char=1;
}else if(c_input=='C'){
    i_char=2;
}else{
    document.write('A,B,C以外は無効。');
    return;
}
```

入力が“A”ならば、0行目。

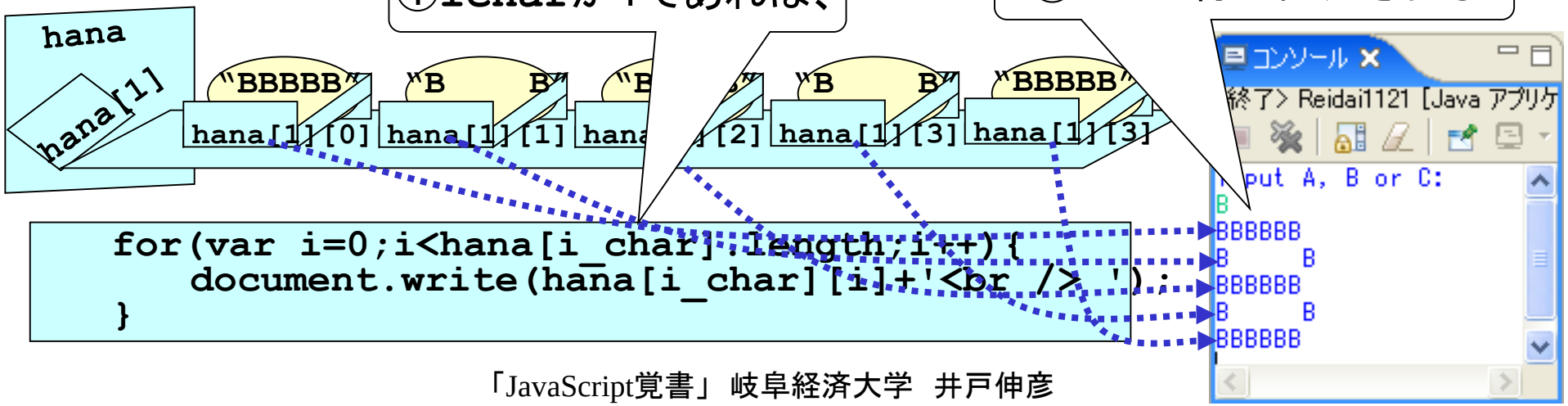
入力が“B”ならば、1行目。

入力が“C”ならば、2行目。

■“ichar”の値により、出力される行が決まります。

①icharが1であれば、

②”B”の行が出力される



（11．8．3）解答例2の問題点

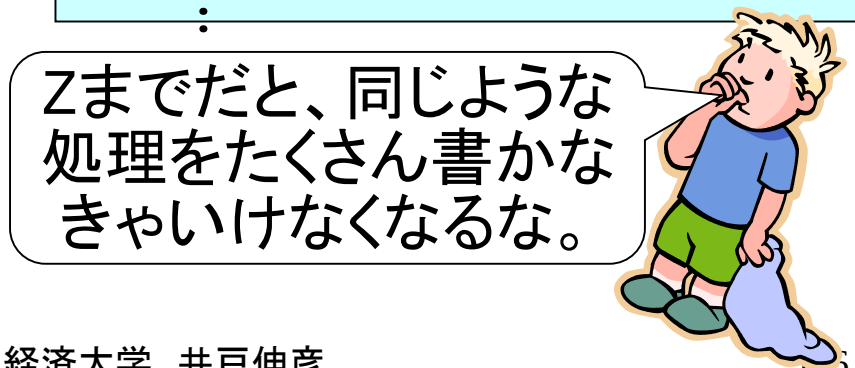
■例題では、AからCまでの３文字としましたが、AからZとした場合には、icharの値を決める（＝どの行を出力するかを決める）ための処理に長い記述が必要になってしまいます。

（AからZになった場合に、行例haraが大きくなることはしかたがありません。）

■これの問題に対応するのが、解答例3になります。

＜出力する行を求める処理＞

```
if(c_input=='A'){
  i_char=0;
}else if(c_input=='B'){
  i_char=1;
}else if(c_input=='C'){
  i_char=2;
}else if(c_input=='D'){
  i_char=2;
}else if(c_input=='E'){
  i_char=2;  }
      :(中略)
}else if(c_input=='Z')){
  ichar=26;
}else{
  :
```

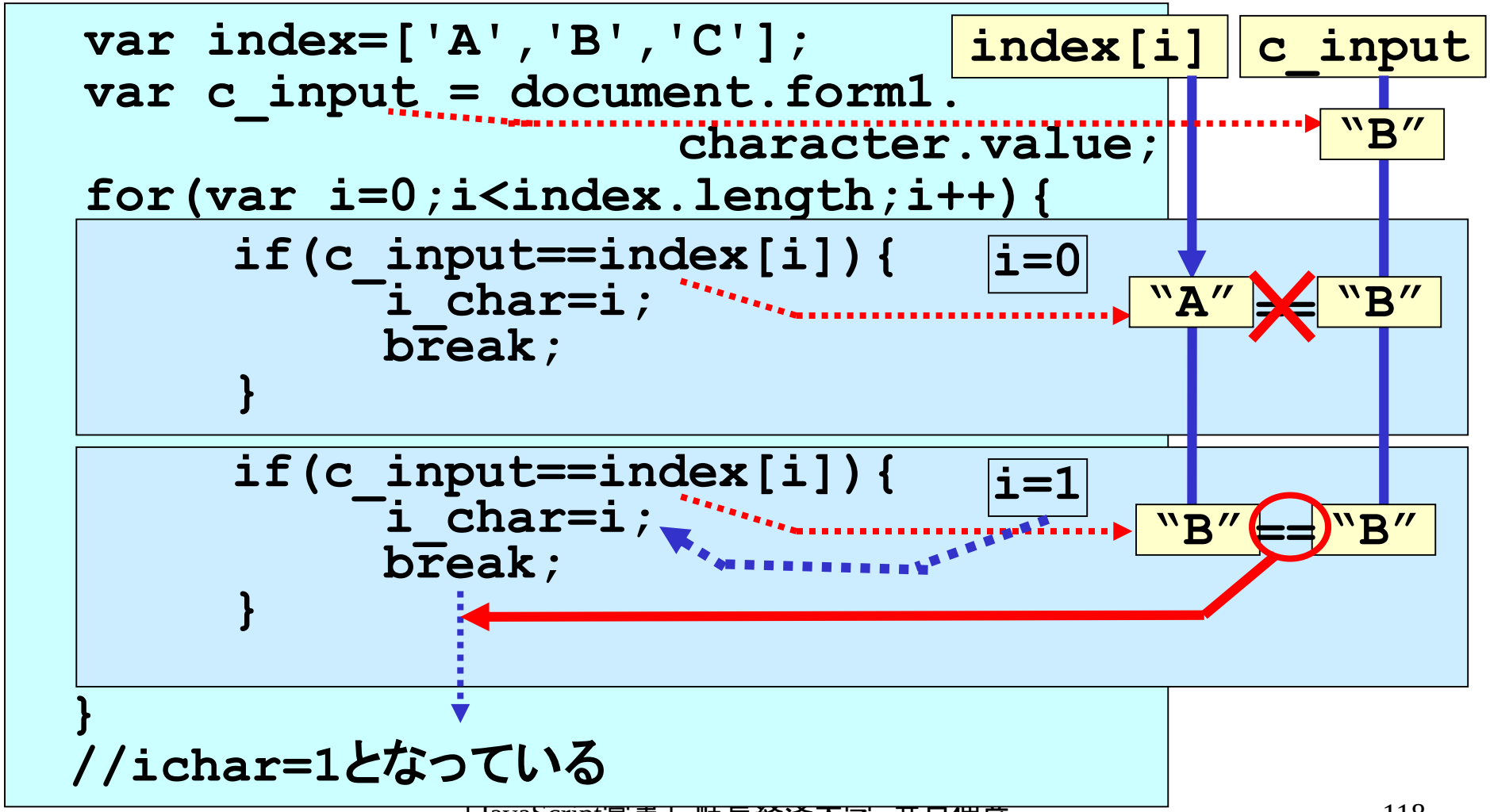


（11．9．1）解答例3（1／1）

```
<script type="text/javascript">
function draw_character(){
    // 文字の定義は解答例1と同じです。
    var index=['A','B','C'];
    var c_input = document.form1.character.value;
    var i_char=-1;
    for(var i=0;i<index.length;i++){
        if(c_input==index[i]){
                i_char=i;
                break;
        }
    }
    if(i_char==-1){
        document.write('A,B,C以外は無効。');
        return;
    }
    for(var i=0;i<hana[i_char].length;i++){
        document.write(hana[i_char][i]+'<br /> ');
    }
}
</script>
<!-- フォームも解答例1と同じです。 -->
```

（11．9．2）解答例3の概要

■解答例3では、次のようにi_charの値（＝hanaの何行目を出力するか）を決めています（入力が“B”の場合）。

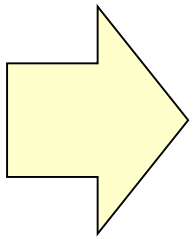


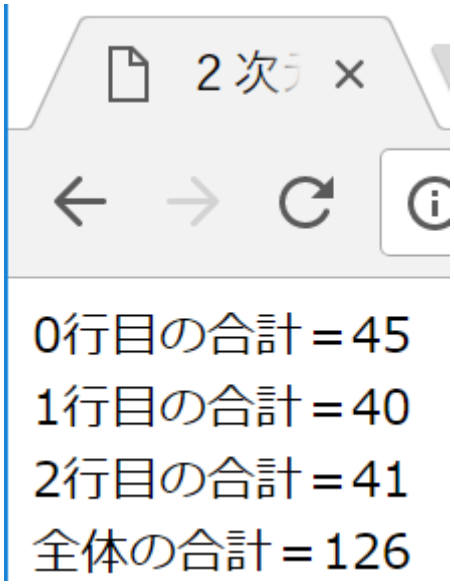
（11．10）課題

■課題

- 左のように初期化された配列の行ごとの合計、全体の合計を求めるプログラムを作成してください。

```
var dt = [
  [10,25, 5, 5],
  [15, 5,12, 8],
  [ 9, 6,15,11],
];
```

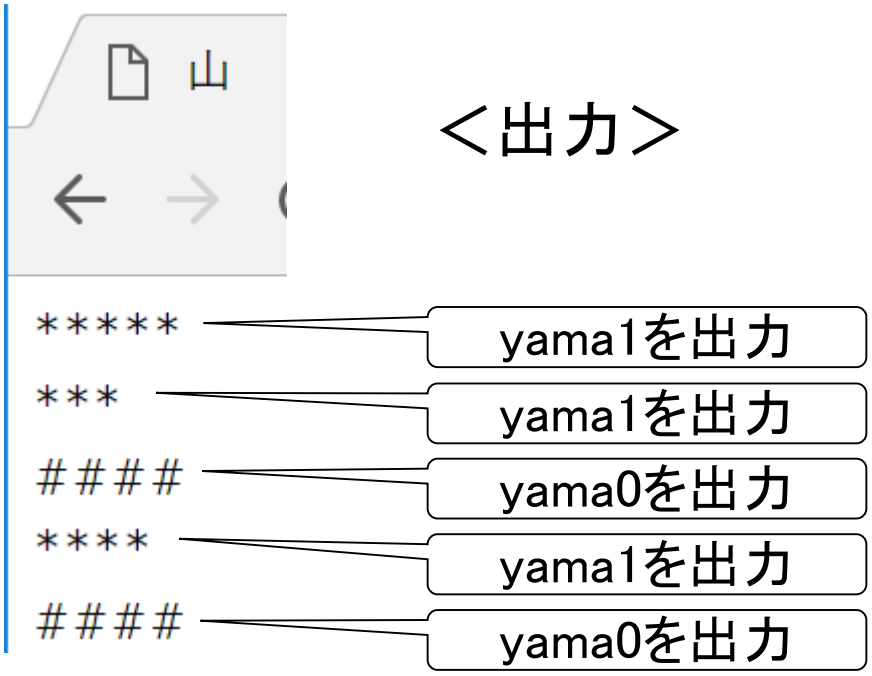




（11．11）課題

■課題

- 右のような配列“yama0”、“yama1”を宣言し、それぞれ対応する2次要素のうち長いほうを出力するプログラムを作成してください。

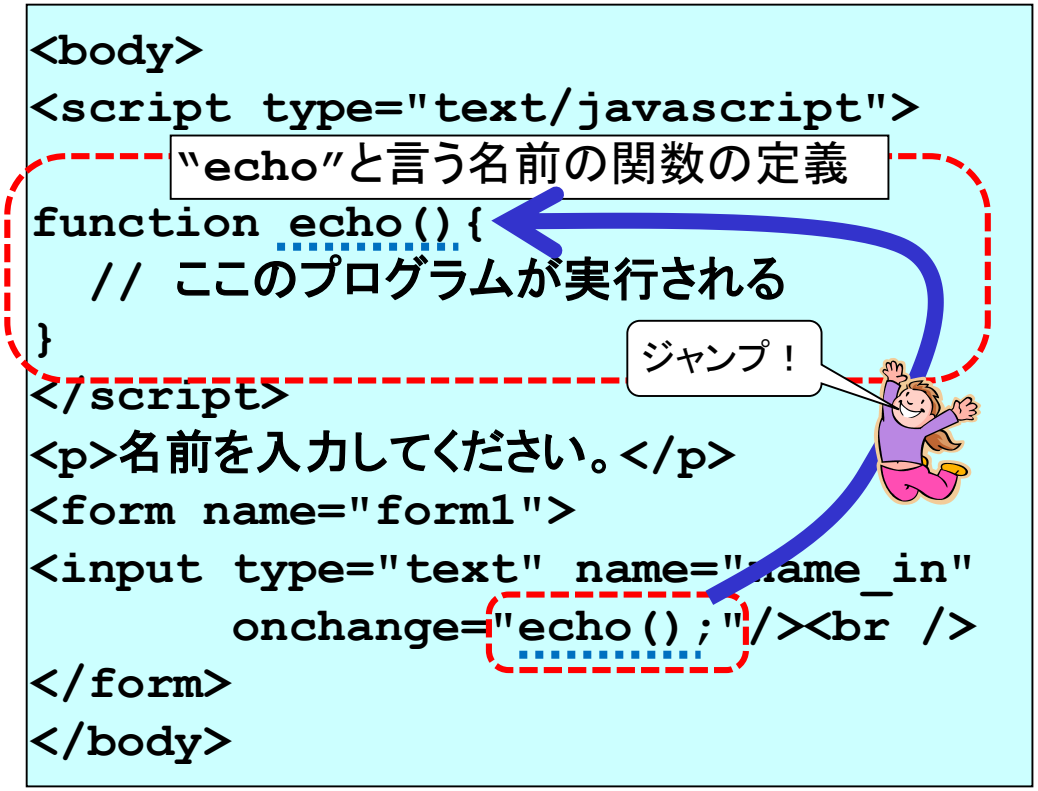



```
var yama0=[
   ['#','#'],
   ['#','#'],
   ['#','#','#','#'],
   ['#','#','#'],
   ['#','#','#','#'],
];
var yama1=[
   ['*','*','*','*','*'],
   ['*','*','*'],
   ['*','*'],
   ['*','*','*','*'],
   ['*','*'],
];
```

長さは2
長さは2
長さは4
長さは3
長さは4
長さは5
長さは3
長さは2
長さは4
長さは2

（12）関数

■スライド(6.4)では関数について、あるところから別のところへプログラムの実行する場所をジャンプ する仕組みであると理解しました。

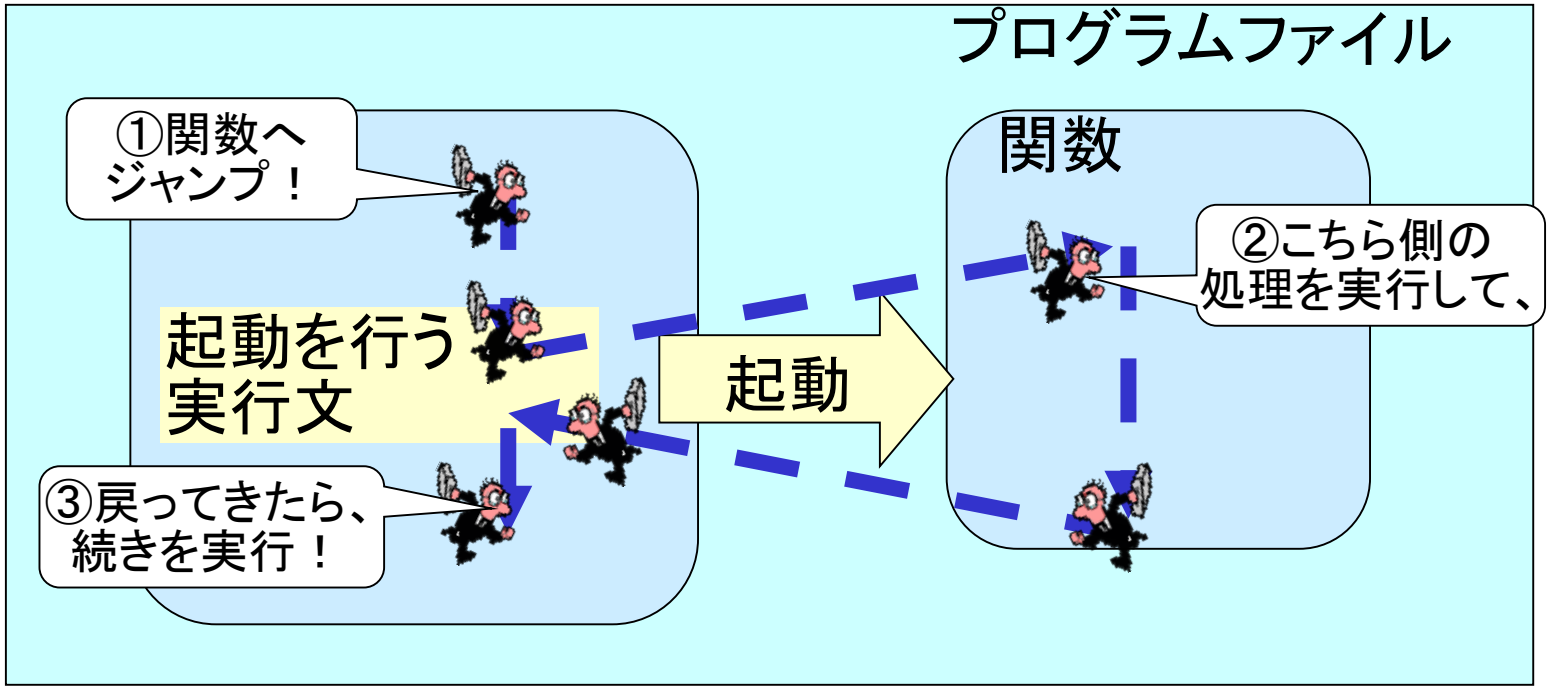



```
<body>
<script type="text/javascript">
function echo(){
  // ここのプログラムが実行される
}
</script>
<p>名前を入力してください。</p>
<form name="form1">
<input type="text" name="name_in"
       onchange="echo();"/><br />
</form>
</body>
```

（12．1）関数を作る

■関数は、実行文として起動することを記述します。
その実行文に処理が移ると、関数が実行されます。

■関数が起動され、その中の実行文が実行し終わると、
起動元の実行文に戻って処理が続けられます。



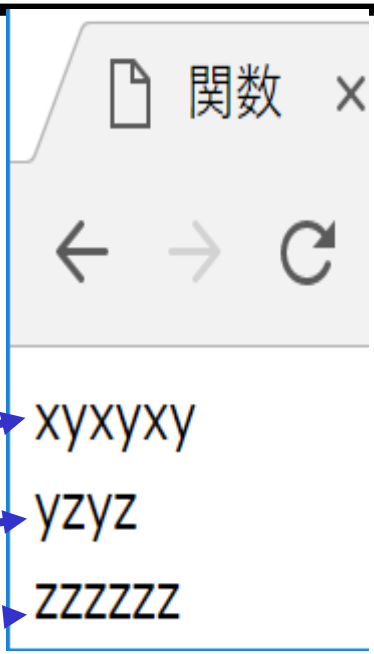
（12．2）関数の例

■次のようなメソッド“repeatString”を作り、実行してみます。

＜プログラム＞

```
<body>
<script type="text/javascript">
repeatString('xy',3);
repeatString('yz',2);
repeatString('zz',3);

function repeatString(str,n){
  for(var i=0;i<n;i++){
   document.write(str);
  }
  document.write('<br />');
}
</script>
</body>
```

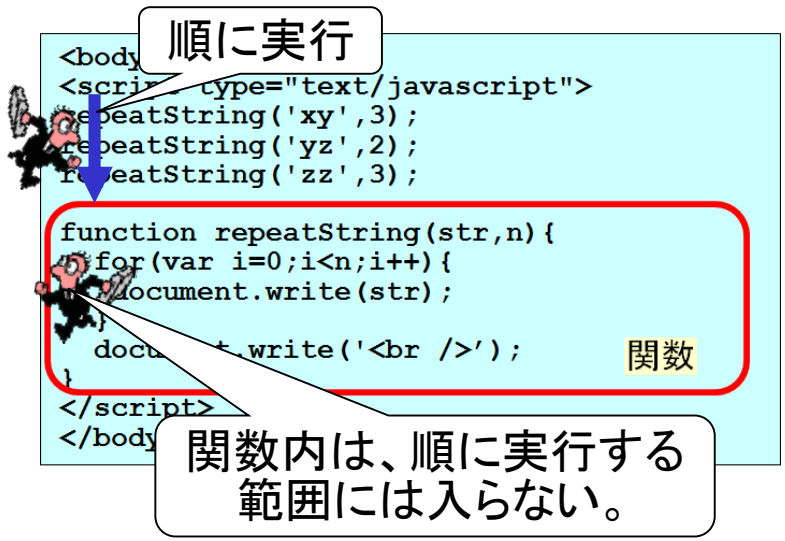
関数



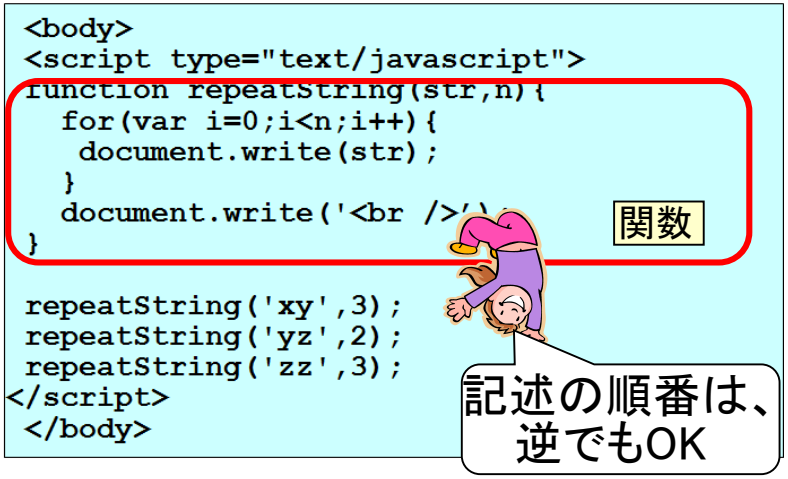
＜出力＞

（12．3）実行される文、記述する場所

■script要素内のJavaScriptプログラムの実行文は、上から順に実行されますが、関数内のプログラムは起動されないと実行されません。

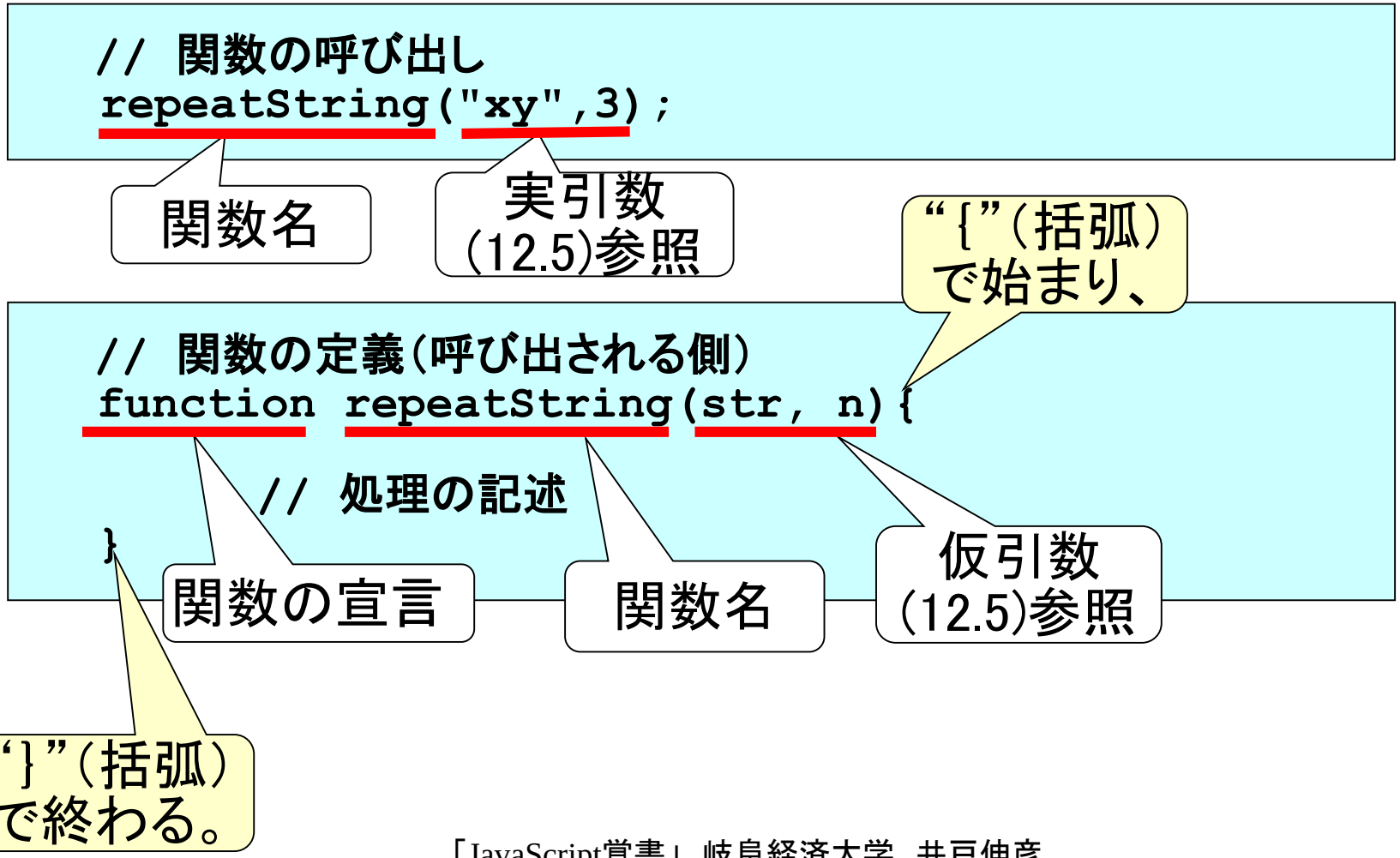


■関数を記述する場所は、起動元よりも前でも構いません（そうする方が多いかも）。



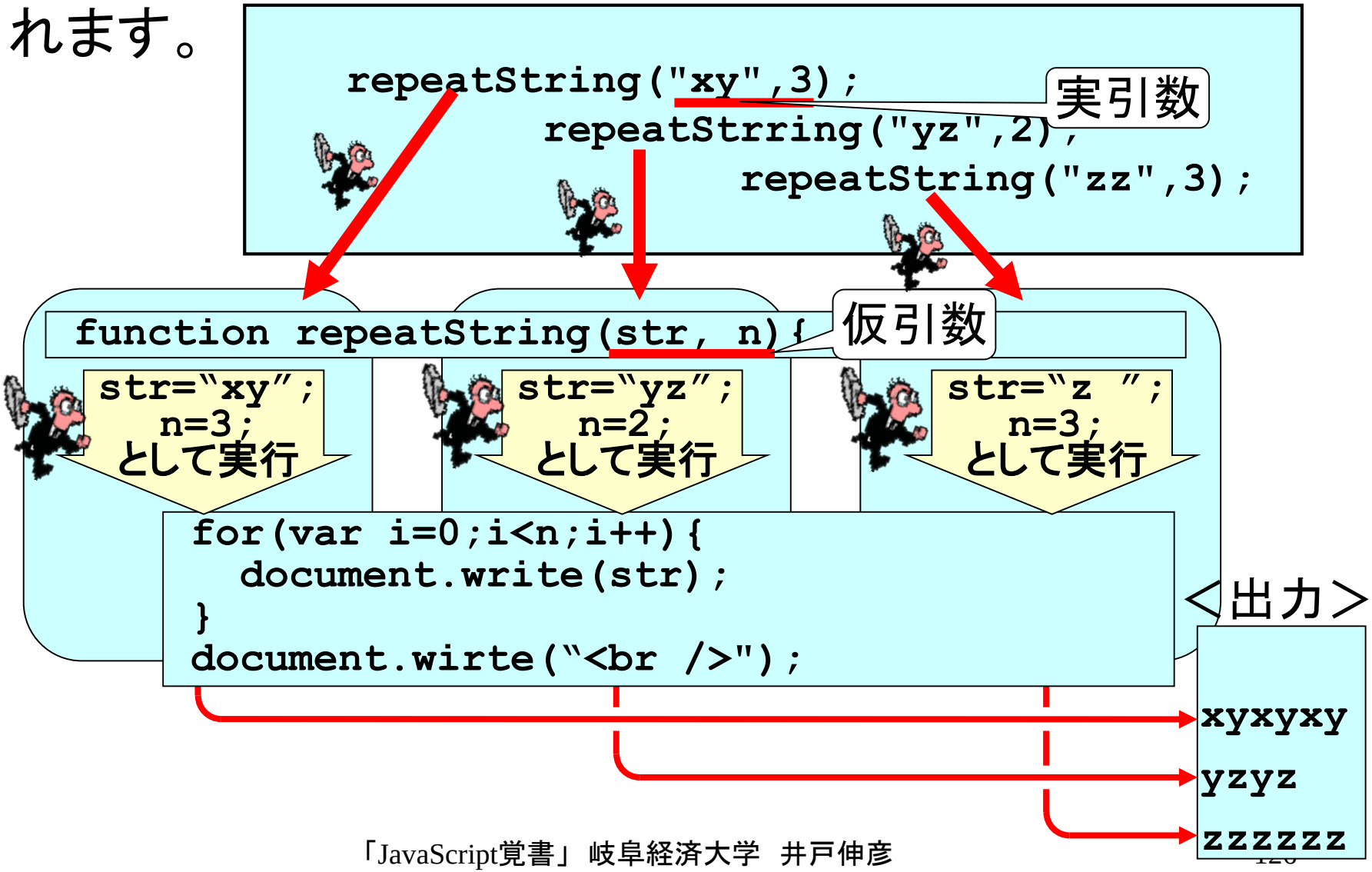
（12．4）関数の形式

■関数の定義では、“{”（括弧）で始まり、“}”（括弧）で終わることに注意してください。



（12．5）引数

■関数での仮引数は、呼び元での実引数の値に設定されます。



（12．6．1）戻り値

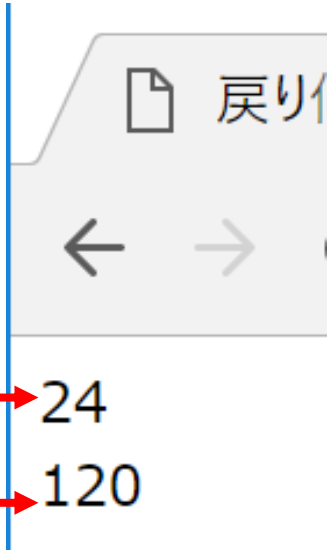
■次のプログラムにて、戻り値について考えます。

戻り値

```
<script type="text/javascript">
var x=calcFactorial(4);
document.write(x+'<br />');
x=calcFactorial(5);
document.write(x+'<br />');

function calcFactorial(a){
        var f=1;
        for(var i=1;i<=a;i++){
                f*=i;
        }
        return f;
}
</script>
```

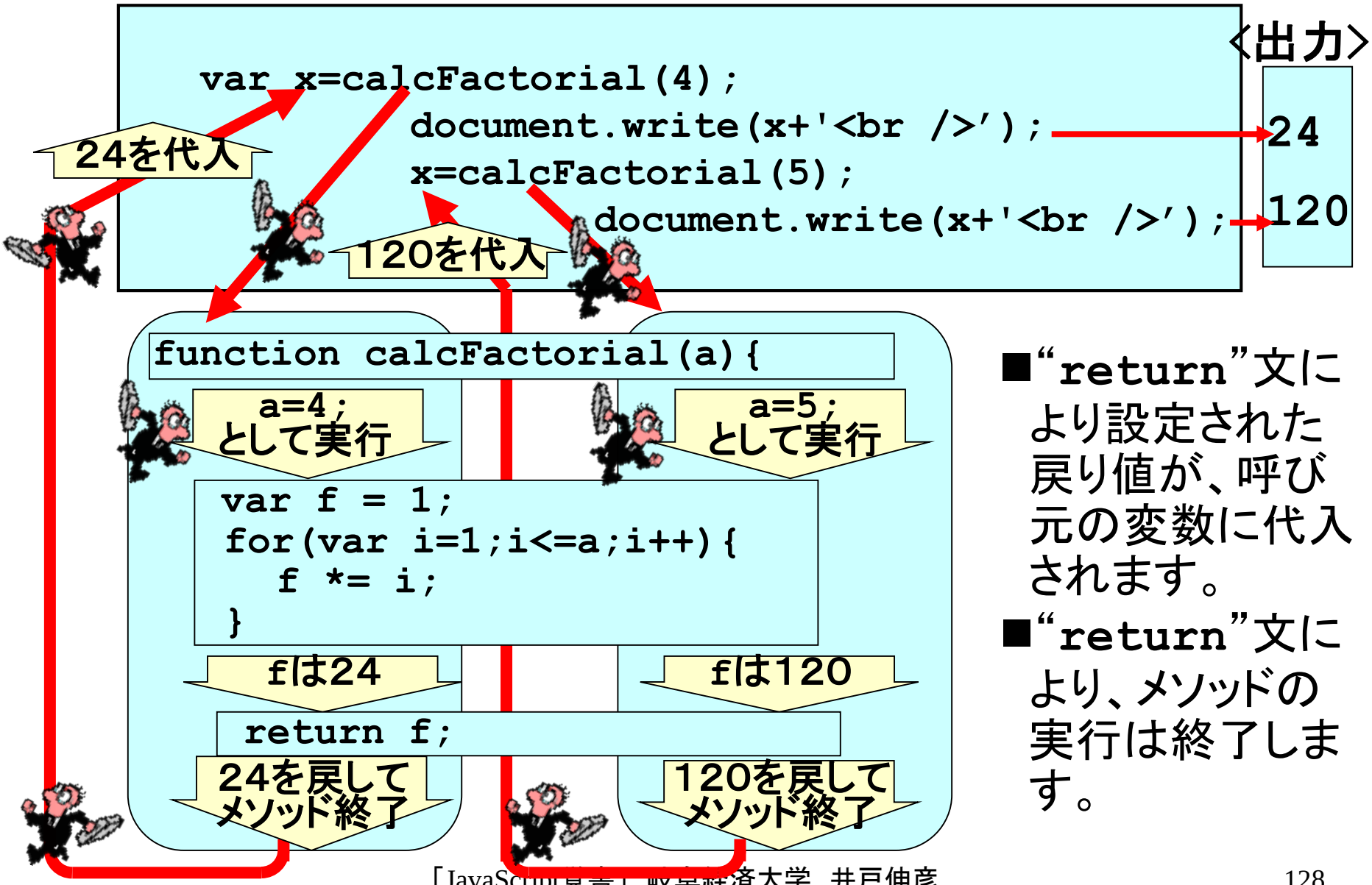
関数

リターン文



＜出力＞

（12．6．2）戻り値



■“return”文により設定された戻り値が、呼び元の変数に代入されます。

■“return”文により、メソッドの実行は終了します。

（12．6．3）戻り値がない場合

■スライド(12.2)のプログラムで見たように、戻り値がない関数もあります。

■戻り値のない関数での“**return**”文では、戻り値は返さず、単にメソッドを終了することになります。

```
function repeatString(str, n){

        // 処理の記述

        return;
}
```

戻り値は指定せず、
単にメソッドを終了する

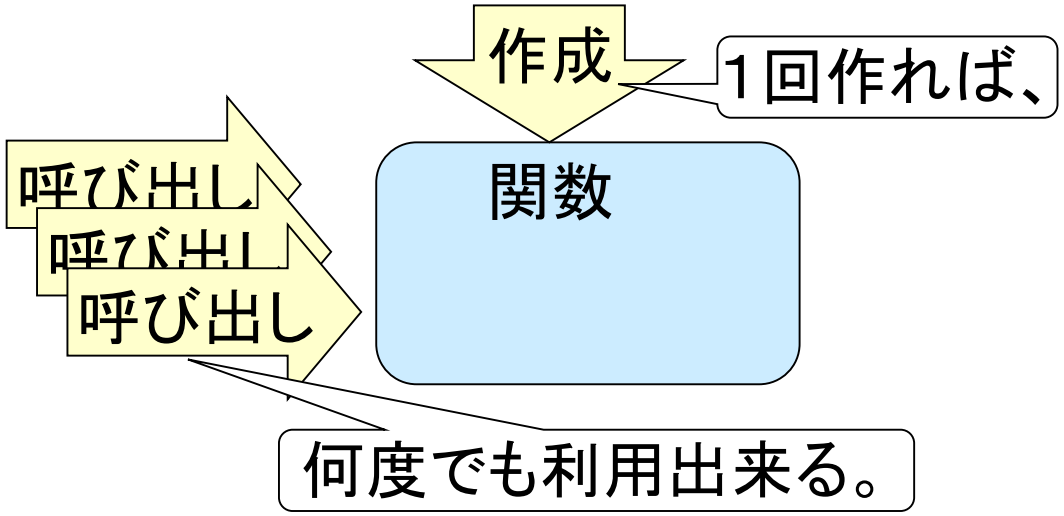
（12．7）どうして関数が必要か？

■スライド(12.2) のプログラムを、関数を使わずに作成すると、右下のようなプログラムになります。

■このプログラムでは、同じような実行文が繰り返し書かれており、これは無駄であるとともに、プログラムも見通しの悪いものになっています。

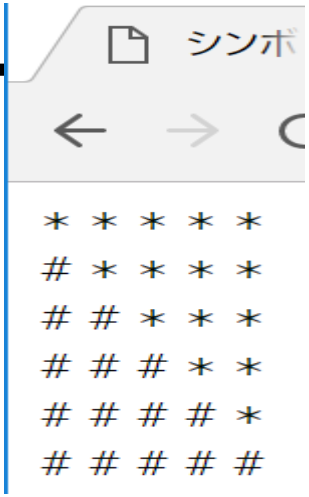
■関数を用いることで、効率的で見通しの良いプログラムを作成することが出来ます。

```
for(var i=0;i<3;i++){
    document.write("xy");
}
document.write("<br />");
for(int i=0;i<2;i++){
    document.write("yz");
}
document.write("<br />");
for(int i=0;i<3;i++){
    document.write("zz");
}
document.write("<br />");
```



（12．8）アスタリスクとシャープ

＜出力＞



■右のような出力を行うプログラムを作成してください。但し、次のような関数を使うこととします。

function drawSyms(w, sym)

- ◆この関数は文字列sを、w回だけ出力する。
- ◆例えば、symが”#”、wが5のとき、“#####”を出力する。

■解答例：

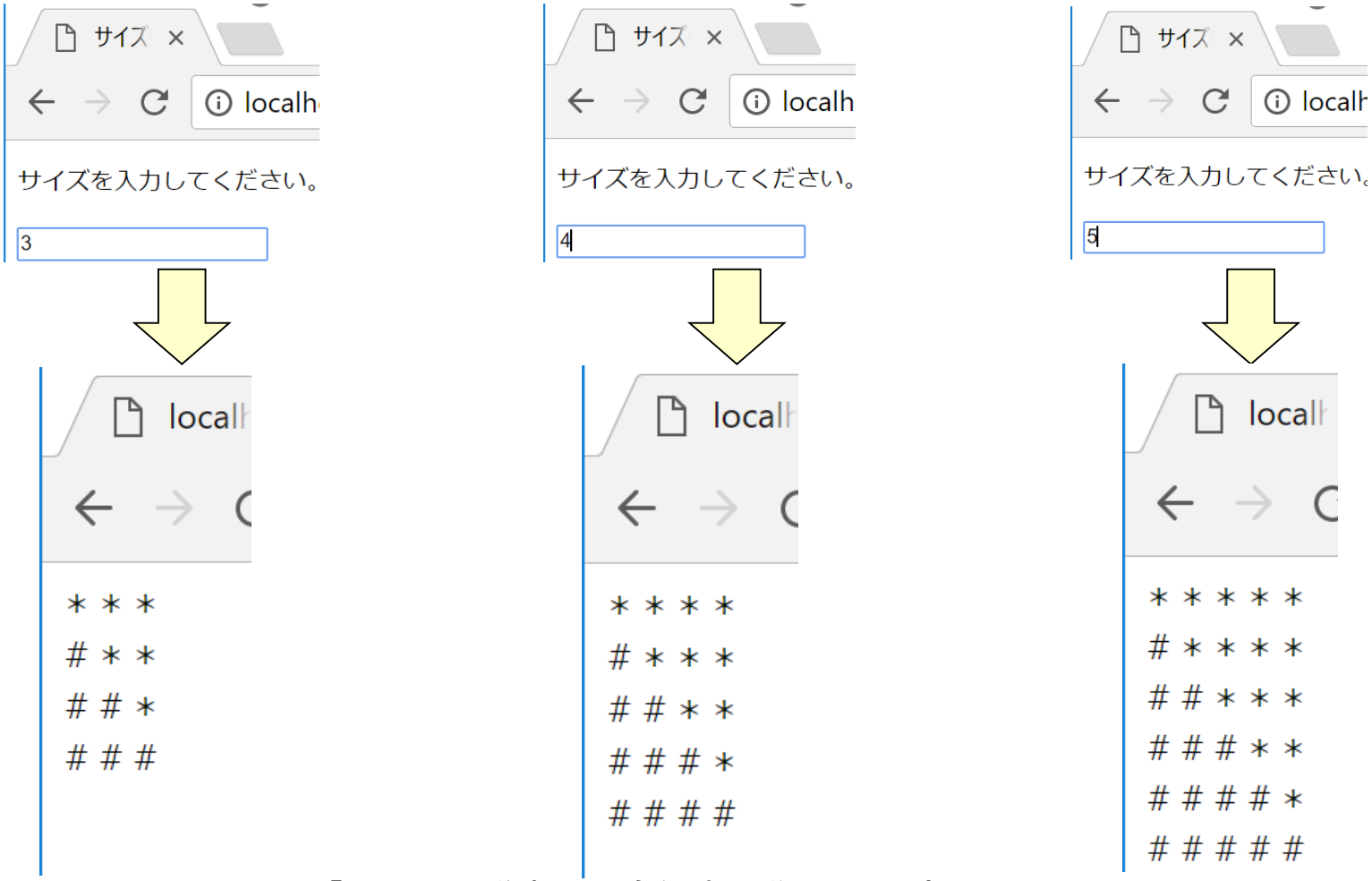
```
<body>
<script type="text/javascript">
for(var i=0;i<6;i++){
    drawSyms(i,'#');
    drawSyms(5-i,'*');
    document.write('<br />');
}
function drawSyms(w,sym){
    for(var i=0;i<w;i++){
        document.write(sym);
    }
}
</script>
</body>
```

等幅となるように、日本語の“＃”、“＊”を使っています

関数の中では、呼び元と同じ名前(i)の変数を宣言しても、両者は別物として扱われます

（12．9．1）サイズを変える

■文字による図形のサイズを、入力で変更できるようにしてください。



（12．9．2）サイズを変える

■解答例：

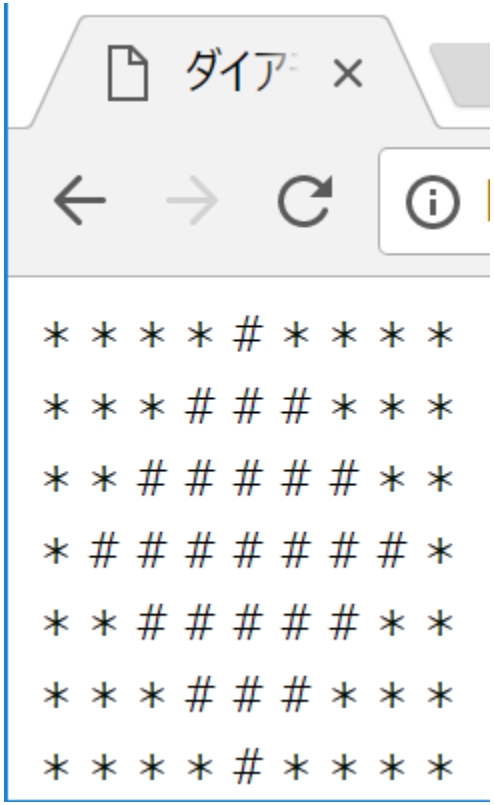
parseInt()は、文字を整数値に変換します。（スライド(6.9)参照）

```
<body>
<script type="text/javascript">
function draw(){
   var d = parseInt(document.form1.size.value);
   for(var i=0;i<d+1;i++){
      drawSyms(i,'#');
      drawSyms(d-i,'*');
      document.write('<br />');
   }
}
function drawSyms(w,sym){
   for(var i=0;i<w;i++){
      document.write(sym);
   }
}
</script>
<p>サイズを入力してください。</p>
<form name="form1">
<input type="text" name="size" onchange="draw();"/><br />
</form>
</body>
```

（12．10）課題

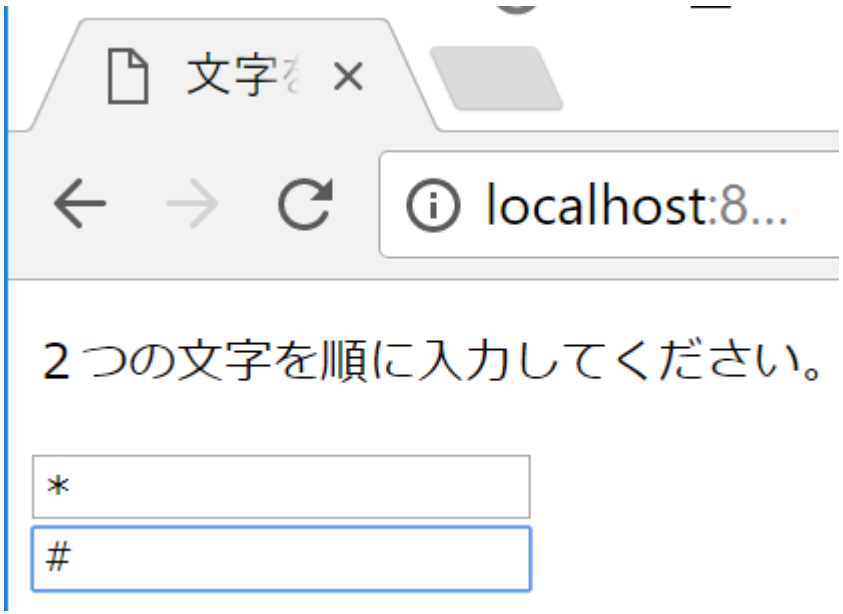
■次のような出力を行うプログラムを作成してください。

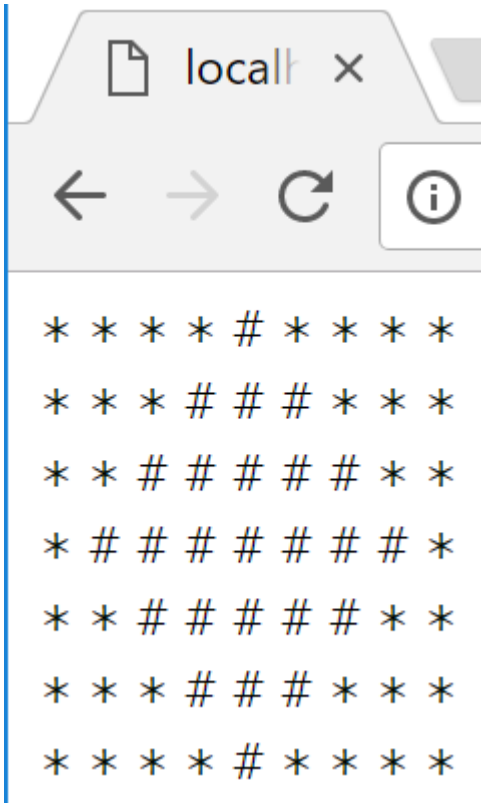
- 例題(12.9)のdrawSymsメソッドを使うこととします。



（12．11）課題

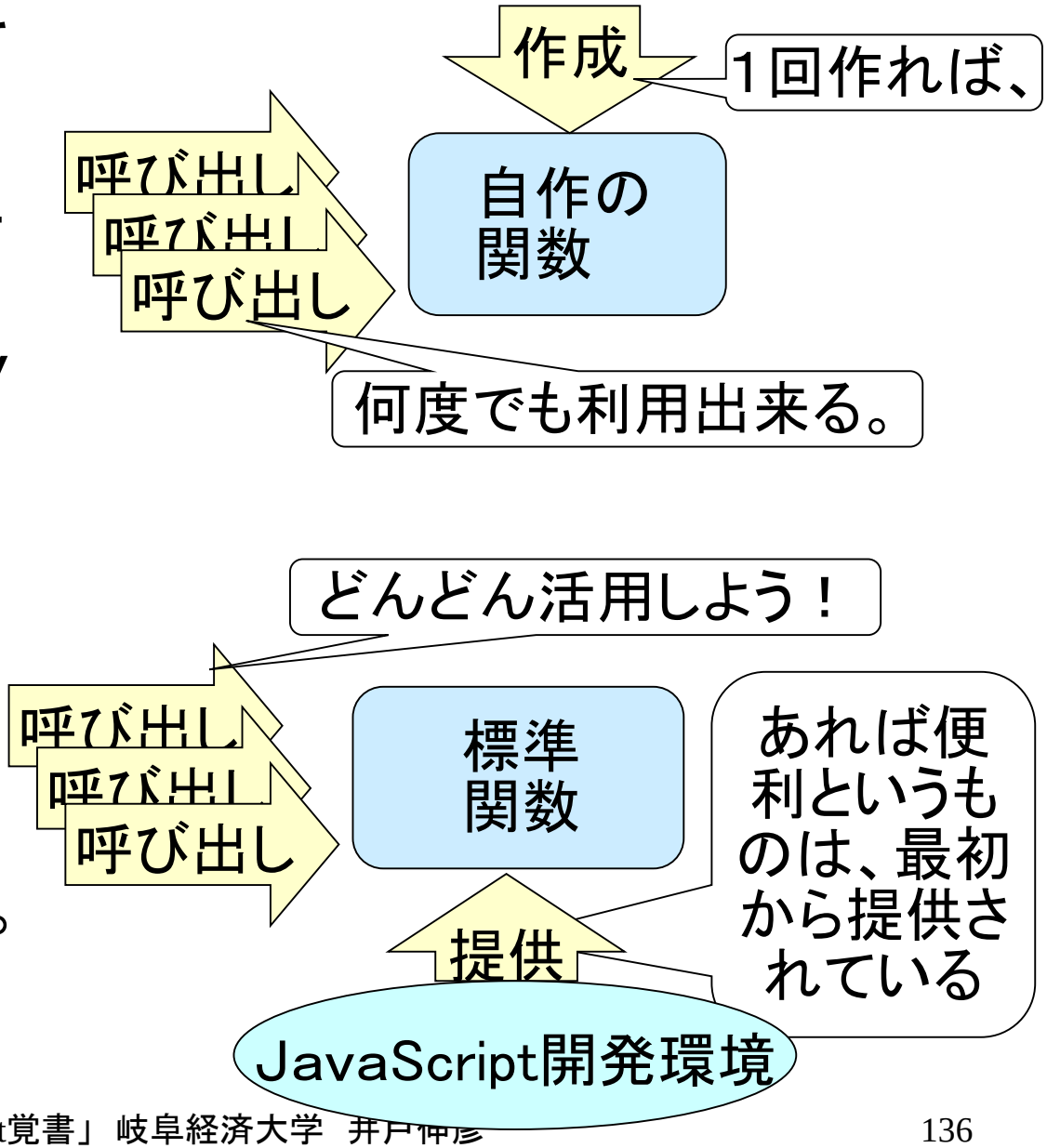
■下のような出力を行うプログラムを作成してください。





（13）標準の関数

- 関数は、一度作ってしまえば何度でも利用出きる点で役に立ちます。
- 標準の関数は、JavaScriptの開発環境にすでに用意されている、“あれば便利な”関数のことです。
- 以下、代表的な標準の関数の使い方を説明していきます。



（13．1）クラスとメソッド

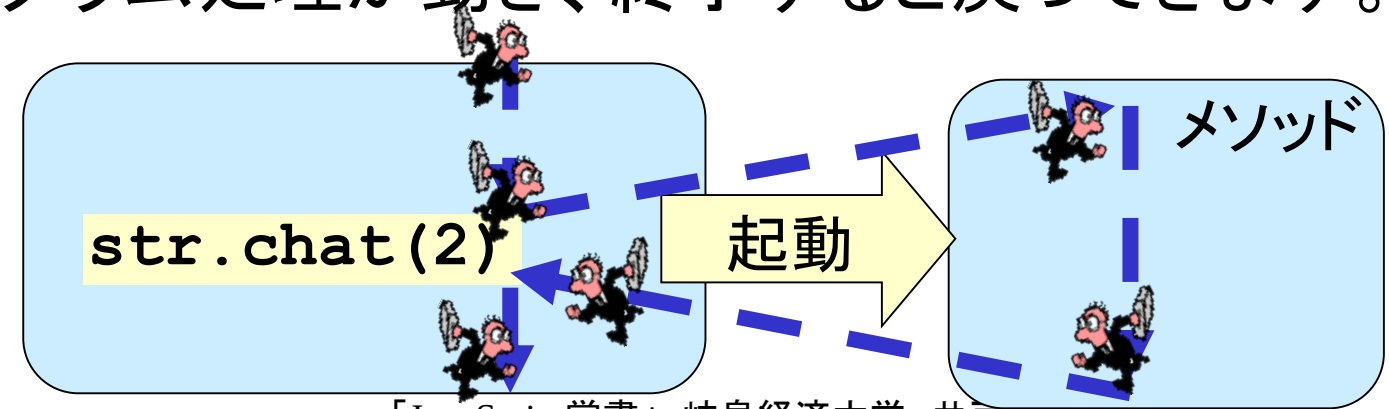
- このスライドではクラスについての説明は行いません。メソッドを利用する方法だけを説明します。

■メソッドは、変数（実際はクラスのインスタンス）に対して、次のような形で用います。

起動の形式： 変数.メソッド名

例：
```
c = str.charAt(2);
```

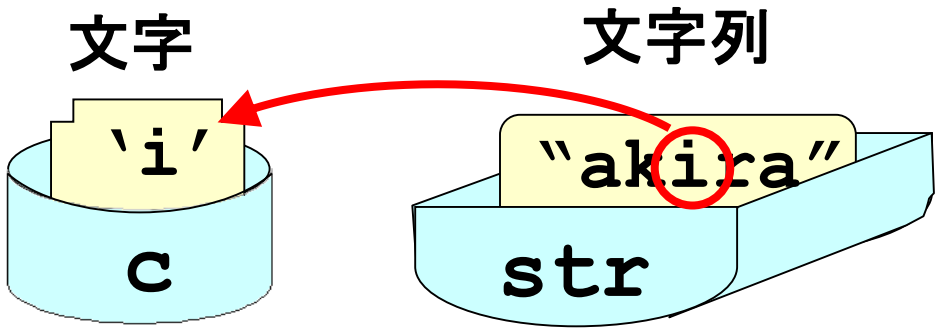
■上記のように起動すると、関数と同様に用意されたプログラム処理が動き、終了すると戻ってきます。



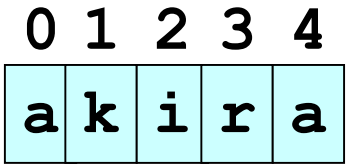
（13．2．1）文字の取得、文字列長の取得

■文字の取得(charAt())

```
var str = "akira";
var c = str.charAt(2);
// cには、'i'が入る
```



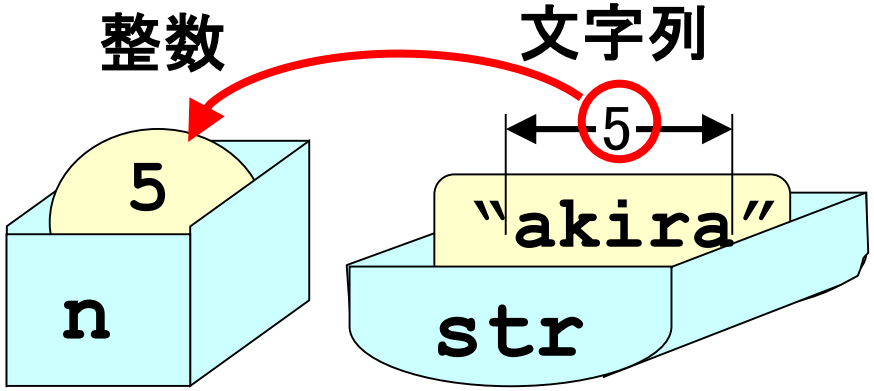
- 文字の数える数は、先頭が0になることに注意！
- 他のメソッドでも、文字は0から数えます。



■文字列長の取得（length）

- メソッドではなくプロパティなのですが、ついでに文字列の長さの求め方も覚えておきます。

```
var str = "akira";
var n = str.length;
// nには、数値の5が入る。
```

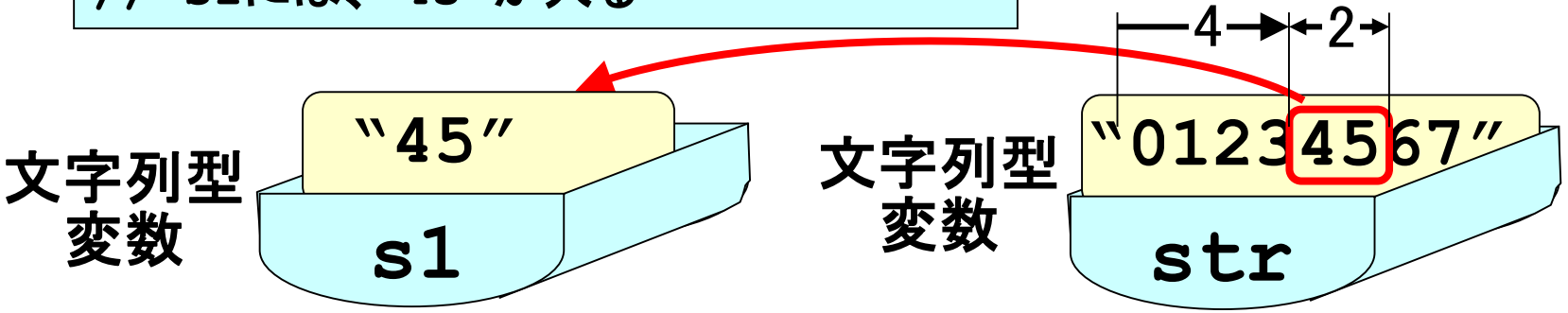


（13．2．2）部分文字列の取得

■substr(開始位置[,長さ])（長さを省略すると末尾まで）

```
var str = "01234567";
var s1 = str.substr(4,2);
// s1には、"45"が入る
```

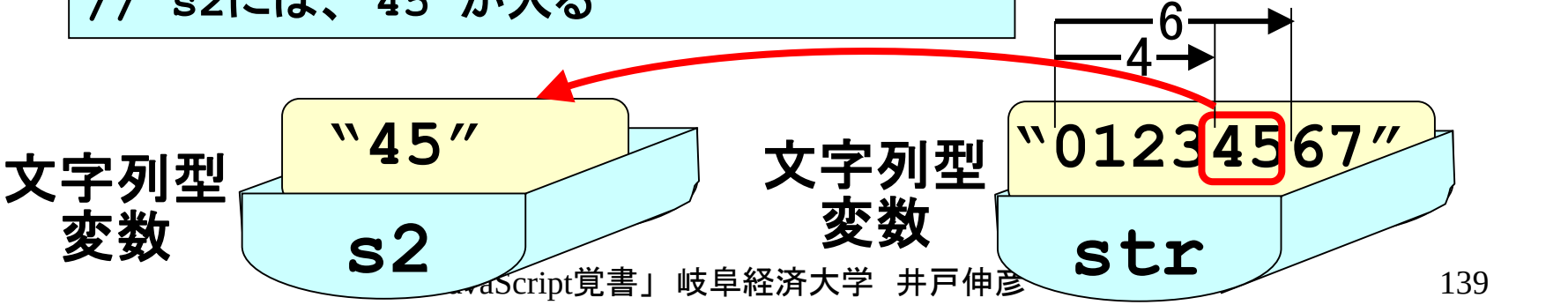
位置4から
2文字



■substring(開始位置、終了位置)

```
var str = "01234567";
var s2 = str.substring(4,6);
// s2には、"45"が入る
```

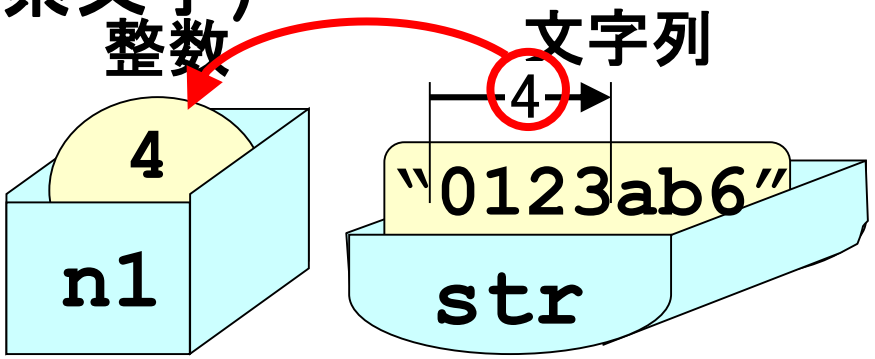
位置4から
位置5（6の前）まで



（13．2．3）文字列検索、文字置換

■文字列検索：indexOf（検索文字）

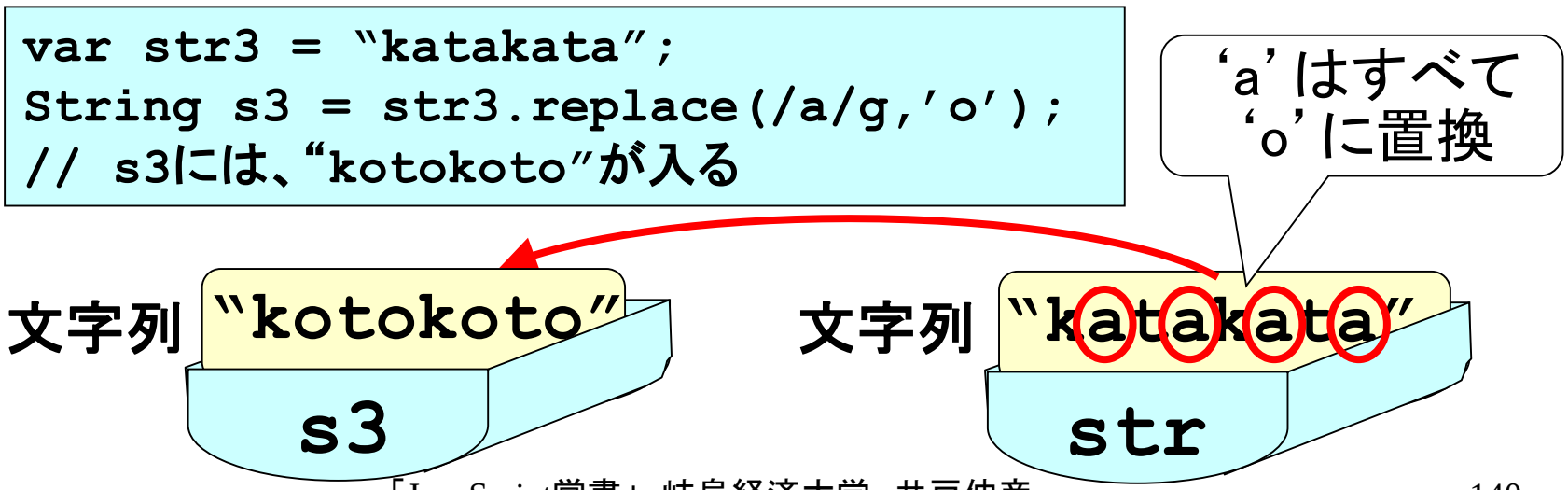
```
var str = '0123ab6';
var n1 = str.indexOf('ab');
// n1には、数値の4が入る。
var n2 = str.indexOf('cd');
// n2には、数値の-1が入る。
```



■置換：replace（/検索文字/g(*1)，置換文字））

(*1)実際には正規表現ですが、その説明は省略します。

```
var str3 = "katakata";
String s3 = str3.replace(/a/g,'o');
// s3には、"kotokoto"が入る
```



（13．2．4）その他のメソッド

■その他にも、文字列には豊富なメソッドが揃っています。「ありそうなものは全てある」と思ってください。

■ですから、「ありそうなもの」を自分で作ることは馬鹿馬鹿しいので避けてください。

■どのようなメソッドがあるかについては、Web利用すれば、メソッドの使い方を含めて楽に調べることができます（これは文字列以外の場合についても同じです）。

■例えば、googleにて“JavaScript”、“String”で検索してみてください。

（13．3．1）文字列と数値の変換

■文字列から数値への変換は、既にスライド(6.10)で見ました。

- 整数値への変換

```
var d = parseInt(ss);
```

- 浮動小数点数値への変換

```
var f = parseFloat(ss);
```

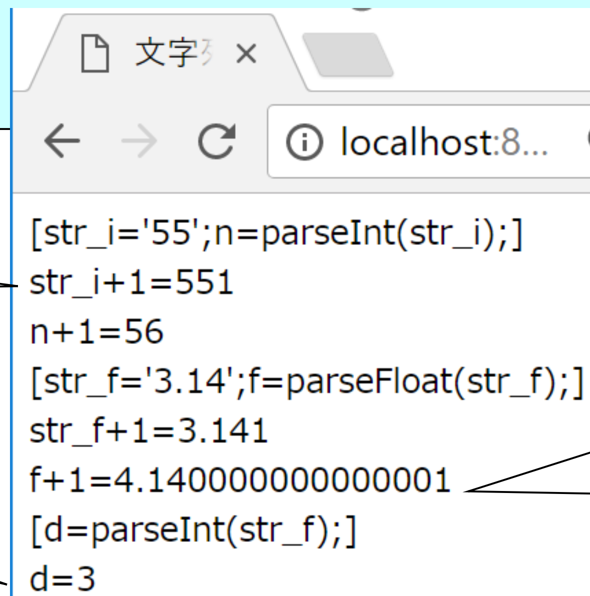
■反対に数値を文字列に変換する場合は次のように行います。

```
var ss = d+'';
```

```
var ss = f+'';
```

（13．3．2）変換の例

```
<body>
<script type="text/javascript">
var str_i='55';
var n=parseInt(str_i);
document.write("[str_i='55';n=parseInt(str_i);]<br />");
document.write("str_i+1="+(str_i+1)+"<br />");
document.write("n+1="+(n+1)+"<br />");
var str_f='3.14';
var f=parseFloat(str_f);
document.write("[str_f='3.14';f=parseFloat(str_f);]<br />");
document.write("str_f+1="+(str_f+1)+"<br />");
document.write("f+1="+(f+1)+"<br />");
var d=parseInt(str_f);
document.write("[d=parseInt(str_f);]<br />");
document.write("d="+d);
</script>
</body>
```



文字列として結合される

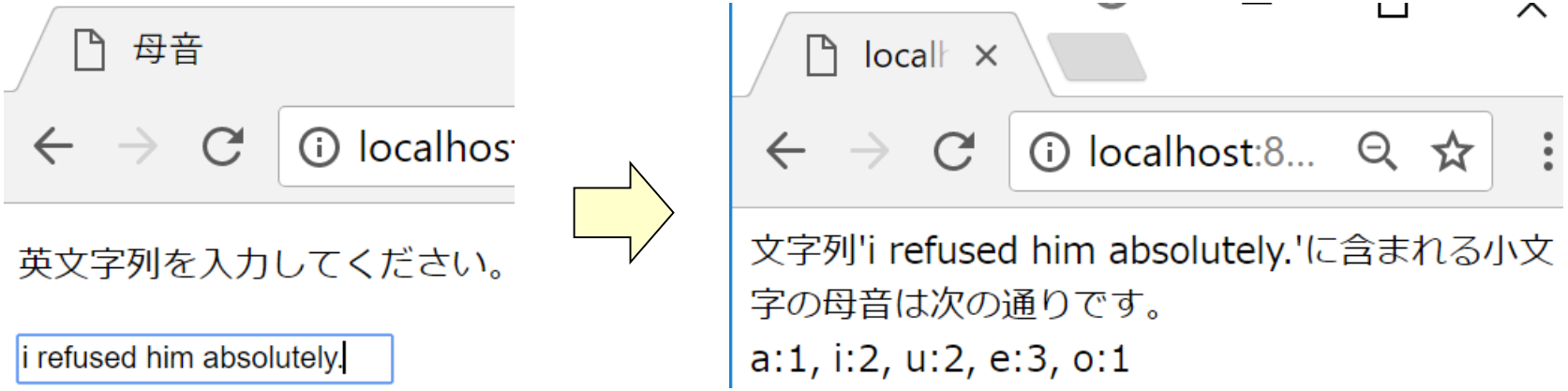
浮動小数の計算には誤差が伴う

切り捨てになる

（13．5）課題

■課題13－5－1

- 英数字の文字列を入力して、小文字母（”a”,”i”,”u”,”e”,”o”）がそれぞれいくつ含まれるかを出力するプログラムを作ってください。



■課題13－5－2

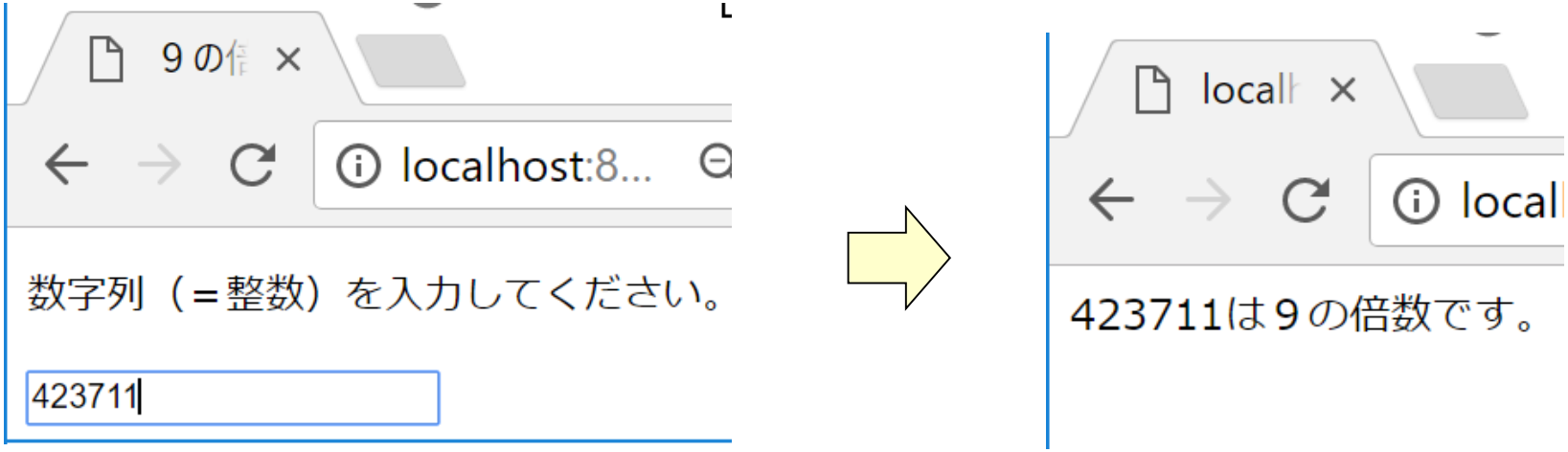
- 上記のプログラムと同じことを、配列を用いて行ってください（結果表示で最後にカンマ(,)があっても良いとします）。

```
var vowels = ['a','i','u','e','o'];
var counters = [0,0,0,0,0];
```

（13．6）課題

■数字による長い文字列を入力して、それが9の倍数であるか否かを出力するプログラムを作成してください。但し、9の倍数であることの判定は、次のように行うこととします。すなわち、%（あまり）は使わないこととします。

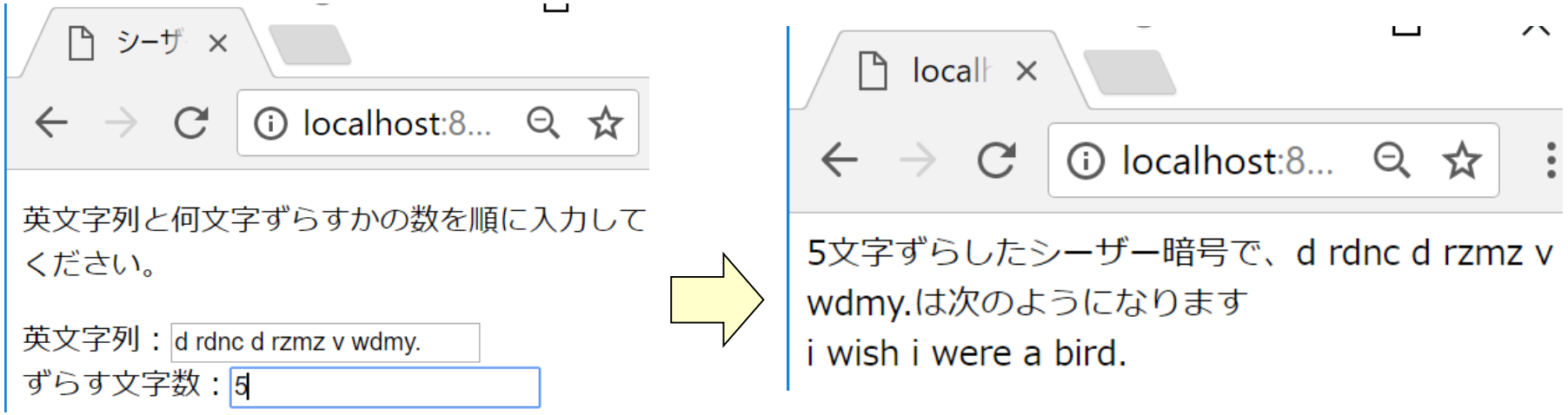
- 423711は、9の倍数である。
  - ◆4＋2＋3＋7＋1＋1＝18
  - ◆1＋8＝9　⇒　9となれば、9の倍数
- 385772は、9の倍数でない。
  - ◆3＋8＋5＋7＋7＋2＝32
  - ◆3＋2＝5　⇒　9未満となれば、9の倍数ではない。



（13．7）課題

■課題：暗号解読

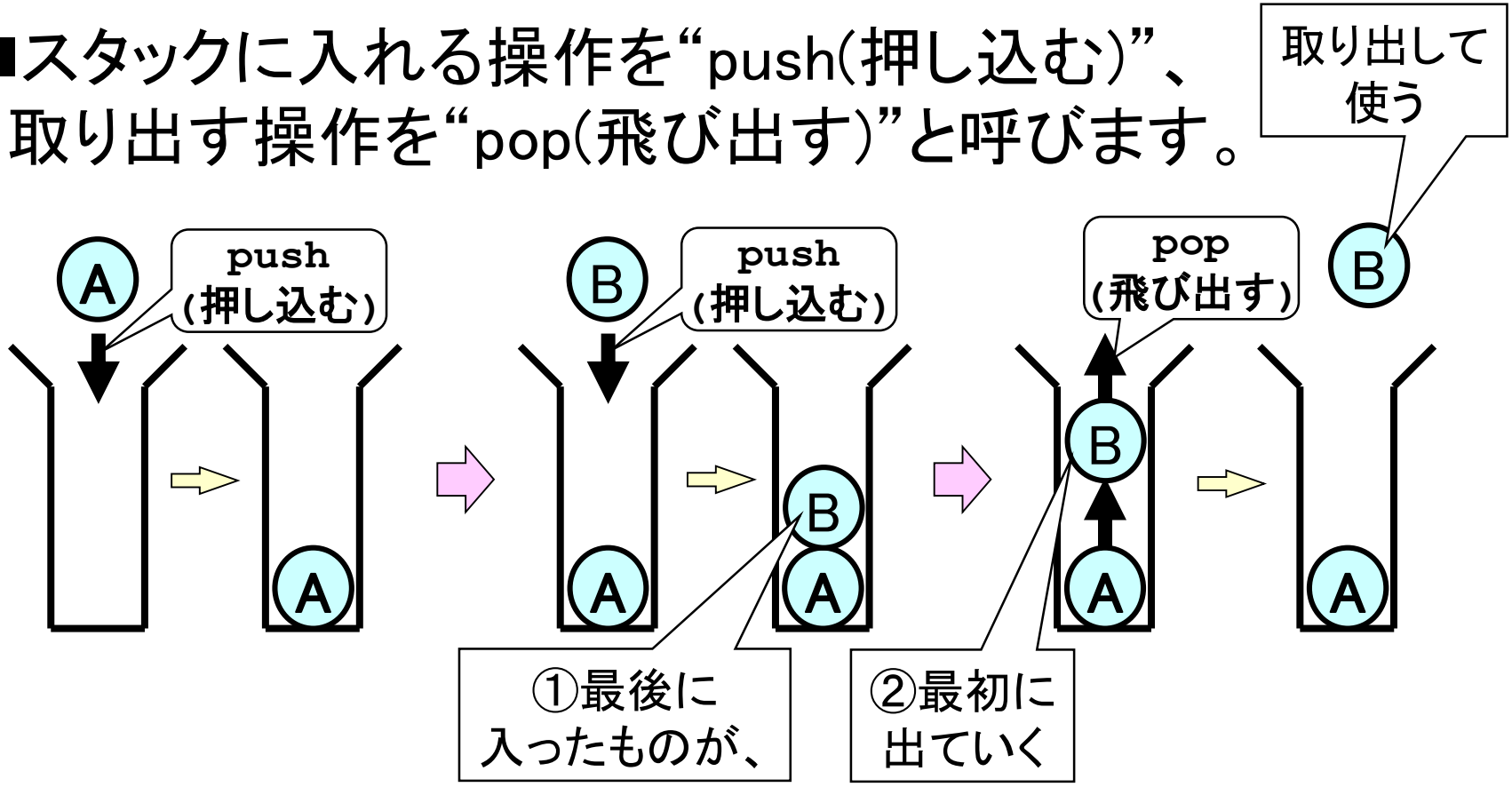
- 次のように、アルファベットの順番をずらして得られる変換による暗号を、シーザー暗号といいます。
- 文字列と数字nとを入力して、文字列のうちの小文字のアルファベットをn個ずらして得られるシーザー暗号を出力するプログラムを作成してください（小文字のアルファベット以外の入力はそのまま出力します）。



（13．8）配列への操作（push,pop）

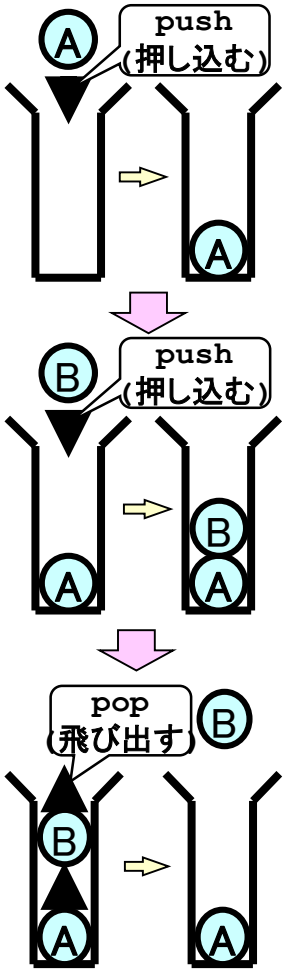
■「最後に入れたデータを最初に取り出す配列」は、良く使われるデータ構造であり、“スタック”とか“LIFO(Last In First Out)”と呼ばれます。

■スタックに入れる操作を“push(押し込む)”、取り出す操作を“pop(飛び出す)”と呼びます。



（13．8．1）JavaScriptでの記述

■JavaScriptではメソッドを用いてスタックに当たる操作を配列に行うことが出来ます。




```
var ar = new Array();
```

“ar”という名前の配列を生成する

```
ar.push('A');
```

配列“ar”の中身は、['A']

```
ar.push('B');
```

配列“ar”の中身は、['A','B']

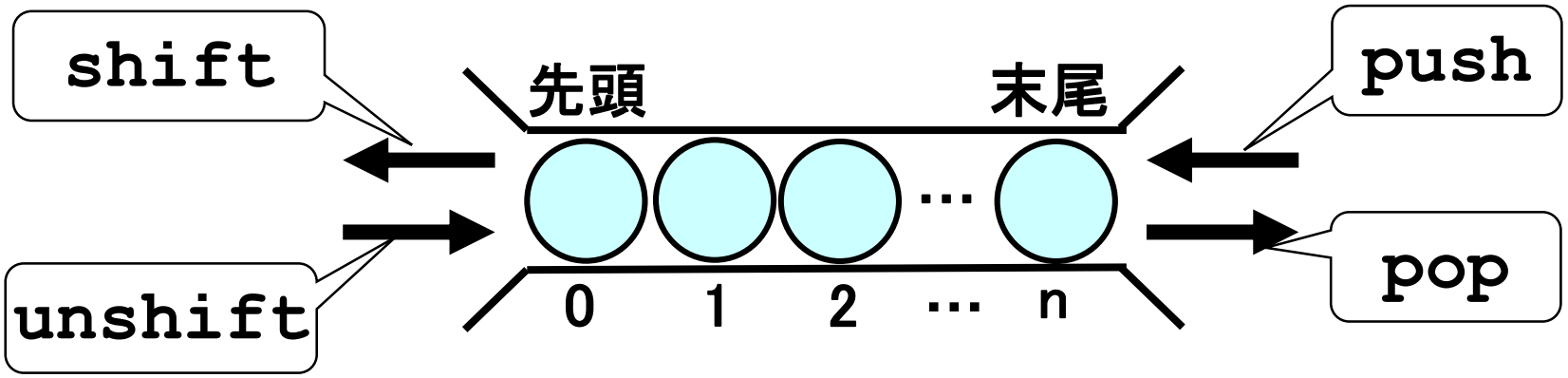
変数”temp”には、'B'が入る

```
var temp = ar.pop();
```

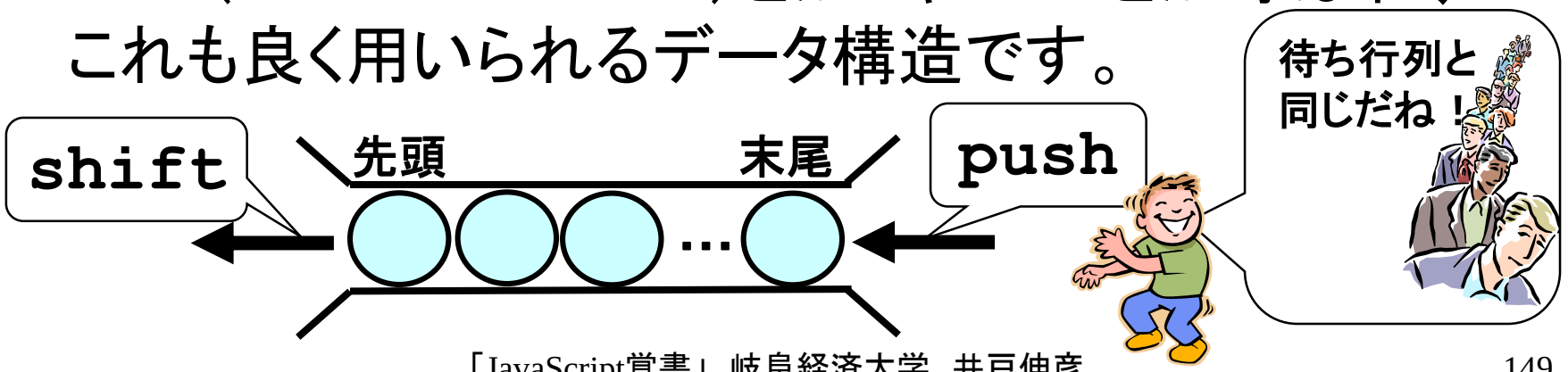
配列“ar”の中身は、['A']

（13．8．2）先頭に対する操作

■push/popは配列の末尾に対する操作ですが、同様に先頭に対する操作(shift/unshift)もあります。



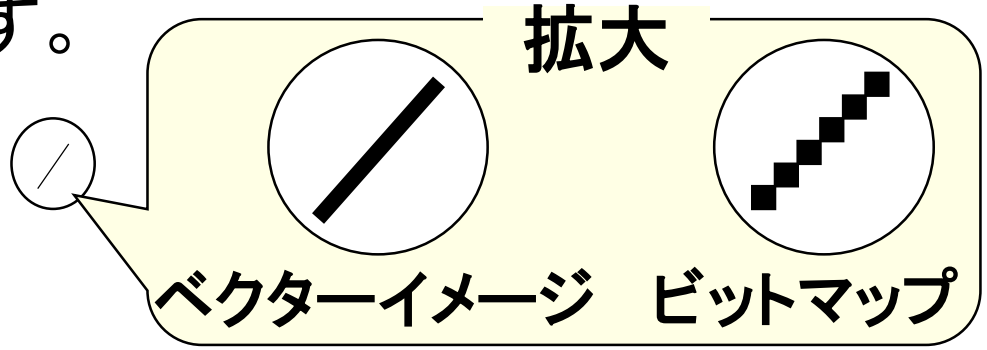
■「最初に入れたデータを最初に取り出す配列」は、FIFO（First In First Out）とか“キュー”とか呼ばれ、これも良く用いられるデータ構造です。



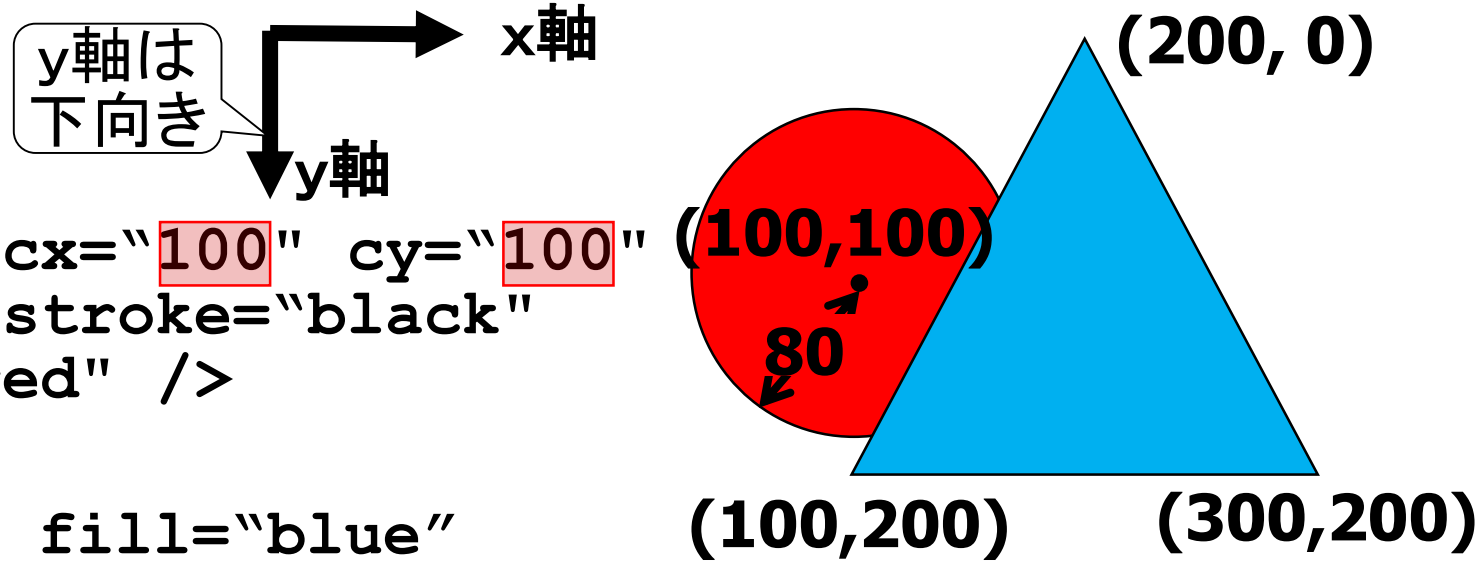
（14）SVG(Scalable Vector Graph)

■SVGは、HTML5で用いられている二次元ベクターイメージ用のデータ形式です。

- ベクターイメージ：ビットマップで無く、拡大してもドットが見えてこない。



■例：




```
<circle cx="100" cy="100"
 r="80" stroke="black"
 fill="red" />
```

```
<polygon fill="blue"
   stroke="black"
  points="200,0 300,200 100,200" />
```

（14．1）JavaScriptからの操作

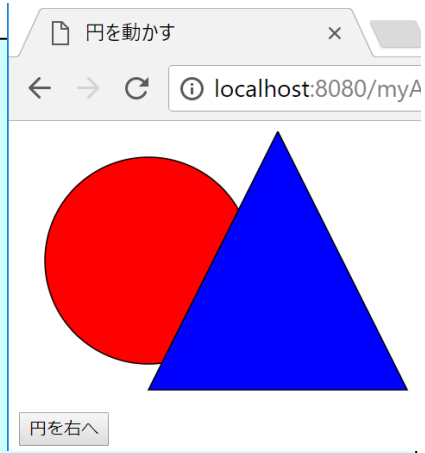
■ボタンをクリックすると、円が右側へ動くプログラムです。DOM（スライド(4.13)参照）経由で操作しています。

```
<script type="text/javascript">
function moveCircle(){

    var cir=document.getElementById('c_1');

    var x=parseInt(cir.getAttribute('cx'));
    cir.setAttribute('cx',x+3);
}
</script>
<svg xmlns="http://www.w3.org/2000/svg"
      width="512" height="210" viewBox="0 0 512 210">
    <g id="g1">
       <circle id="c_1" cx="100" cy="100" r="80"
               stroke="black"  fill="red" />
       <polygon fill="blue" stroke="black"
                points="200,0 300,200 100,200" />
    </g>
</svg>
<form name="form1">
<input type="button" value="円を右へ" onclick="moveCircle();"/>
</form>
```

円のオブジェクトを取得

円の中心のx座標値を取得

取得したx座標値に3を足して設定



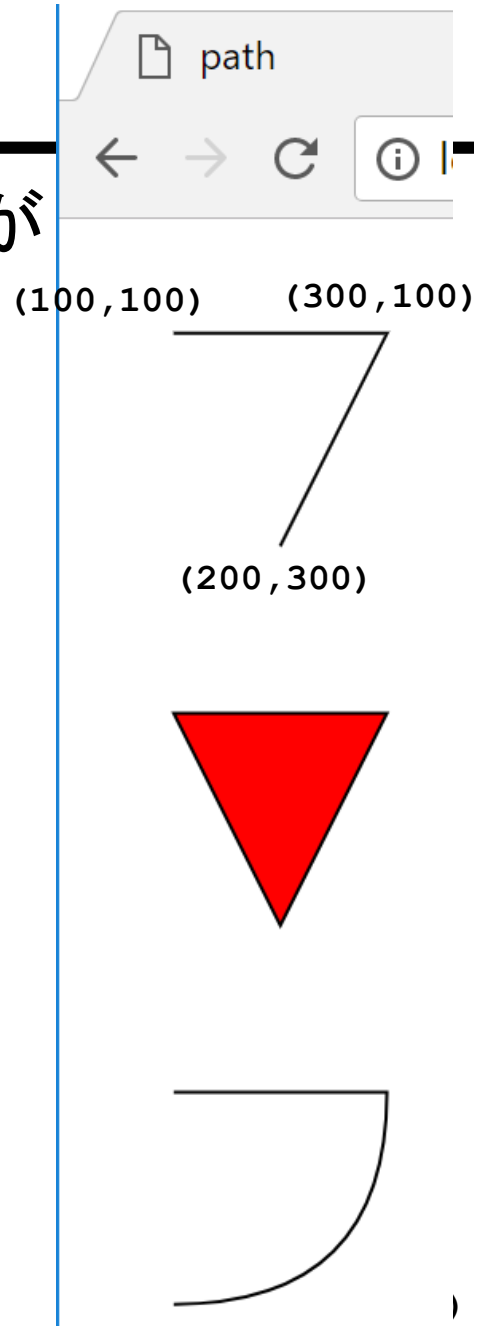
（14．2）path

■色々な形状をベジェ曲線により描くことが出来るのが、path要素です。

```
<svg xmlns="http://www.w3. 50">
 <g>
  <path
    d="M100,100 L300,100 L200,300"
    fill="none" stroke="black"
    stroke-width="3" />
   </g>
</svg>
<br />
<svg xmlns="http://www.w3. 350">
 <g>
  <path
    d="M100,100 L300,100 L200,300 z"
    fill="red" stroke="black"
    stroke-width="3" />
 </g>
</svg>
<br />
<svg xmlns="http://www.w3. 350">
 <g>
  <path
    d="M100,100 L300,100 Q300,300 100,300"
    fill="none" stroke="black"
    stroke-width="3" />
 </g>
</svg>
```

z:パスを閉じます

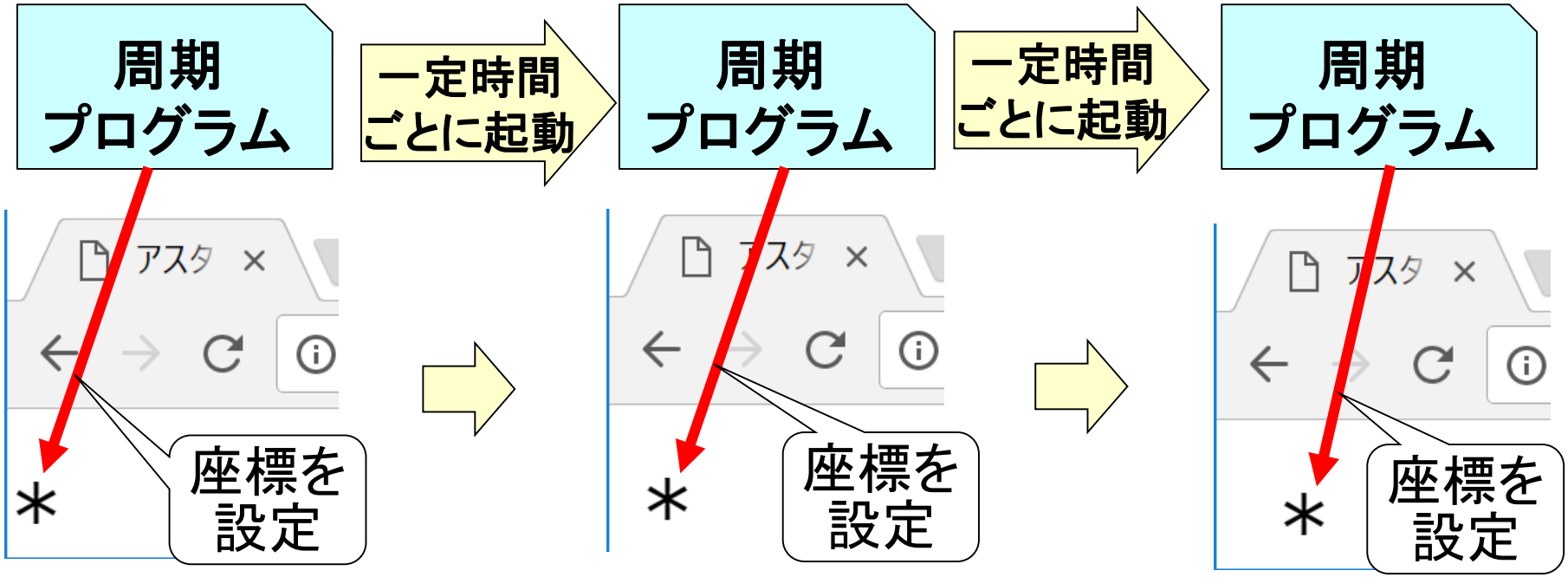
Q:曲線も描けます



（15）周期プログラム

■周期プログラムとは、一定時間間隔で起動されるプログラムです。

■周期プログラムにて、DOMを通じてHTMLの要素の座標値を少しずつ変化させると、アニメーションのように動かすことが出来ます。



（15．1）アスタリスクのプログラム

■周期プログラムは、“setInterval()”という関数により起動することが出来ます。

```
<body>
<script type="text/javascript">
setInterval('moveAsterisk()',50);
var x=0;
function moveAsterisk(){
   var obj=document.getElementById("star0");
   obj.style="position:absolute; left:"+x+"px";
   x+=5;
   if(x>=200){
      x=0;
   }
}
</script>
<div id="star0">*</div>
</body>
```

周期プログラムを起動（50msec毎）

オブジェクトを取得

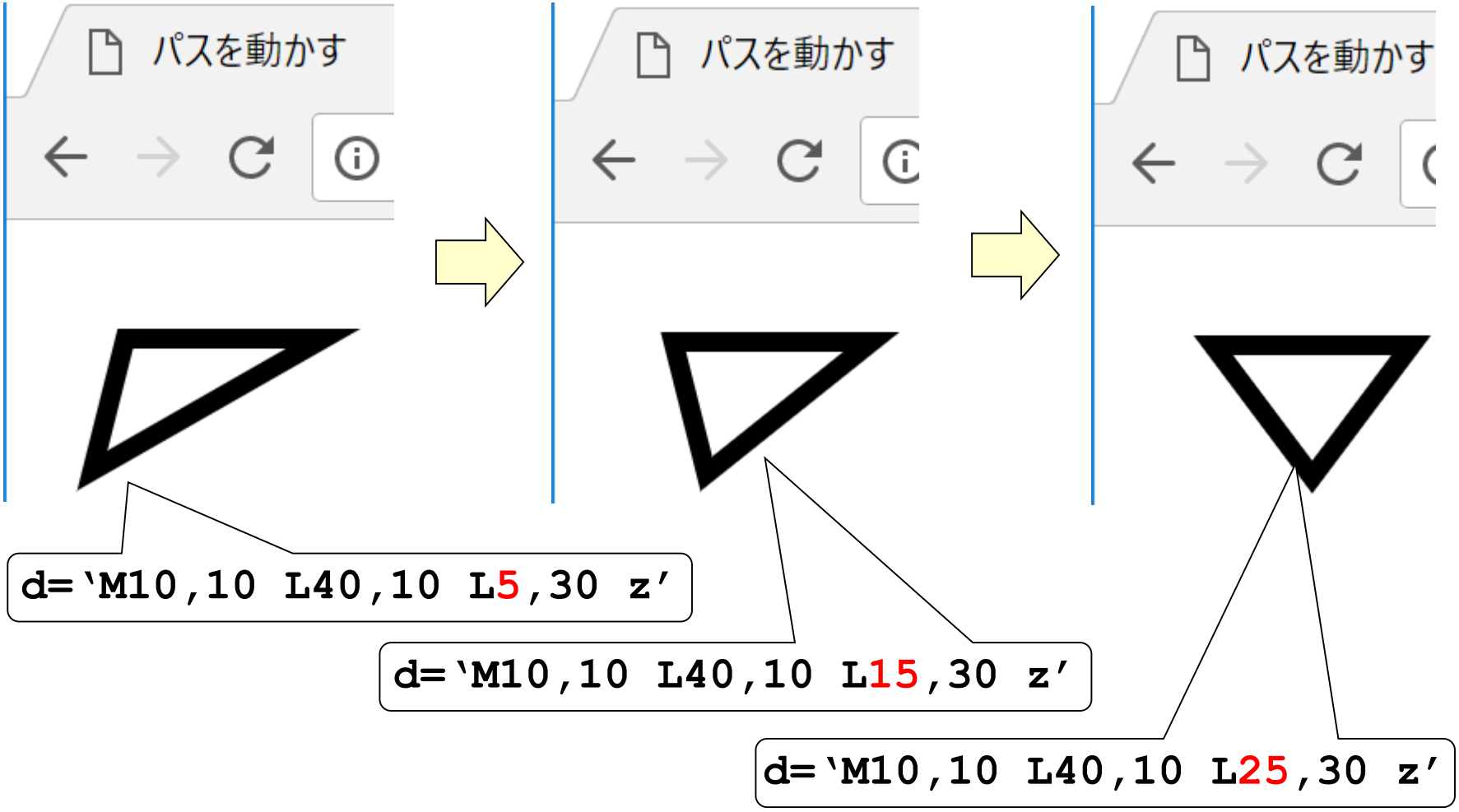
座標値を更新

座標値を設定（CSSという記法を用いています）

操作対象のオブジェクト

（15．2）pathの操作

■円を動かした際と同様に、属性”d”の設定によりパスを操作できます。



（15．3）pathの操作のプログラム

```
<body>
<script type="text/javascript">
setInterval('movePath()',50);
var x=0;
function movePath(){
   var obj=document.getElementById("path0");
   obj.setAttribute('d',
                    'M10,10 L40,10 L'+x+',30 z');
   x+=5;
   if(x>=50){
      x=0;
   }
}
</script>
<svg xmlns="http://www.w3.org/2000/svg"
width="350" height="350" viewBox="0 0 350 350">
 <g>
  <path id="path0" fill="none" stroke="black"/>
 </g>
</svg>
```

属性”d”を設定

（15．4）注意！：setIntervalのパラメータ

■ブラウザ（Chrome）で動くjavascriptでは、setIntervalのパラメータを次のように記しました。

```
setInterval('moveAsterisk()',50);
```

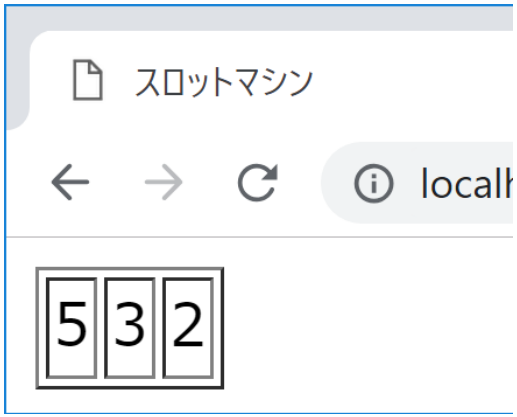
■node.jsでは、次のように記すようです。

```
setInterval('moveAsterisk',50);
```

■上記の違いに関しての情報は無いのですが、実際動作させると一方の記し方は他方で動かないようです（2018.11.05現在）。

（15．5．1）スロットマシン

■表示される数字がどんどん変わっていきます。



■スライド(17.1)等を参考にして、ボタン操作で止める機能を作ってください。

（15．5．2）スロットマシンのプログラム

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>スロットマシン</title>
</head>
<body onload='init();'>        <!-- bodyがロードされたら、”init()〃を呼ぶ-->
<script type="text/javascript">
var d=[0,0,0]; // スロットで表示する番号を求めるための数
var s=[2,3,5]; // スロットの各列を変更する周期
var timer=0;   // 経過時間を数えるタイマー
function init(){
    setInterval('changeNumbers()',50);// 5ms毎にchangeNumbersを起動
}
function changeNumbers(){
    timer++;    // タイマーの値を増やす。
    for(var i=0;i<d.length;i++){      // スロットの各列について処理
        if(timer%s[i]==0){                    // 周期である場合
            d[i]++;                           // 表示する数を増やす
            d[i]%=10;                         // 0から9までにする
        }
        var obj=document.getElementById("d"+i);
        obj.innerText=d[i];                   // 数を設定
    }
}
</script>
<table border='1'>
<tr><td id="d0">0</td><td id="d1">0</td><td id="d2">0</td></tr>
</table>
</body>
</html>
```

（16．1）乱数

■次のようにして、0以上1未満の乱数を発生させます。

```
■  var random = Math.random();
```

0以上1未満の乱数

```
for(var i=0;i<5;i++){
  console.log(Math.random());
}
```

＜出力＞

```
0.21994744242728792
0.7563056356044705
0.9366330372956264
0.4782882145211 79
0.8513166363922027
```

■randomメソッドを用いて、様々な範囲・値の乱数を作成できます。

- Math.floorは、切り捨てを行う関数です。

```
var x = 10*Math.random();                  // 0.0から10.0未満の乱数
var iran = Math.floor(10*Math.random()); // 0、1、……、9の乱数
var jran = Math.floor(1+6*Math.random());// 1、2、……、6の乱数
var kran = Math.floor(2*Math.random());    // 0か1の乱数
```

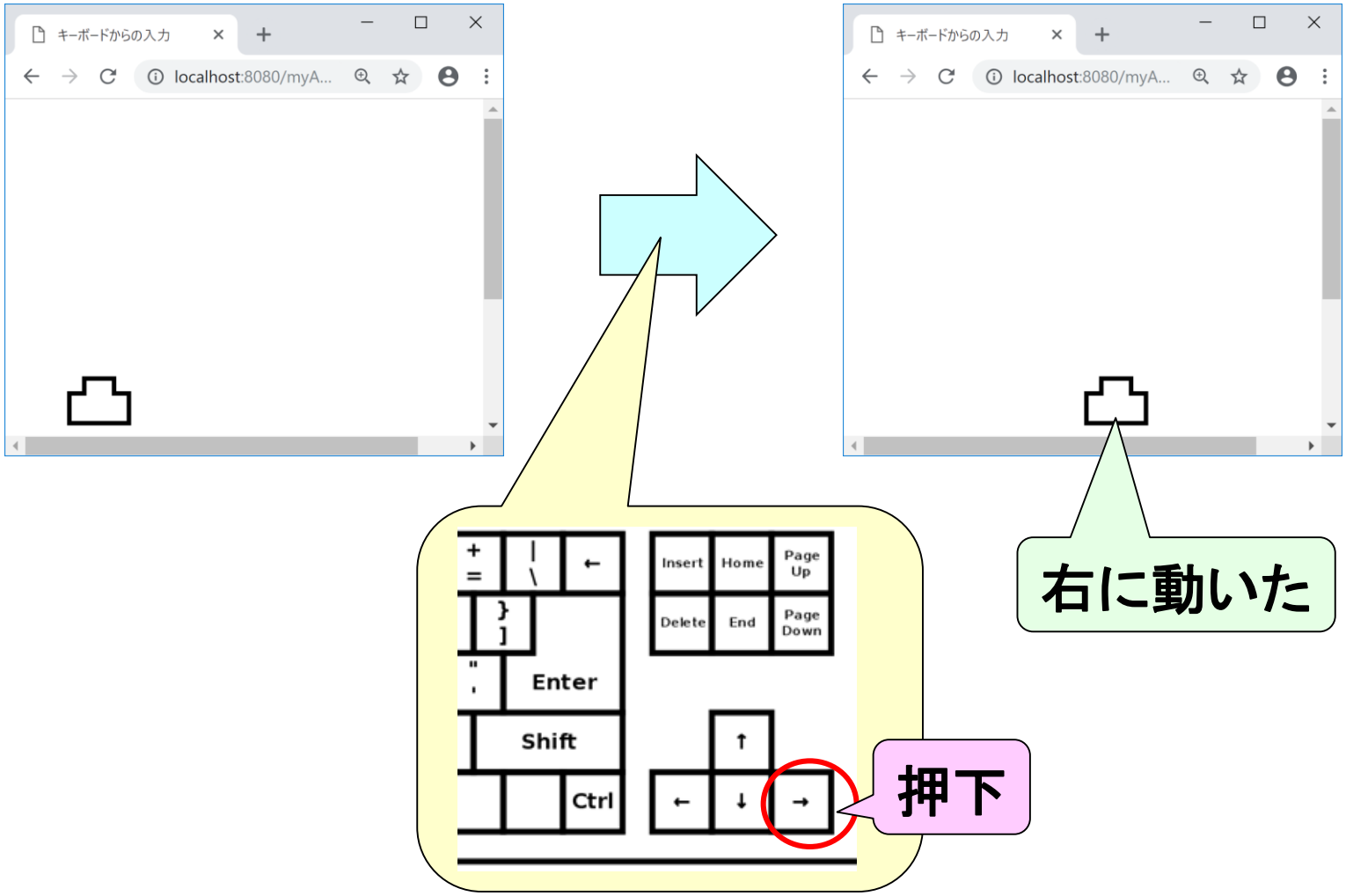
（16．2）課題

■（15）で動かしたアスタリスクのプログラムを、ランダムな位置に動くように変更してください。

■コンピューターとじゃんけんを行うプログラムを作成してください。

（17）キーボードからの入力

■左右のカーソルキーを押下すると、図形が動きます。



（17．1）キーボード入力のプログラム

```
<body onload="init();">        <!-- bodyがロードされたら、"init()"を呼ぶ-->
<script type="text/javascript">
var shape_x=[-10,-10,-20,-20, 20, 20, 10, 10]; // 図形のx座標
var shape_y=[-20,-10,-10, 10, 10,-10,-10,-20]; // 図形のy座標
var x=100; var y=200;  // 図形を描画する位置
function init(){
    document.onkeydown = keydown;      // キーが押されたら、関数keydownを呼ぶ。
    setPath(x,y);       // 初期の位置に図形を描画
}
function keydown(event){
    if(event.key==‘ArrowRight’){ x+=5; // 右矢印ならば位置を右に5だけずらす
    }else if(event.key=='ArrowLeft'){ x-=5;}
    setPath(x,y);                              // パスとして図形を描画
    event.preventDefault();                    // 画面が動くことを防ぐ
}
function setPath(x,y){
    var obj=document.getElementById("path0");
    var pathStr='M'+(shape_x[0]+x)+','+(shape_y[0]+y)+' ';
    for(var i=1;i<shape_x.length;i++){
        pathStr+='L'+(shape_x[i]+x)+','+(shape_y[i]+y)+' ';
    }
    pathStr+='z';
    obj.setAttribute('d',pathStr);
}
</script>
<svg xmlns="http://www.w3.org/2000/svg" width="350" height="350"
                                                    viewBox="0 0 350 350">
    <g><path id="path0" fill="none" stroke="black" stroke-width="3" />
    </g>
</svg>
</body>
```

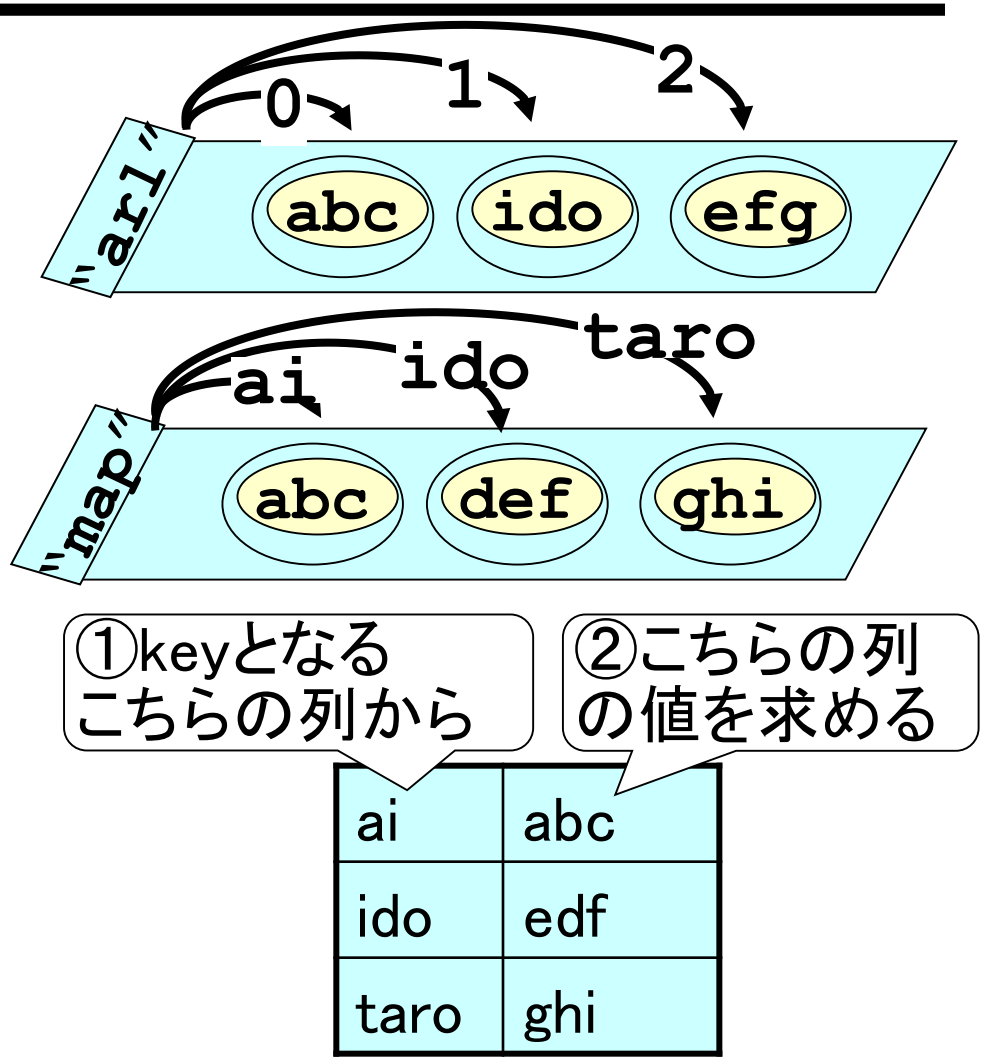
(15.3)参照

（18）オブジェクト（連想配列）

■配列は、非負整数（0,1,2,...）で索引するものでした。

■オブジェクトでは、文字列などの引数で、値を索引することができます。

■すわなち、オブジェクトは2列の表であり、キー(key)となる1列目の値から2列目の値を求めることが出来ます。



■JavaScriptではこのような“表”をオブジェクトとして実現していますが、普通のプログラミング言語では、“連想配列”、もしくは“ハッシュ”と呼びます。

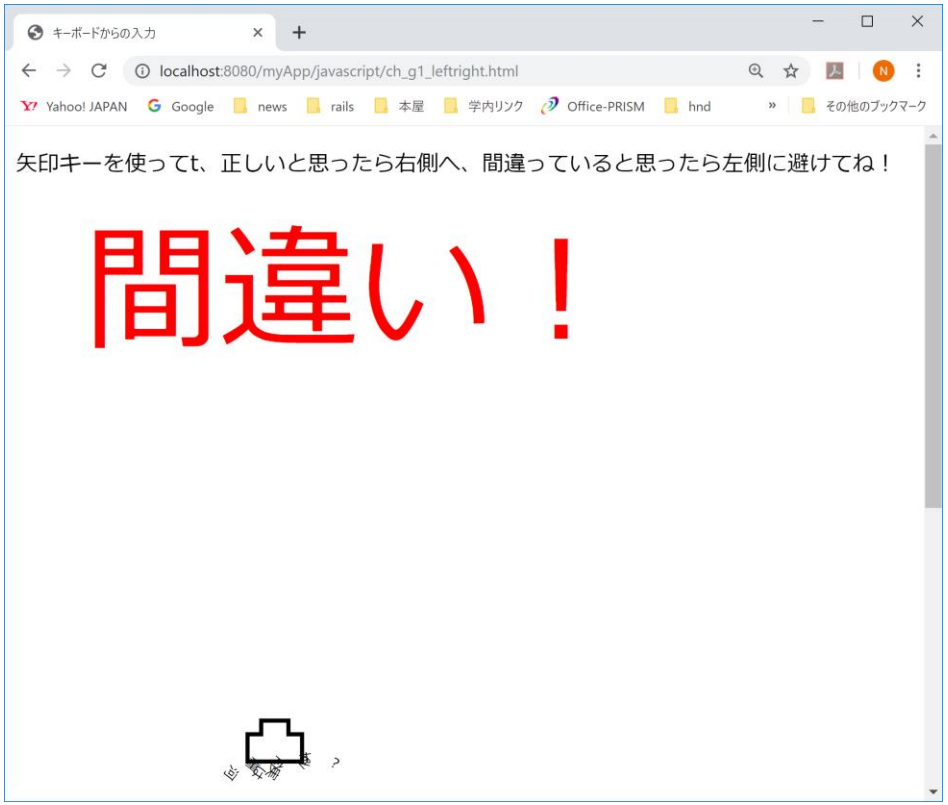
（18．1）オブジェクトの作り方

■未

(G1)クイズ レフトライト

■次のファイルをネットワークドライブからインポートしてアクセスしてください。

ch_g1_leftright.html



（G2）動体視力

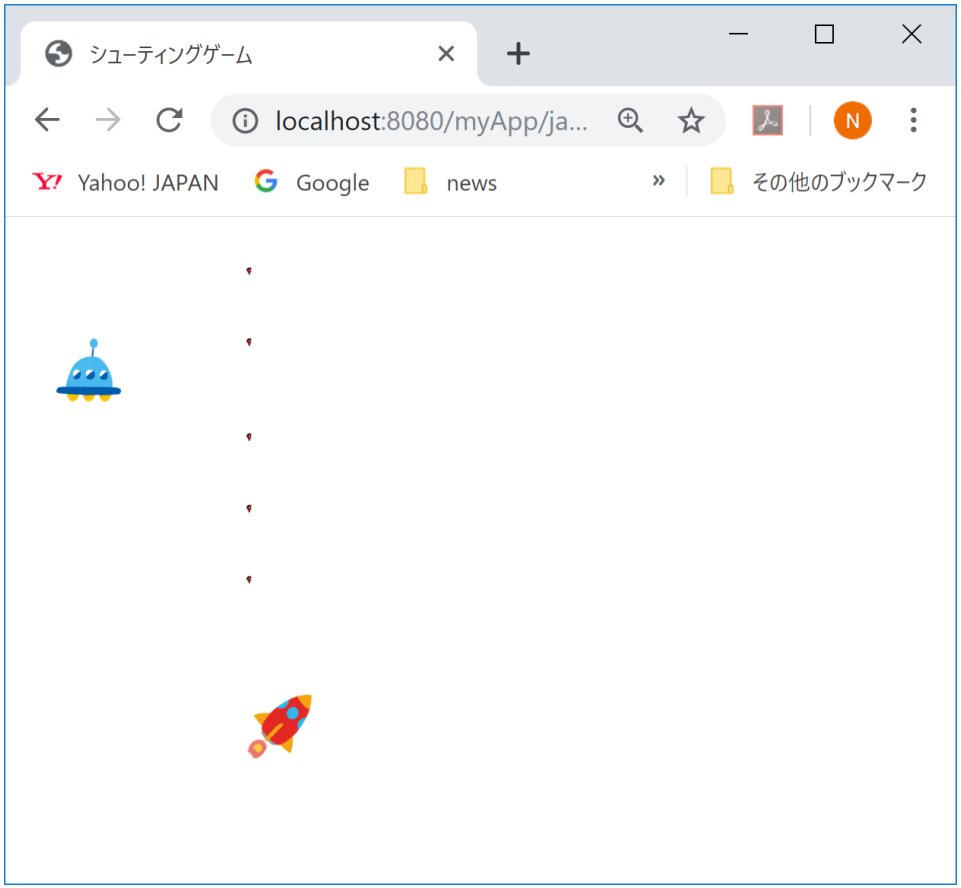
■次のファイルをネットワークドライブからインポートしてアクセスしてください。

ch_g2_eyesight.html

(G3)シューティングゲーム

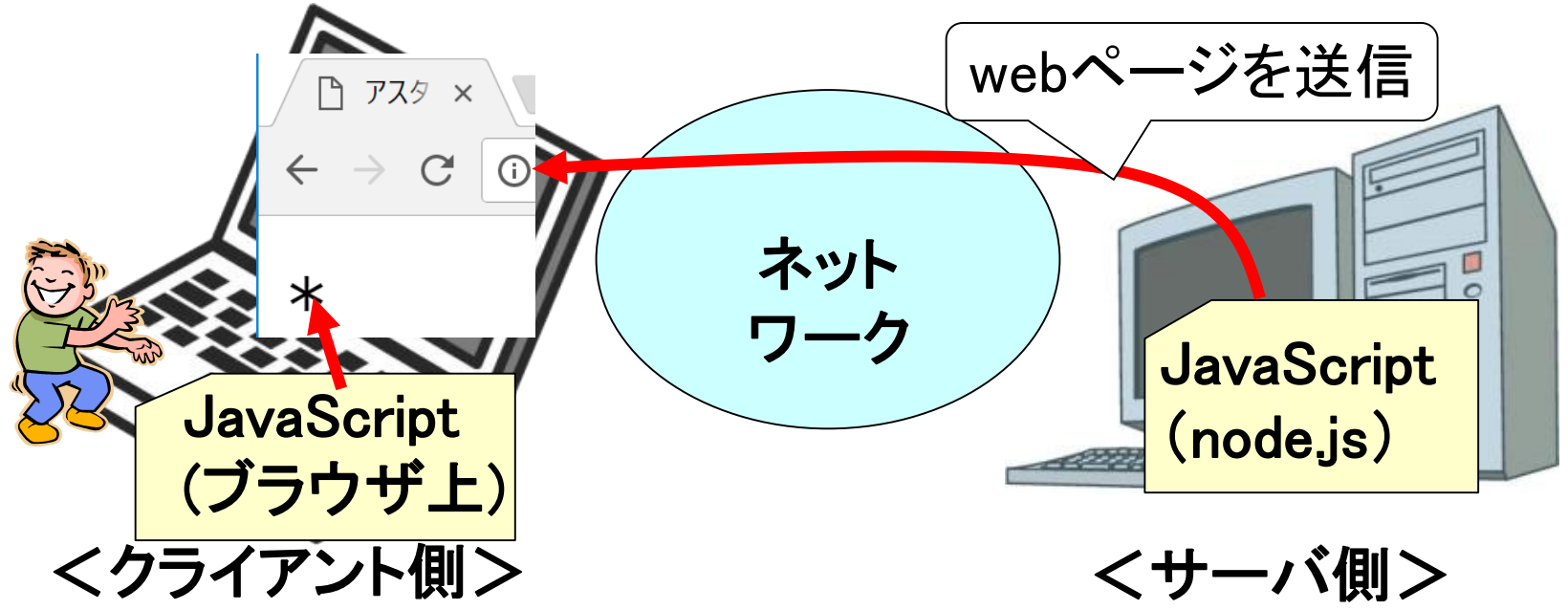
■次のファイルをネットワークドライブからインポートしてアクセスしてください。

ch_g2_eyesight.html



（N1）node.jsとファイル操作

■ブラウザ上で動作するJavaScriptに対し、サーバ上で動作するJavaScriptのことを、“node.js”と言います。



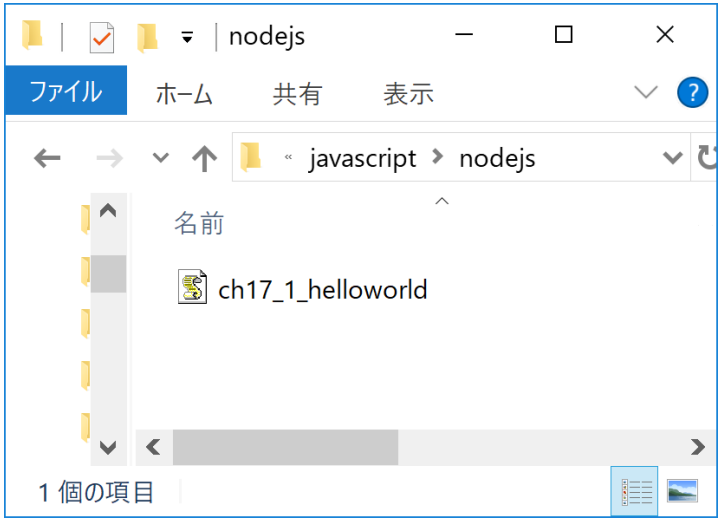
■node.jsは、必ずしもwebサーバ（次の(N2)項で説明）として動作するプログラムに使用目的が限られる訳ではありませんが、主にそのような形態で用いられています。

（N1．１）Hello World.

■大学の第2演習室のPCにはnode.jsがインストールされており、次のように実行できます。

＜コマンドプロンプト上＞

```
> node
ch_n1_1_helloworld.js
hello,world.
>
```



＜ch_n1_1_helloworld.js＞

```
console.log("hello,world.");
```

（N1．2）ファイルの読み出し

■サーバ側で動作するので、node.jsではファイルの読み書きが可能です。

＜コマンドプロンプト上＞

```
> node
ch_n1_2_readfile.js
白鳥は哀しからずや空の青
海のあをにも染まずただよふ
>
```

＜ex1.txt＞

```
白鳥は哀しからずや空の青
海のあをにも染まずただよふ
```

＜ch_n1_2_readfile.js＞

```
var fs=require('fs');

var lines=fs.readFileSync("./ex1.txt", 'utf8')

console.log(lines);
```

ファイルシステムのモジュール“fs”を読み込む

“fs”を使って“ex1.txt”のファイルを読み込む

出力する

（N1．３）ノンブロッキング、非同期

- node.jsでは“イベントループ”という考え方でwebサーバを実現しており、その処理には“非同期関数”を用いて“ノンブロッキング”なプログラムを用います（詳細省略）。
- しかしながら、本スライド（１７）では簡略化のために、“同期関数”を用いて“ブロッキング”なプログラムを示しています。
- 上記（１７．２）の“**`fs.readFileSync`**”も同期(Sync)関数です。
  以下のスライドで紹介するものも、同様です。

（N1．4）ファイルの書き込み（追記）

■ファイルへの書き込み（追記）は、次のように行うことが出来ます。

＜コマンドプロンプト上＞

```
> node ch_n1_4_appendfile.js
> type ex2.txt
白鳥は哀しからずや空の青
海のあをにも染まずただよふ
若山牧水
>
```

追記された

“type”はファイル内容を打ち出すコマンドですが、通常は文字化けします。
＜参考＞
http://www.koikikukan.com/archives/2013/06/27-025555.php

＜ch_n1_4_appendfile.js＞

```
var fs=require('fs');

fs.appendFileSync("./ex1.txt","若山牧水", 'utf8');
```

ファイルシステムのモジュール“fs”を読み込む

“fs”を使って“ex1.txt”のに“若山牧水”を追記する

（N1．5）ファイルの各行ごとの処理

■ここでは、次に示す簡易な方法を記すに留めます。

＜コマンドプロンプト上＞

```
> node ch_n1_5_eachline.js
1:白鳥は哀しからずや空の青
2:海のあをにも染まずただよふ
3:若山牧水
>
```

行番号と“:”を表示

①文字列に直して、

白鳥は哀しからずや空の青
海のあをにも染まずただよふ
若山牧水

to_string()

白鳥は哀しからずや空の青¥n海のあをにも染まずただよふ¥n若山牧水

split('¥n')

②改行”¥n”で区切る

lines[0] 白鳥は哀しからずや空の青
lines[1] 海のあをにも染まずただよふ
lines[2] 若山牧水

＜ch_n1_5_eachline.js＞

```
var fs=require('fs');
var lines=fs.readFileSync("./ex1.txt", 'utf8')
        .toString().split('¥n');
for(var i=0;i<lines.length;i++){
    console.log((i+1)+":"+lines[i]);
}
```

前の行に続けて書いて良い

行番号

“:”

各行の文字列

（N1．6）ファイルの書き込み（全体）

■ファイルへの書き込み（全体）は、次のように行うことが出来ます。

＜コマンドプロンプト上＞

```
> node ch_n1_6_writefile.js
> type ex2.txt
白玉の歯にしみとほる秋の夜の
酒は静かに飲むべかりけり
>
```

書き込まれた

“type”はファイル内容を打ち出すコマンドですが、通常は文字化けします。
＜参考＞
http://www.koikikukan.com/archives/2013/06/27-025555.php

＜ch_n1_6_writefile.js＞

```
var fs=require('fs');
fs.writeFileSync("./ex2.txt","白玉の歯にしみとほる
秋の夜の¥n酒は静かに飲むべかりけり", 'utf8');
```

ファイルシステムのモジュール“fs”を読み込む

改行しない

“fs”を使って“ex2.txt”に書き込む

（N2．１）最も単純なwebサーバ

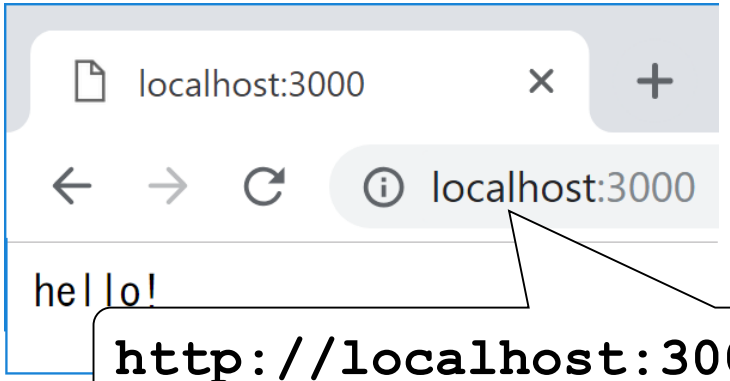
■もっとも単純なwebサーバを動かしてみます。

＜コマンドプロンプト上＞

```
> node ch_n2_1_webserver.js
```

ずっと実行中になります。
終了するには、[Ctrl]+[c]

＜webブラウザ＞



＜ch_n2_1_webserver.js＞

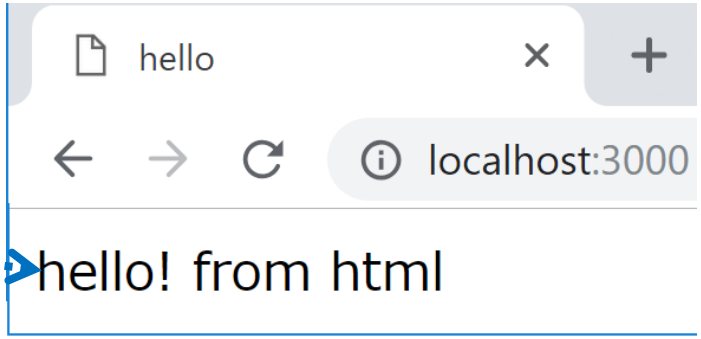
```
var http = require('http');          // httpパッケージを読み込み

var server = http.createServer();    // webサーバーを生成
server.on('request', getRequest);    // リクエストが来たら
                                     // getRequestに処理させる
server.listen(3000);                 // ポート3000番で待つ

function getRequest(req,res){        // req:リクエスト、res:レスポンス
    res.write('hello!');             // レスポンスに"hello!"と書き込む
    res.end();                       // レスポンスへの処理を終了する
}
```

（N2．2）ファイルを用いた表示

■“hello!”のみを書き込む乱暴な方法でなく、ちゃんとhtml文書を”**res.write(…);**”で書き込むのは大変です。ファイルから読み取って書き込むことにします。

<hello.ejs>

```
<html>
    <head><title>hello</title></head>
    <body>
        <p>hello! from html.</p>
    </body>
</html>
```



<ch_n2_2_webfile.js>

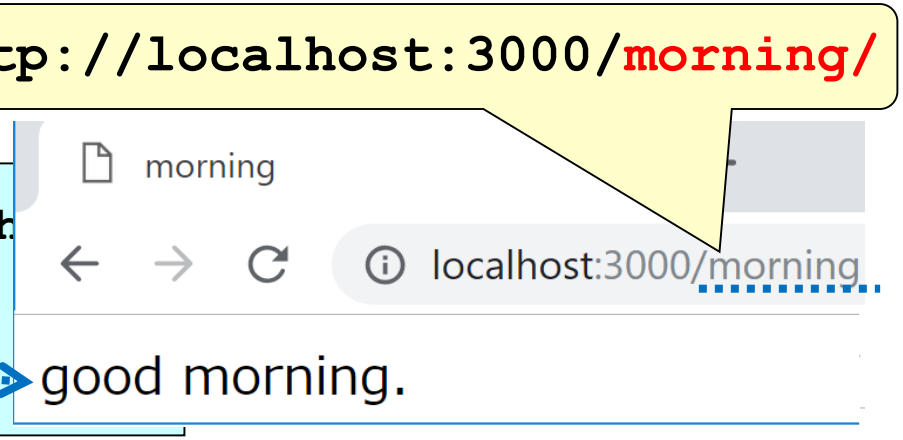
```
var http = require('http');
var fs     = require('fs');                    // ファイルシステムの読み込み
var server = http.createServer();
server.on('request', getRequest);
server.listen(3000);
var data = fs.readFileSync('./hello.ejs', 'utf-8');
                                               // html文書を読み取っておく。
function getRequest(req,res){
    res.write(data);      // 読み込んだhtml文書をレスポンスに書き込む
    res.end();
}
```

（N2．3）urlによる呼び分け

■urlの“hello”と“morning” により、表示するページを呼び分けることにします。

<morning.ejs>

```
<html>
 <head><title>morning</title></h
 <body>
  <p>good morning.</p>
 </body>
</html>
```



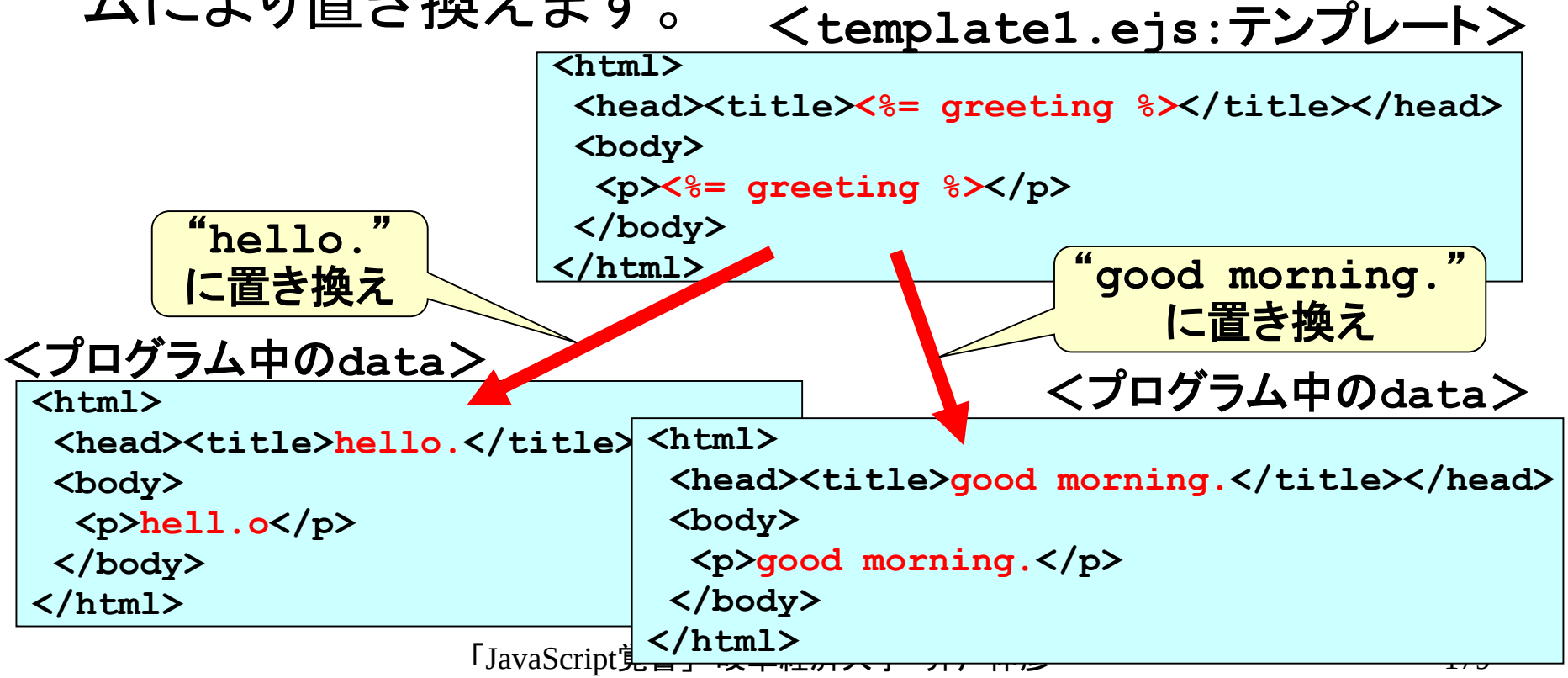
<ch_n2_3_weburl.js>

```
（前略(ch18_2_webfile.jsの5行目までと同じ））
var data = fs.readFileSync('./hello.ejs', 'utf-8'); // ひとつめの文書
var data2 = fs.readFileSync('./morning.ejs', 'utf-8'); // ふっつめの文書
function getRequest(req,res){
    if(req.url==`/hello`){          // urlが"/hello"であれば、
        res.write(data);            // ひとつめのhtml文書を書き込む
    }else if(req.url==`/morning`){  // urlが"/morning"であれば、
        res.write(data2);           // ふたつめのhtml文書を書き込む
    }
    res.end();                      // レスポンスへの処理を終了する
}
```

（N2．４．１）テンプレートの利用―１―

■テンプレートとは、変数を埋め込んだhtml文書です。変数は、”<%=　変数名　%>”の形で埋め込みます。

■ここで使用する下記のテンプレートには、“greeting”という名の変数が埋め込まれています。これをプログラムにより置き換えます。



（N2．４．２）テンプレートの利用―2―

■テンプレートの変数を置き換える際には、ejsパッケージのrender関数を使用します。

＜ch_n2_4_webtemplate.js＞

```
var http = require('http');
var fs   = require('fs');
var ejs  = require('ejs');          // ejsパッケージの読み込み

var server = http.createServer();
server.on('request', getRequest);
server.listen(3000);

var template = fs.readFileSync('./template1.ejs', 'utf-8');
                // テンプレートを読み取っておく。
function getRequest(req,res
    var greetingWords;                  // 挨拶を決める変数
    if(req.url=='/hello'){              // urlが"/hello"であれば、
        greetingWords="hello.";            // 挨拶は"hello."にする
    }else if(req.url=='/morning'){      // urlが"/hello"であれば、
        greetingWords="good morning.";  // 挨拶は"good morning."にする
    }
    var data = ejs.render(template, {greeting: greetingWords});
                // テンプレート中の変数"greeting"にgreetingWordの値を設定する。
    res.write(data);                    // レスポンスにdataを書き込む
    res.end();                          // レスポンスへの処理を終了する
}
```

（N2.5.1）テンプレート中のプログラム-1-

■テンプレートには、”<% プログラム %>”の形で、プログラムを埋め込むことも可能です。

■プログラムの繰り返しや分岐により、表示を変化させることが出来ます。

＜template2.ejs＞

```
<html>
    <head><title><%= greeting %></title></head>
    <body>
        <% for(var i=3;i<6;i++){ %>
        <p><font size="<%= i %>"><%= greeting %></font></p>
        <% } %>
    </body>
</html>
```

＜ch_n2_5_webtemplateP.js＞

```
（前略）
var template = fs.readFileSync('./template2.ejs', 'utf-8');
                                        // テンプレートのみ変更
（後略）
```

（N2.5.2）テンプレート中のプログラム−2−

■テンプレート中のfor文により表示が繰り返され、その大きさも変化しています。

＜template2.ejs＞

```
<body>
  <% for(var i=3;i<6;i++){ %>
    <p><font size="<%= i %>">
    <%= greeting %></font></p>
  <% } %>
 </body>
```

ejs.render

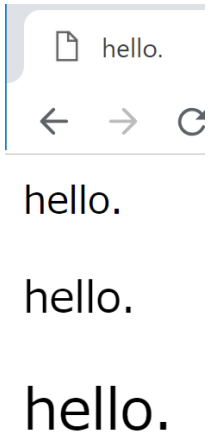
＜変数dataの内容＞

```
<body>
 <p><font size="3">hello.</font></p>
 <p><font size="4">hello.</font></p>
 <p><font size="5">hello.</font></p>
</body>
```

“i”の値に置き換え

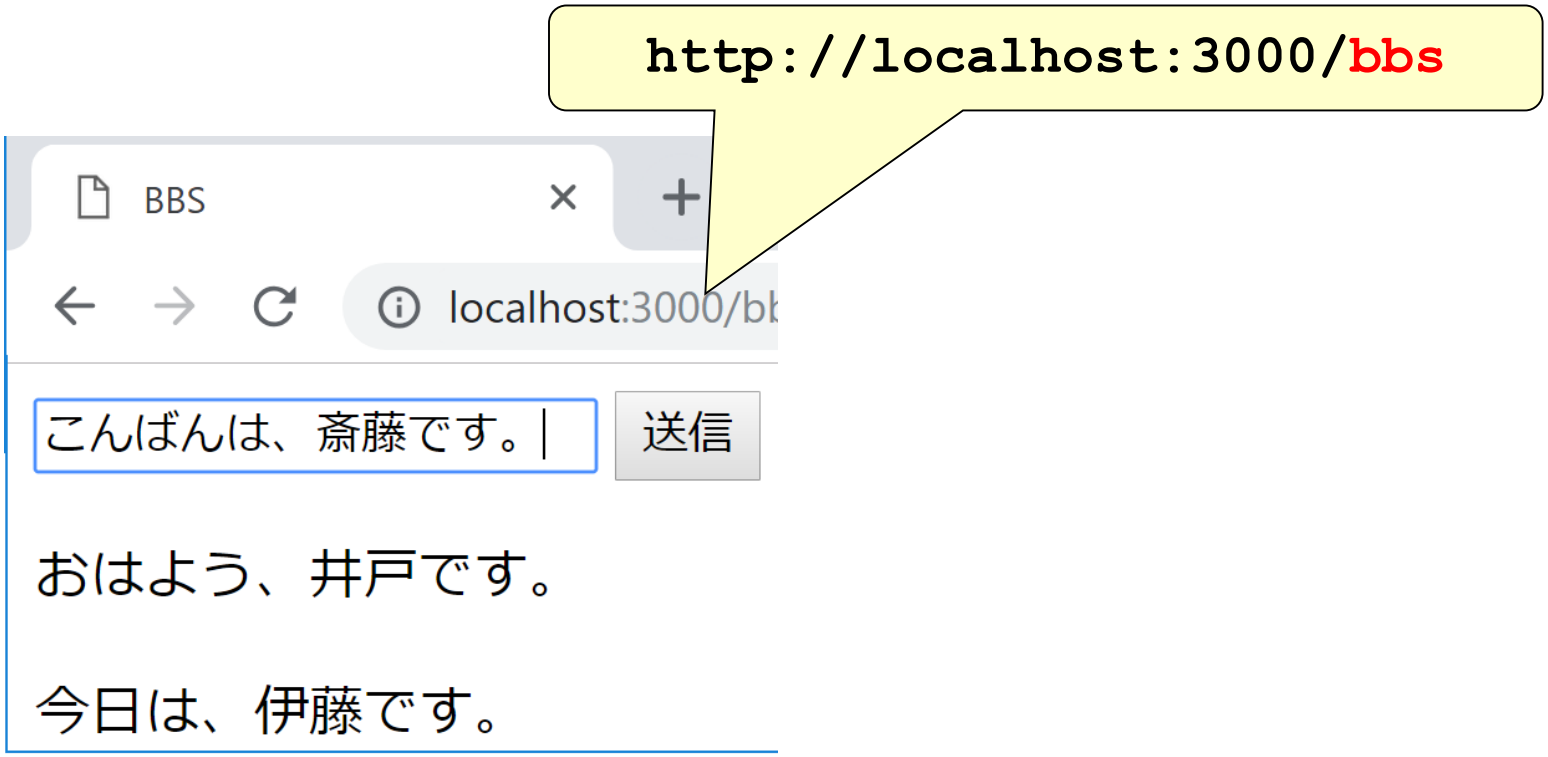
```
<body>
 <p><font size="3">hello.</font></p>
 <p><font size="4">hello.</font></p>
 <p><font size="5">hello.</font></p>
</body>
```

ブラウザでの表示



（N2．6）簡易BBS

■フォームを使って送信した内容が、その下に順次表示されるような簡易BBSを考えます。



（N2．6．1）簡易BBSのテンプレート

■フォームについては付録を参照してください。

■表示部分では、**for**文により配列“**messages**”の内容を順次表示するようになっています。

＜templateBBS.ejs＞

```
<html>
    <head>
        <meta charset="UTF-8">
        <title>BBS</title>
    </head>
    <body>
        <form action="/bbs" method="get">
            <input type="text" name="message"/>
            <input type="submit" value="送信"/>
        </form>
        <% for(var i=0;i<messages.length;i++){ %>
        <p><%= messages[i] %></p>
        <% } %>
    </body>
</html>
```

日本語表示の
文字化けを防ぐため、
文字コードを指定します。

＜フォーム＞

＜表示部分＞

（N2．6．2）簡易BBSのプログラム

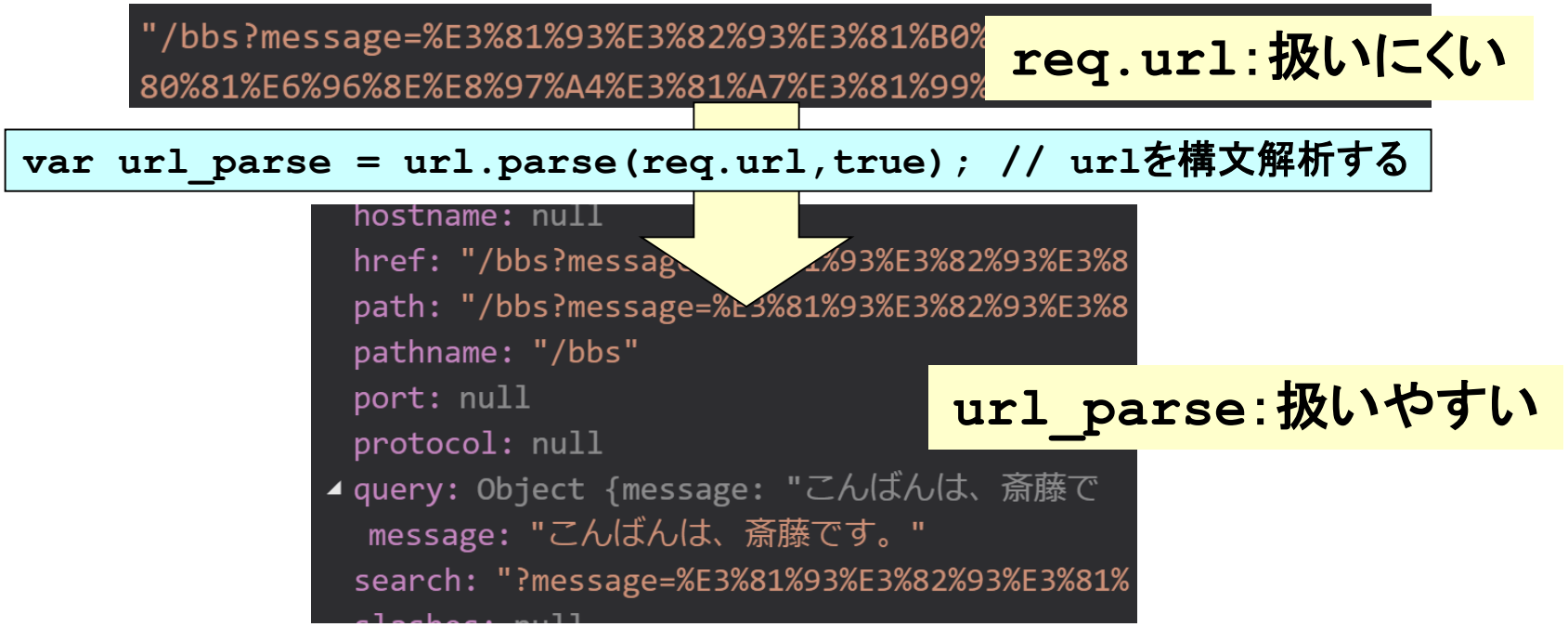
＜ch_n2_6_webBBS.js＞

```
var http = require('http');
var fs   = require('fs');
var ejs  = require('ejs');
var url = require('url');                    // urlの構文解析パッケージ
var server = http.createServer();
server.on('request', getRequest);
server.listen(3000);
var template = fs.readFileSync('./templateBBS.ejs', 'utf-8');
var messages=[];                             // メッセージを蓄積する配列を宣言する
function getRequest(req,res){
    var url_parse = url.parse(req.url,true); // urlを構文解析する
    if(url_parse.pathname=='/bbs'){          // urlが"/bbs"であれば、
        var message=url_parse.query.message; //"message"のフィールド読取
        if(message!=null){                   // 空でなければ、
            messages.push(message);            // 配列に追加する。
        }
        var data = ejs.render(template, {messages: messages});
                        // テンプレート中の変数"messages"にmessagesを設定する
        res.write(data);
    }
    res.end();
}
```

（N2．6．3）urlの構文解析

■送信された日本語のメッセージは、url内（“**req.url**”）に蓄積されています。このため、urlはエンコードされた扱いにくい文字列です。

■これを、urlパッケージの構文解析（“parse”関数）を使いことにより、扱いやすいデータに変換します。



（N2．6．4）urlの参照

■扱いやすくした“**url_parse**”を参照して、送信されたメッセージを、配列“**messages**”に設定しています。

```
if(url_parse.pathname=='/bbs'){                  // urlが"/bbs"であれば、
    var message=url_parse.query.message;         //"message"のフィールド読取
    if(message!=null){                           // 空でなければ、
        messages.push(message);                  // 配列に追加する。
    }
}
```

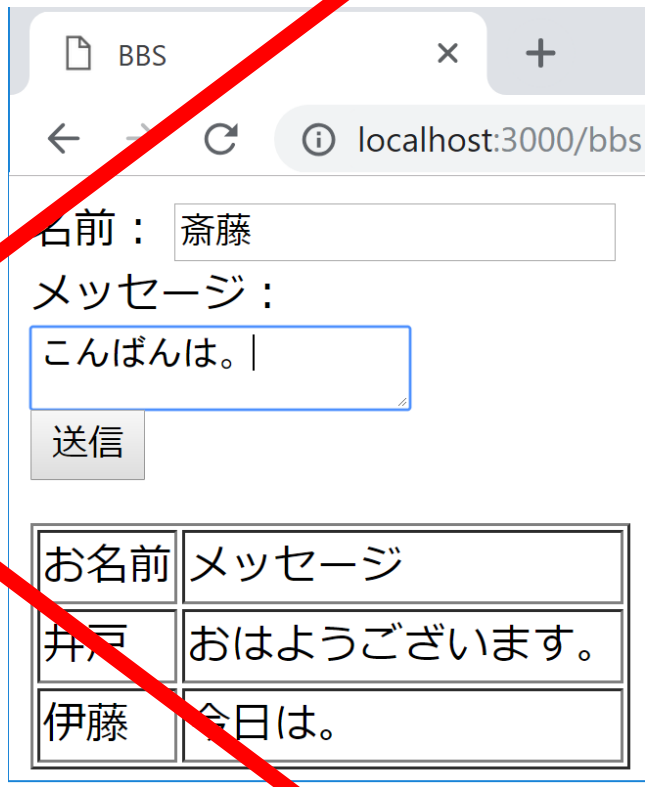
```
hostname: null
href: "/bbs?message=%E3%81%93%E3%82%93%E3%8
path: "/bbs?message=%E3%81%93%E3%82%93%E3%8
pathname: "/bbs"
port: null
protocol: null
query: Object {message: "こんばんは、斎藤で
 message: "こんばんは、斎藤です。"
search: "?message=%E3%81%93%E3%82%93%E3%81%
```

url_parse:扱いやすい

（N2．7）ファイルに保存するBBS

■名前とメッセージを区別して投稿し、投稿をファイルに保存するようにします。

- なお、ファイル“**posted.txt**”は予め作成されている（空でも良い）ことを前提としており、このファイルが無いとプログラムはエラーとなります。



＜posted.txt：保存するファイルの形式＞

```
{"user_name":"井戸","message":"おはようございます。"}
{"user_name":"伊藤","message":"今日は。"}
```

（N2．7．1）テンプレート

■表示部分はテーブルになっています。

＜templateBBSF.ejs＞

```
<html>
    <head>
        <meta charset="UTF-8">
        <title>BBS</title>
    </head>
    <body>
        <form action="/bbs" method="get">
            名前: <input type="text" name="user_name" /><br />
            メッセージ: <br />
            <textarea name="message"></textarea><br />
            <input type="submit" value="送信"/>
        </form>
        <table border='1'>
            <tr><td>お名前</td><td>メッセージ</td></tr>
            <% for(var i=0;i<posts.length;i++){ %>
            <tr><td><%= posts[i].user_name %></td>
                <td><%= posts[i].message %></td></tr>
            <% } %>
        </table>
    </body>
</html>
```

＜フォーム＞

(N2.7.4)参照

＜表示部分＞

(N2．7．2)ファイル保存BBSのプログラム

<ch_n2_7_webBBSF.js>

```
(前略:ch_n2_6_webBBS.jsと同じ)
var template = fs.readFileSync('./templateBBSF.ejs', 'utf-8');
function getRequest(req,res){
    var url_parse = url.parse(req.url,true);
    var file_path='./posted.txt';                    // 保存するファイル名
    if(url_parse.pathname=='/bbs'){
        var message=url_parse.query.message
        if(message!=null){ // ファイルへ保存
            var post='{"user_name":"'+url_parse.query.user_name+'"';
            post+=',"message":"'+message+'"}¥n';
            fs.appendFileSync(file_path,post,'utf-8');
        }
        var posts=[];
        var lines=fs.readFileSync(file_path,'utf8').toString().split('¥n');
        for(var i=0;i<lines.length;i++){
            var line=lines[i];
            if(line=="")continue;
            var line_p=JSON.parse(line);
            posts.push(line_p);
        }
        var data = ejs.render(template, {posts: posts});
                        // テンプレート中の変数"posts"にpostsを設定する。
        res.write(data);
    }
    res.end();
}
```

保存

読み出し・表示

(N1.4)参照

(N1.5)参照

(N2.7.4)参照

（N2．7．3）JSON形式

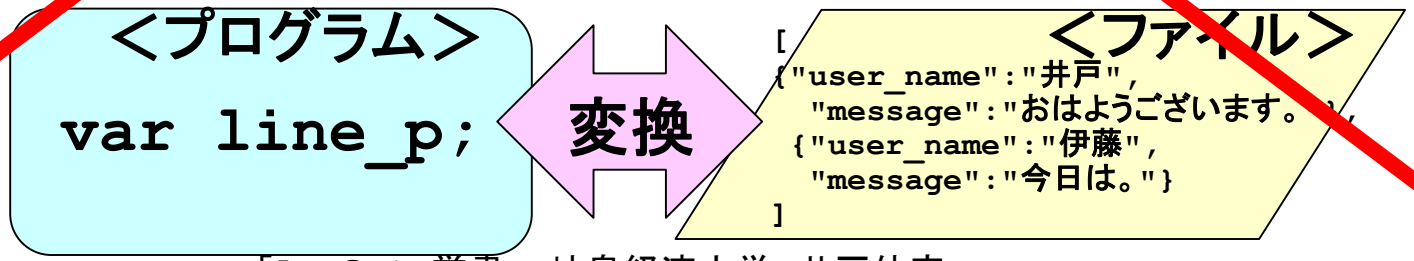
■投稿を保存するファイル“posted.txt”は、各行がJSONという形式になっています。

```
{"user_name":"井戸","message":"おはようございます。"}
{"user_name":"伊藤","message":"今日は。"}
```

本当はファイル全体で次のようなJSON形式にすべきですが、
ファイル操作の行数と必要な知識とが増えないように、
上記のような変則的な形にしています。

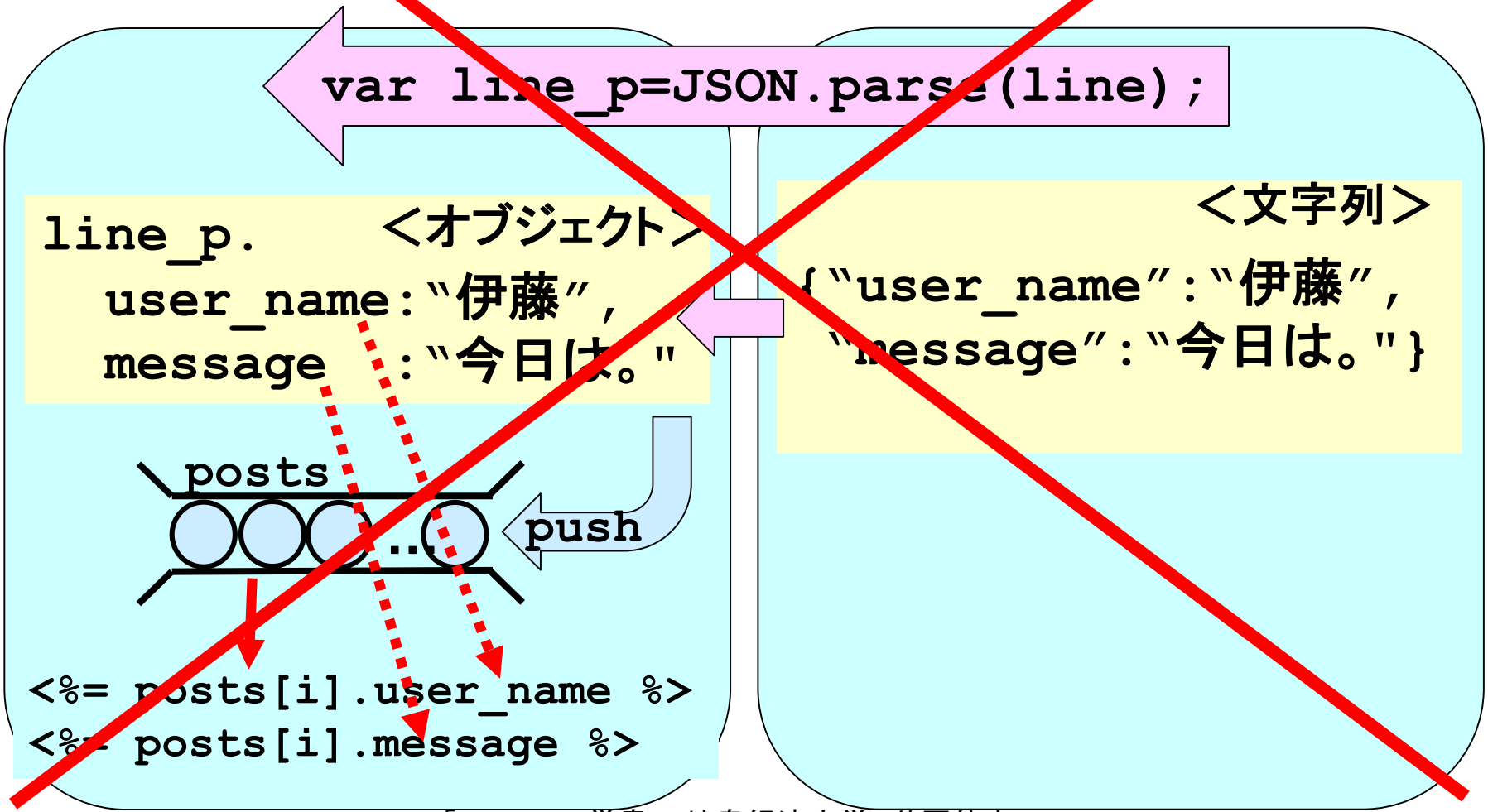
```
[{"user_name":"井戸","message":"おはようございます。"},
 {"user_name":"伊藤","message":"今日は。"}]
```

■JSON形式は、プログラム中のデータ(配列とオブジェクト(=連想配列))と相互に変換できるファイル形式です。



（N2．7．4）JSON形式の構文解析

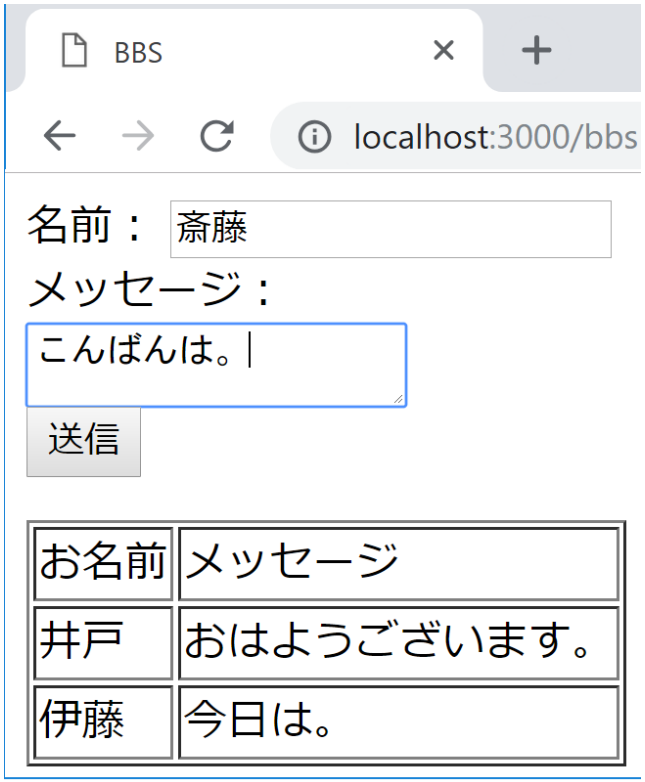
■前のスライドに示した“変換”を、“JSON.parse()”で行っています。



（N2．8）ファイルに保存するBBS（修正版）

■名前とメッセージを区別して投稿し、投稿をファイルに保存するようにします。

- ファイル“**posts.txt**”は予め作成されている（空でも良い）ことを前提としており、このファイルが無いとプログラムはエラーとなります。



＜posts.txt：保存するファイルの形式＞

```
[{"user_name":"井戸","message":"おはようございます。"},
 {"user_name":"伊藤","message":"今日は。"}]
```

(N2．8．1)テンプレート

■表示部分はテーブルになっています。

<templateBBSF.ejs>

```
<html>
    <head>
        <meta charset="UTF-8">
        <title>BBS</title>
    </head>
    <body>
        <form action="/bbs" method="get">
            名前: <input type="text" name="user_name" /><br />
            メッセージ: <br />
            <textarea name="message"></textarea><br />
            <input type="submit" value="送信"/>
        </form>
        <table border='1'>
            <tr><td>お名前</td><td>メッセージ</td></tr>
            <% for(var i=0;i<posts.length;i++){ %>
            <tr><td><%= posts[i].user_name %></td>
                <td><%= posts[i].message %></td></tr>
            <% } %>
        </table>
    </body>
</html>
```

＜フォーム＞

＜表示部分＞

(N2.8.4)参照

（N2．8．2）ファイル保存BBSのプログラム

＜ch_n2_8_webBBSF.js＞

```
（前略:ch_n2_6_webBBS.jsと同じ）
var template = fs.readFileSync('./templateBBSF.ejs', 'utf-8');
function getRequest(req,res){    // req:リクエスト、res:レスポンス
    var url_parse = url.parse(req.url,true);    // urlを解析する
    if(url_parse.pathname=='/bbs'){             // urlが"/bbs"であれば、
        var file_path='./posts.txt';
        var posts = JSON.parse(fs.readFileSync(file_path, 'utf8'));
                        // JSON形式のファイルを読み取る
        var message=url_parse.query.message;
                        // name="message"のフィールドを読み取る
        if(message!=null){
            posts.push({user_name:url_parse.query.user_name,
                        message:message});
            fs.writeFileSync(file_path,JSON.stringify(posts));
                // JSON形式のファイルを書き込む
        }
        var data = ejs.render(template, {posts: posts});
                        // テンプレート中の変数"messages"にmessagesを設定する。
        res.write(data);                // レスポンスにdataを書き込む
    }
    res.end();                          // レスポンスへの処理を終了する
}
```

読出

保存

表示

(N2.8.4)参照

(N1.6)(N2.8.4)参照

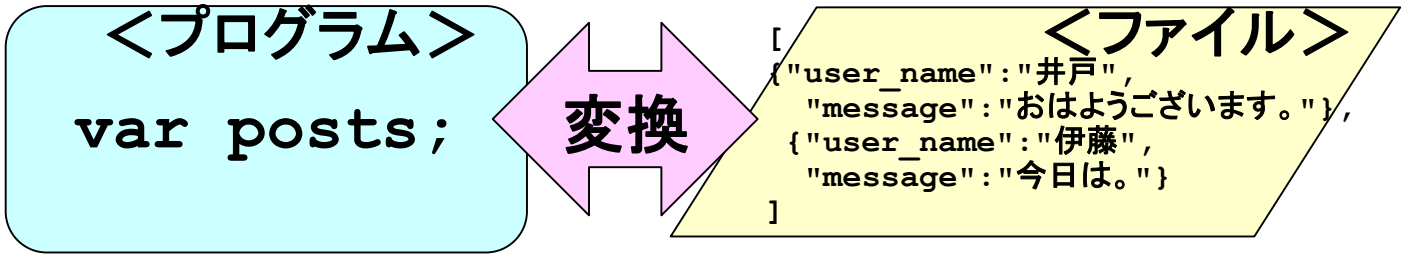
（N2．8．3）JSON形式

■投稿を保存するファイル“posts.txt”は、JSONという形式になっています。

```
[{"user_name":"井戸","message":"おはようございます。"},
 {"user_name":"伊藤","message":"今日は。"}]
```

（プログラムからファイルを保存すると、改行は失われます）

■JSON形式は、プログラム中のデータ（配列とオブジェクト(=連想配列)）と相互に変換できるファイル形式です。



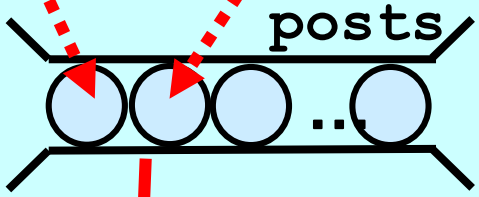
（N2．8．4）JSON形式の構文解析

■前のスライドに示した“変換”は、次のように行います。

```
var posts = JSON.parse(ファイル読み出し);
ファイル読み出し(JSON.stringify(posts));
```

```
{user_name:“井戸”,
 message   :“おはよう..“}
```

```
{user_name:“伊藤”,
 message   :“今日は。“}
```




```
<%= posts[i].user_name %>
<%= posts[i].message %>
```

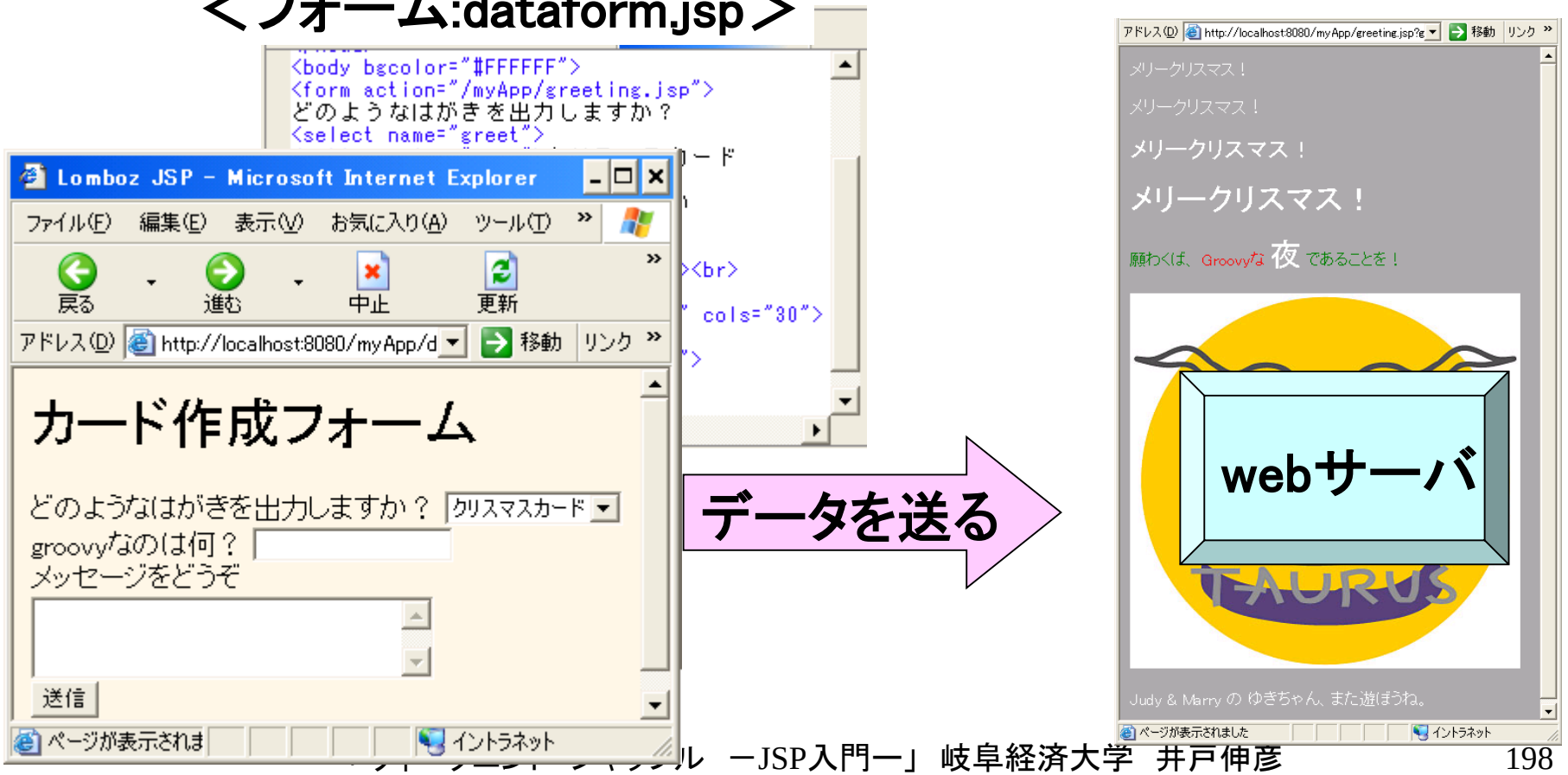
＜文字列＞

```
[{"user_name":"井戸",
  "message":"おはようございます。"},
 {"user_name":"伊藤",
  "message":"今日は。"}]
```

（付録）フォーム

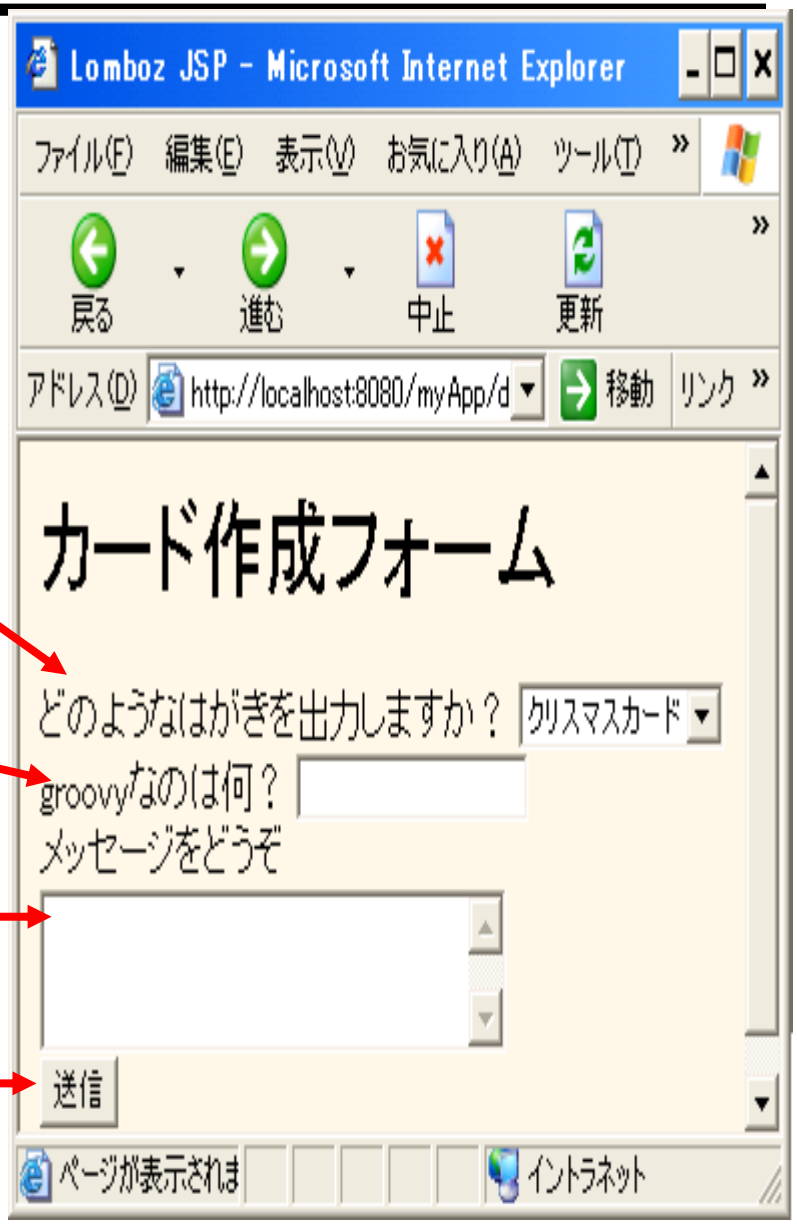
■ここでは、HTMLのフォームを用いて、webサーバにデータを送るための次のようなページ（dataform.jsp）を例に説明を行います。

＜フォーム:dataform.jsp＞



（付録．１）作成するフォーム（dataform.jsp）

```
<body bgcolor="##FDF5E6">
<p><font size="6">カード作成フォーム
  </font></p>
<form action="/greetingCard">
どのようなはがきを出力しますか？
<select name="greet">
<option value="xmas">クリスマスカード
<option value="newyear">年賀状
<option value="summer">書中見舞い
</select><br>
groovyなのは何？
<input type="text" name="groovy"><br>
メッセージをどうぞ<br>
<textarea name="message" rows="3"
  cols="30">
</textarea><br>
<input type="submit" value="送信">
</form>
</body>
```



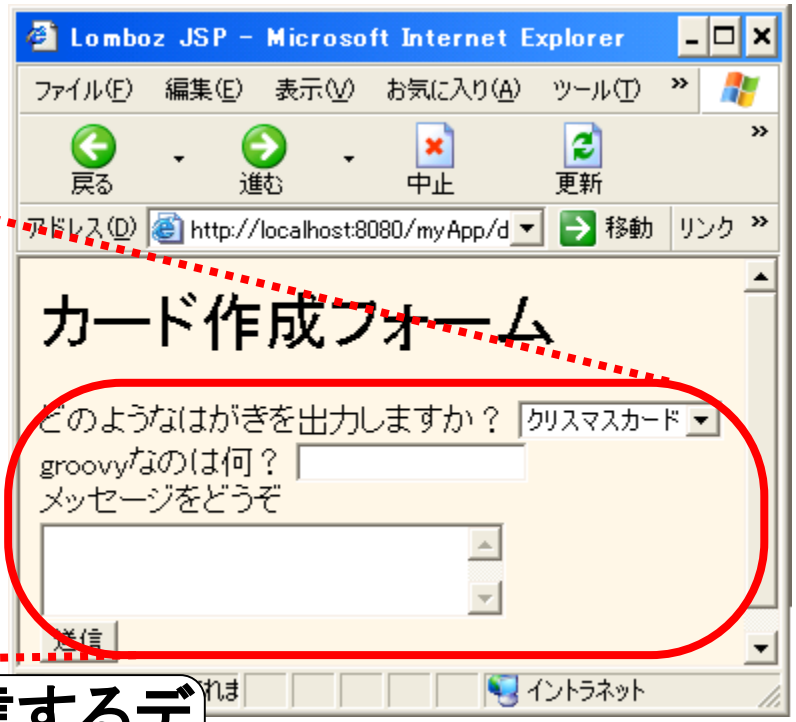
（付録．２）フォーム要素の構成

■フォームについても学習済みであることを前提としていますが、ここでは説明を繰り返します。

■タグ“<form>”と“</form>”に囲まれた部分で、送信するデータを指定する部分を作ります。

ここでは、
データを送る先のURL

```
<form action=“ 送信先”>
    HTML要素
    HTML要素
        :
    HTML要素
</form>
```



入力画面と、それにより送信するデータについて書く

（付録．３）テキスト・フィールド

■HTML要素：`<input type="text" name="…">`

■属性：name ⇒ フィールドに入力されたデータの名前

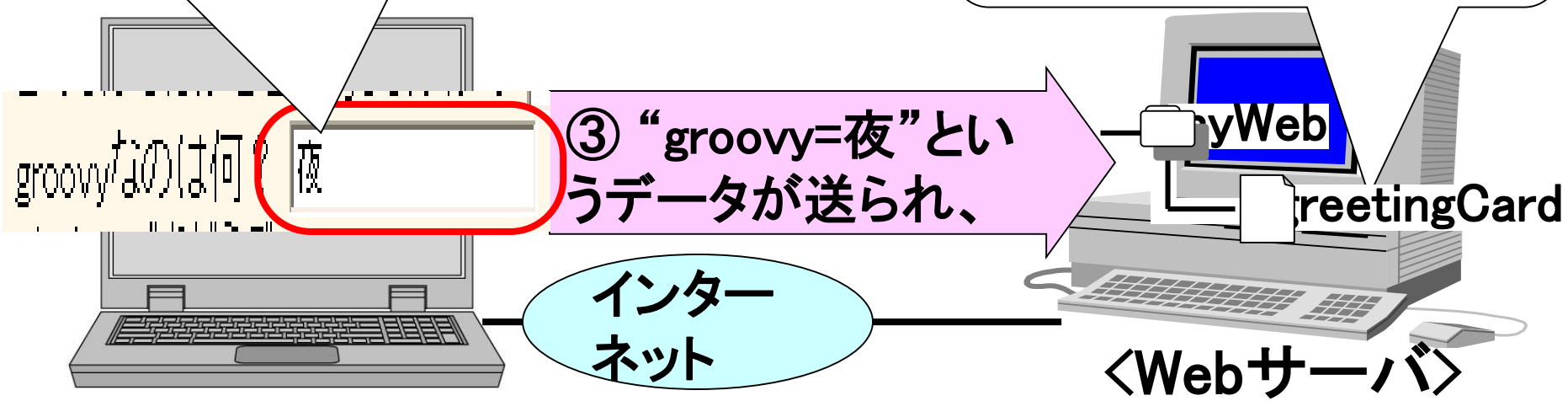
value ⇒ フィールドに予め入れておく文字列

```
<input type="text"
  name="groovy">
```

①このように書くと。。。

②テキストを入力する箱が作成され、（ここでは“夜”と入力したとする）

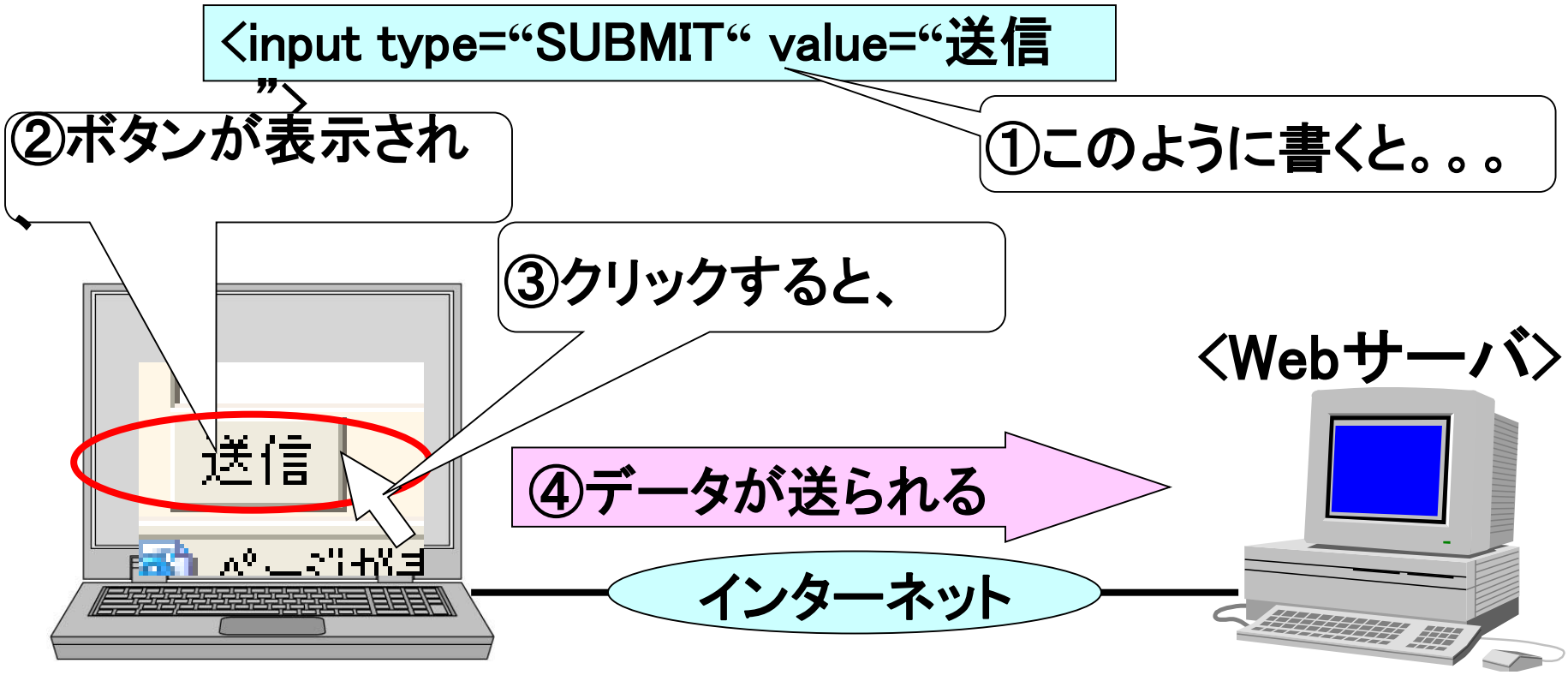
④プログラムでは、“groovy=夜”を利用して処理をする



（付録．４）送信ボタン

■HTML要素：`<input type="submit" value="…">`

■属性：value ⇒ ボタンの上のラベルとなる



（付録．５）テキスト・エリア

■テキスト・エリア：複数行のテキスト領域

■HTML要素：`<textarea name="…"></textarea>`

■属性：name ⇒エリアに入力されたデータの名前

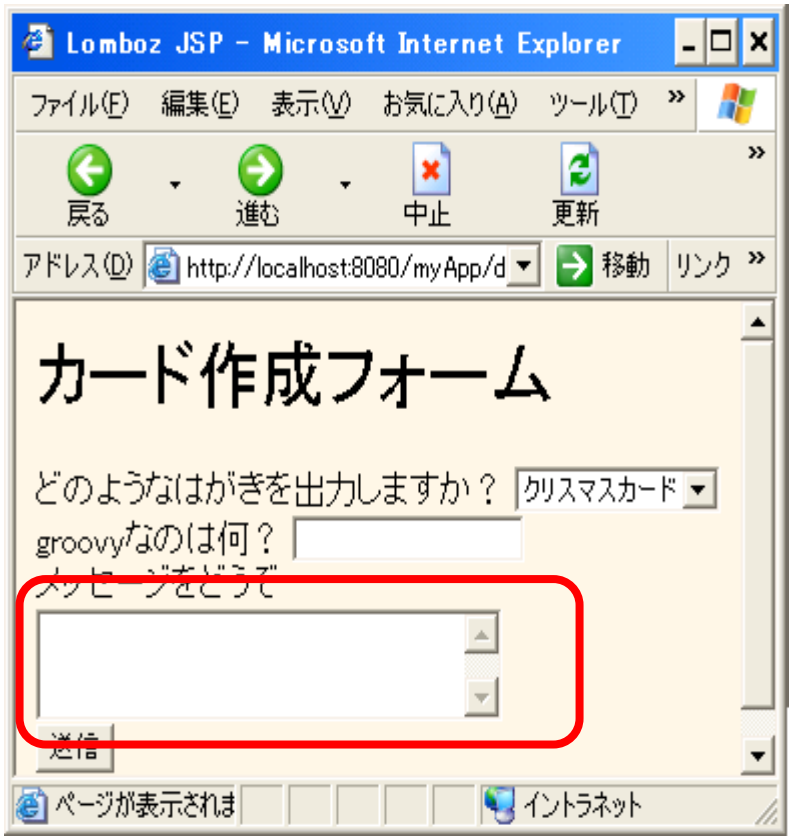
rows ⇒ 行の数

cols ⇒ 行の長さ

■動作は、テキストフィールドと同じです。

```
<textarea name="message"
  rows="3" cols="30">
</textarea><br>
```

“
”は改行です



（付録．６）コンボ・ボックス

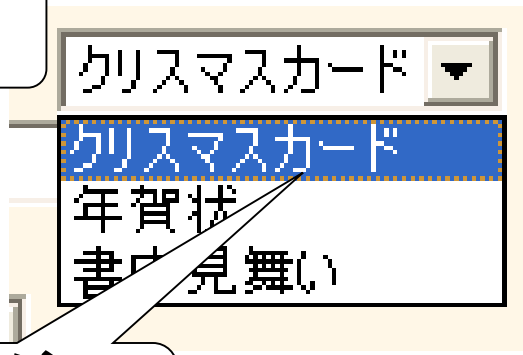
■コンボ・ボックス：選択項目から、ひとつをリストで選択

■HTML要素 `<select name="…"><option value="…">..</select>`

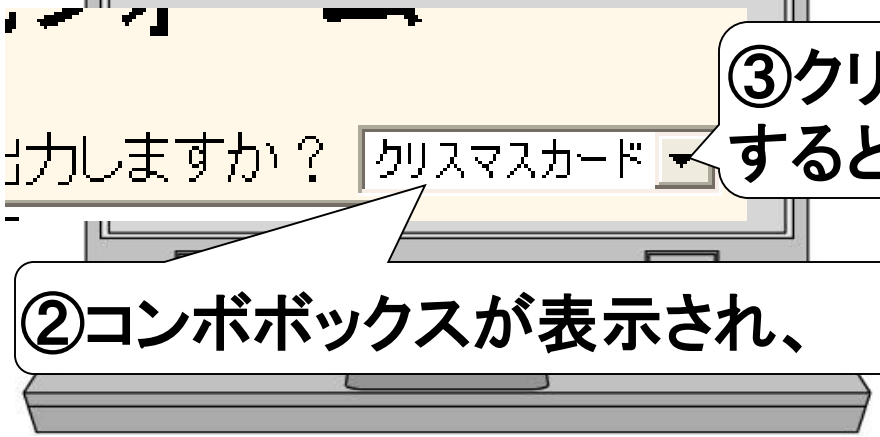
どのようなはがきを出力

```
<select name="greet">
<option value="xmas">クリスマスカード
<option value="newyear">年賀状
<option value="summer">書中見舞い
</select><br>
```

①このように書くと。。。



④選択が可能となり

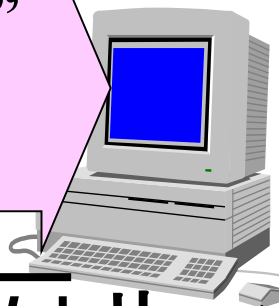


③クリックすると、

②コンボボックスが表示され、

⑤“greet=xmas”というデータが送られる

インターネット



〈Webサーバ〉