Operators and Statements

Bok, Jong Soon
javaexpert@nate.com
www.javaexpert.co.kr

The Format of a JavaScript Statement

- Usually end with a semicolon.
- But, a semicolon is not required.
- Can omit the semicolon

```
var bValue = true
var sValue = "this is also true“
```

- If multiple statements appear on the same line, Must use the semicolon to terminate each one

```
var bValue = true; var sValue =
"this is also true"
```

The Assignment Statement

- The most common JavaScript statement is the assignment statement.

```
nValue = 35.00;
nValue = nValue + 35.00;
nValue = someFunction();
var firstName = 'Shelley'; var lastName =
  'Powers';
var firstName = lastName = middleName = "";
var nValue1,nValue2 = 3;
    // nValue2 is undefined
var nValue1=3, nValue2=4, nValue3=5;
```

The Assignment Statement (Cont.)

- For readability and to ensure that no bugs creep in, separate your assignments with semicolons.

Object and Array Initializers

- Are expressions whose value is a newly created object or array.
- Are sometimes called *object literals* and *array literals*.
- An array initializer is a comma-separated list of expressions contained within square brackets(`[]`).

```
[]                  // An empty array
[1+2,3+4]           // A 2-element array
```

Object and Array Initializers (Cont.)

- An array initializer expression can create nested arrays.

```
var matrix = [[1,2,3], [4,5,6], [7,8,9]];
```

- Undefined elements can be included in an array literal by simply omitting a value between commas.

```
var sparseArray = [1,,,,5];
```

Lab1 : An Array Initializer Expressions

- Web Browsers
  - IE10, Firefox, Google Chrome, Opera, Safari
- Text Editors
  - Notepad++ or Editplus
- Files
  - arrayinitializer.html
  - table.css

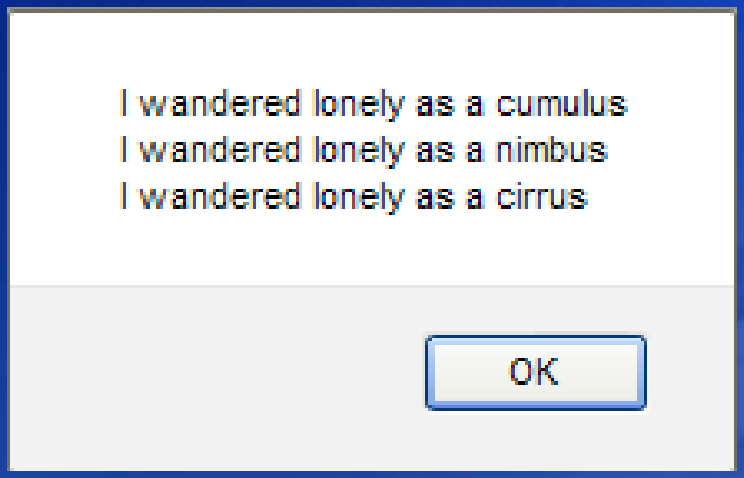
Lab1 : arrayinitializer.html

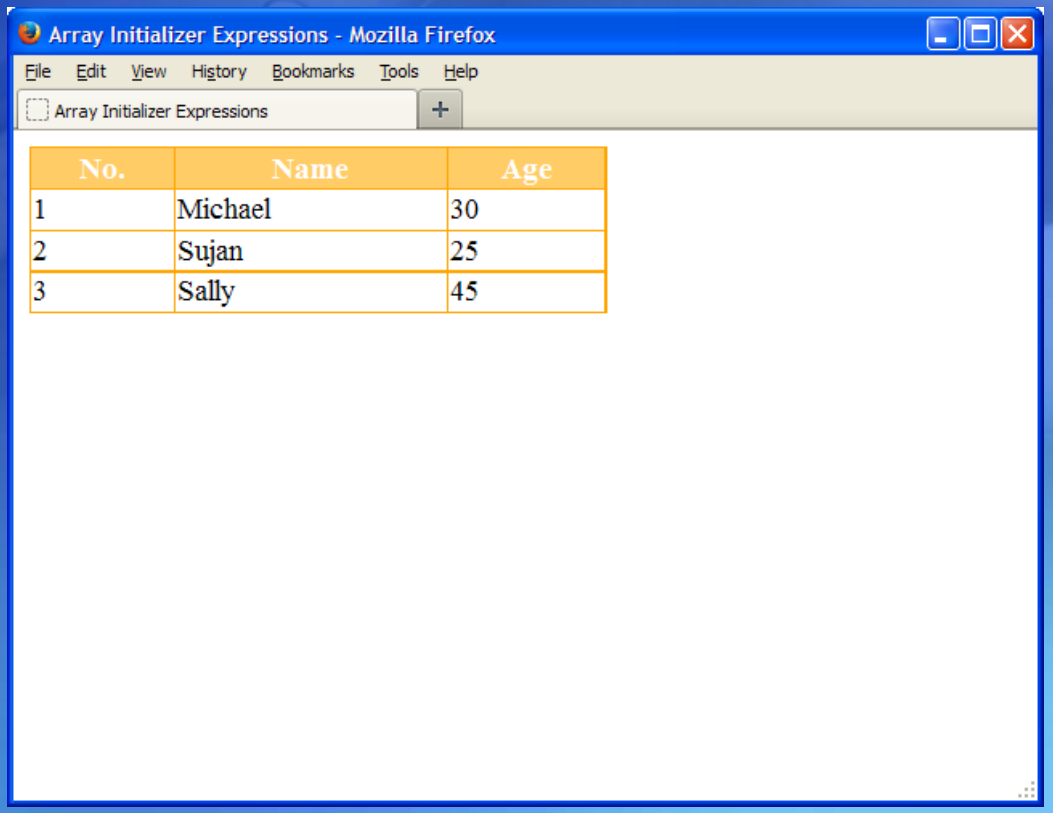
```
3   <head>
4     <title> Array Initializer Expressions </title>
5     <meta charset="utf-8">
6     <link rel="stylesheet" type="text/css" href="table.css">
7   </head>
8   <body>
9     <script>
10      var clouds = ["cumulus", "nimbus", "cirrus"];
11      var str = "";
12      for(var i = 0 ; i < clouds.length ; i++){
13        str += "I wandered lonely as a " + clouds[i] + "\n";
14      }
15      alert(str);
16
17      document.write("<table>");
18      document.write("<tr>");
19      document.write("<th>No.</th><th>Name</th><th>Age</th>");
20      document.write("</tr>");
21      var cards = [[1, "Michael", 30],[2, "Sujan", 25],[3, "Sally", 45]];
22      for(var i = 0 ; i < cards.length ; i++){
23        document.write("<tr>");
24        for(var j = 0 ; j < cards[i].length ; j++){
25          document.write("<td>" + cards[i][j] + "</td>");
26        }
27        document.write("</tr>");
28      }
29      document.write("</table>");
30    </script>
31  </body>
```

Lab1 : table.css

```
1  table {
2      border : solid  1px  orange ;
3      border-collapse : collapse;
4      width : 300px;
5  }
6  th {
7      border : solid 1px orange ;
8      background-color:#FFCC66;
9      color : white;
10 }
11 td {
12     border : solid 1px orange ;
13 }
```

Lab1 : Result


I wandered lonely as a cumulus
I wandered lonely as a nimbus
I wandered lonely as a cirrus
OK


Array Initializer Expressions - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Array Initializer Expressions
No. Name Age
1 Michael 30
2 Sujan 25
3 Sally 45

Object and Array Initializers (Cont.)

- Object initializer expressions are like array initializer expressions.
- *But*, the square brackets are replaced by curly brackets({}).
- And each subexpression is prefixed with a property name and a colon.

```
var p = { x:2.3, y:-1.2 }; // An object with 2
                           // properties
var q = {};      // An empty object with no properties
q.x = 2.3; q.y = -1.2;   // Now q has the same
                         // properties as p
```

Object and Array Initializers (Cont.)

- Object literals can be nested.

```
var rectangle = {
                  upperLeft: { x: 2, y: 2 },
                  lowerRight: { x: 4, y: 5 }
                };
```

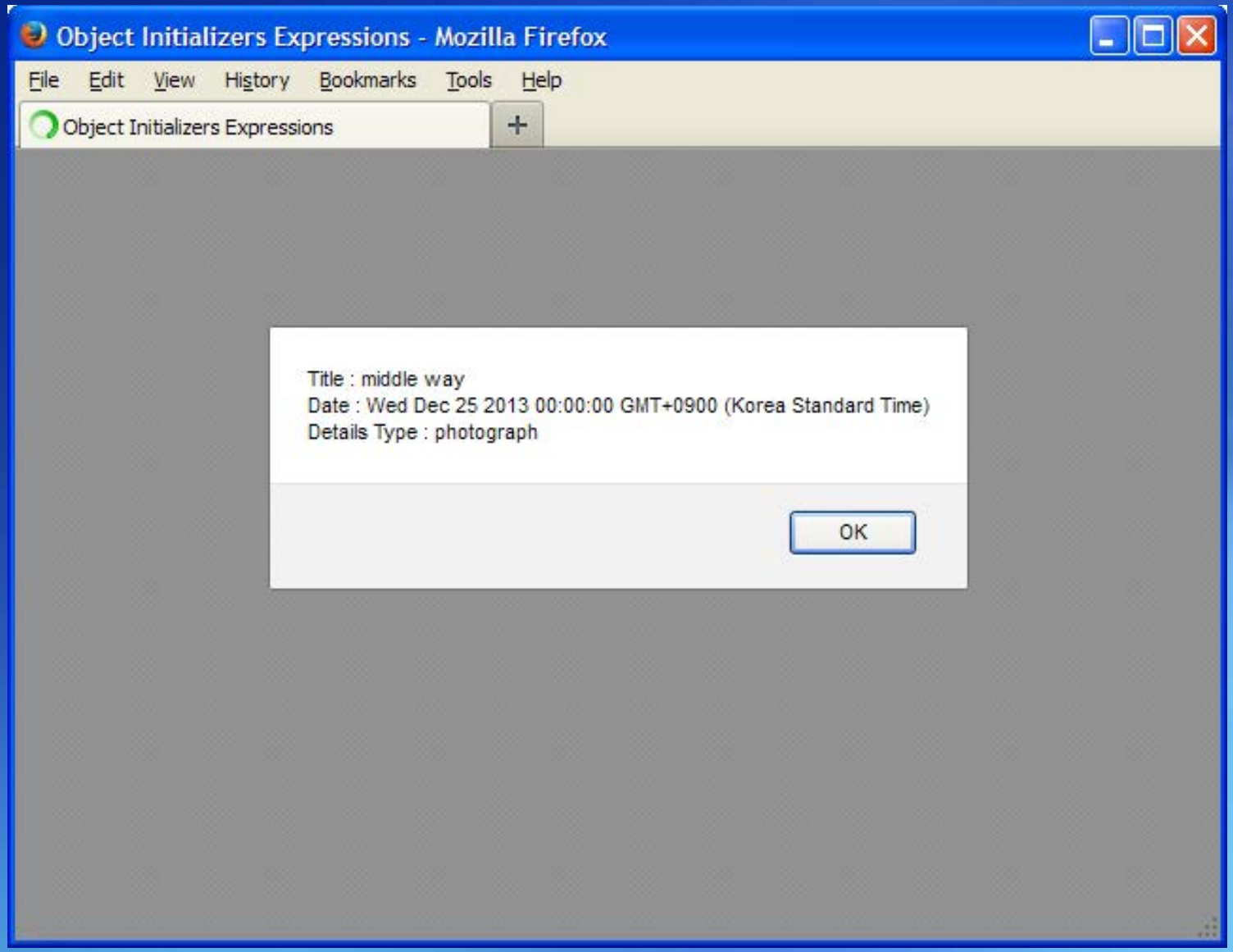
Lab2 : A Object Initializer Expressions

- Web Browsers
  - IE10, Firefox, Google Chrome, Opera, Safari
- Text Editors
  - Notepad++ or Editplus
- Files
  - objectinitializer.html

Lab2 : objectinitializer.html

```
3   <head>
4     <title> Object Initializers Expressions </title>
5     <meta charset="utf-8">
6   </head>
7   <body>
8     <script>
9       var Picture = { id : 3,
10                      title : "middle way",
11                      date : new Date("12/25/2013"),
12                      details : {
13                        type : "photograph",
14                        keywords : ["landscape", "tranquil", "green", "vegetation"]
15                      }
16                    }
17      var str = "Title : " + Picture.title + "\n";
18      str += "Date : " + Picture['date'] + "\n";
19      str += "Details Type : " + Picture['details']['type'];
20      alert(str);
21    </script>
22  </body>
```

Lab2 : Result


Object Initializers Expressions - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Object Initializers Expressions
Title : middle way
Date : Wed Dec 25 2013 00:00:00 GMT+0900 (Korea Standard Time)
Details Type : photograph
OK

Function Definition Expressions

- Defines a JavaScript function.
- The value of such an expression is the newly defined function.
- Is a *function literal*.
- Typically consists of the keyword **`function`** followed by a comma-separated list of zero or more identifiers (the **`parameter`** names) in parentheses and a block of JavaScript code (the **`function body`**) in curly braces.

```
var square = function(x) { return x * x; }
```

Evaluation Expressions

- JavaScript has the ability to interpret strings of JavaScript source code, evaluating them to produce a value.
- JavaScript does this with the global function `eval()`.

```
eval("3+2")      // => 5
```

Operator Precedence

Precedence	Operator Type	Individual Operators	Associativity
1	member	`. []`	L → R
	new	`new`	R → L
2	function call	`()`	L → R
3	incre / decrement	`++ --`	N/A
4	local NOT	`!`	R → L
	bitwise NOT	`~`	
	unary +	`+`	
	unary negation	`-`	
	typeof	`typeof`	
	void	`void`	
	delete	`delete`	
5	multiplication	`*`	L → R
	division	`/`	
	modulus	`%`	
6	addition	`+`	L → R
	subtraction	`-`	

Operator Precedence (Cont.)

Precedence	Operator Type	Individual Operators	Associativity
7	bitwise shift	`<<` `>>` `>>>`	L → R
8	relational	`<` `<=` `>` `>=`	L → R
	in	`in`	
	instanceof	`instanceof`	
9	equality	`==` `!=` `===` `!==`	L → R
10	bitwise AND	`&`	L → R
11	bitwise XOR	`^`	L → R
12	bitwise OR	`|`	L → R
13	logical AND	`&&`	L → R
14	logical OR	`||`	L → R
15	conditional	`?:`	R → L
16	yield	`yield`	R → L
17	assignment	`=` `+=` `-=` `*=` `/=` `%=` `<<=` `>>=` `>>>=` `&=` `^=` `|=`	R → L
18	comma	`,`	L → R

Member Operators

- Provide access to an object's properties and methods.
- An object is ***associative array***(map, dictionary, hash ...)
- .
- []

```
<script type="text/javascript">
  var car = { name : "Sonata",  price : 25000000,  maker : "Hyundai"};
  alert(car.price);
  alert(car["price"]);
  alert(car[price]);      //error "price is undefined"
</script>
```

Operators

- Performs a function on one, two, or three operands and returns a result.
- An operator that requires one operand is called a ***unary operator***.
- An operator that requires two operands is a ***binary operator***.
- A ***ternary operator*** is one that requires three operands.
- Contains precedence order(top to bottom) along with their ***associativity*** (left to right or right to left).

new operator

- Creates an instance of a user-defined object type or of one of the built-in object types that has a constructor function.
- Syntax

```
new constructor[([arguments])]
```

```
<script type="text/javascript">
   function car(make, model, year){
      this.make = make;
      this.model = model;
      this.year = year;
   }
   var sonata = new car("Hyundai", "EF Sonata", 2005);
   document.write(sonata.make + "<br />");
   document.write(sonata["make"]);
</script>
```

Unary Operators – Increment and Decrement Operators

- A more common requirement is to add or subtract 1 from a variable.
- Either ++ or – can appear before (*prefix*) or after (*postfix*) its operand.

Purpose	Operator	Example
Pre - Increment	++	a = ++b;
Post - Increment	++	a = b++;
Pre - Decrement	- -	a = - -b;
Post - Decrement	- -	a = b- -;

Unary Operators

Operator type	Operator	Example
Unary plus	+	+4
Unary negation	-	-4
Bitwise NOT	~	var su = ~5;
Logical NOT	!	var isTrurh = !true;

Unary Operators (Cont.)

```
<script type="text/javascript">
    var su = 5;
    document.write(su.toString(2) + "<br />");  //101
    var num = ~su;
    document.write(num.toString(2));          //-110 --> -6
</script>
```

```
<script type="text/javascript">
    var a = true;
    var b = !a;
    document.write("a = " + a + "<br />"); //a = true
    document.write("b = " + b);                //b = false
</script>
```

Unary Operators – typeof

- Returns a string indicating the type of the unevaluated operand.
- Syntax

```
typeof operand
typeof (operand)
```

```
<script type="text/javascript">
    document.write(typeof 37 + "<br />");    //'number'
    document.write(typeof 3.14 + "<br />");  //'number'
    document.write(typeof "Hello,World" + "<br />"); //'string'
    document.write(typeof true + "<br />");   //'boolean'
    document.write(typeof [1,2,4] + "<br />"); //'object'
    document.write(typeof new Date());          //'object'
</script>
```

Unary Operators – `void`

- Evaluates the given expression and returns **`undefined`**.
- Syntax

```
void expression
```

```
<a href="javascript:void(0);">Click here to do nothing</a><br />
<a href=
"javascript:void(document.body.style.backgroundColor='green');">
Click here for green background</a>
```

Unary Operators – delete

- Removes a property from an object.
- Syntax

  delete expression

```
<script type="text/javascript">
   x = 42;
   var y = 43;
   document.write(delete x + "<br />");  // print true
   document.write(delete y + "<br />");  // print false
   document.write(delete Math.PI + "<br />");   //print false
   function car(make){
      this.make = make;
   }
   var mycar = new car("Hyundai");
   document.write(mycar.make + "<br />");          //print 'Hyundai'
   document.write(delete mycar.make + "<br />");   //print true
   document.write(mycar.make);                     //print undefined
</script>
```

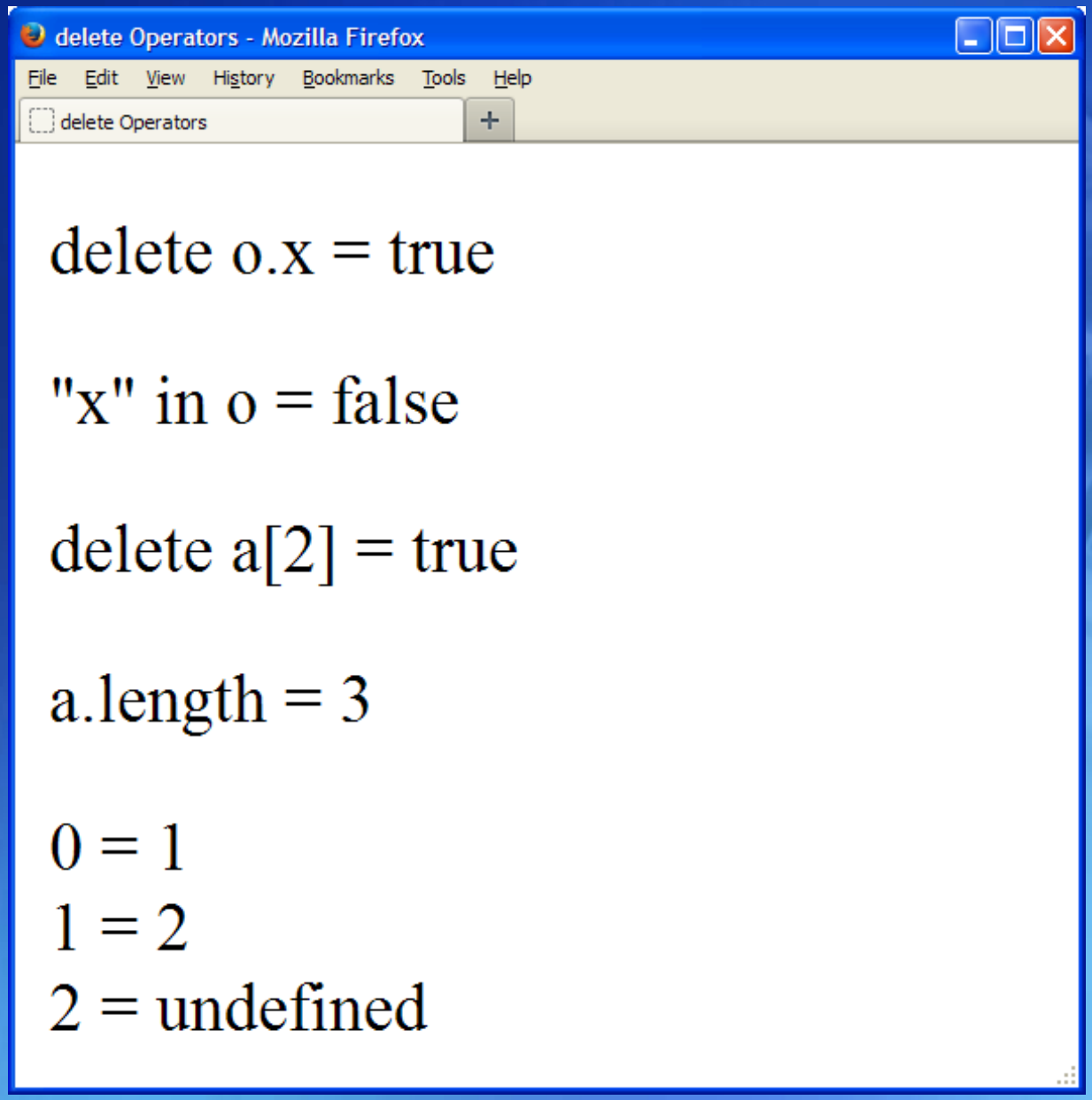
Lab3 : `delete` Operators

- Web Browsers
  - IE10, Firefox, Google Chrome, Opera, Safari
- Text Editors
  - Notepad++ or Editplus
- Files
  - deleteop.html

Lab3 : deleteop.html

```
3   <head>
4     <title> delete Operators </title>
5     <meta charset="utf-8">
6   </head>
7   <body>
8     <script>
9       var o = {x : 1, y : 2};
10      document.write("<p>delete o.x = " + (delete o.x) + "</p>");
11      document.write("<p>\"x\" in o = " + ("x" in o) + "</p>");
12
13      var a = [1,2,3];
14      document.write("<p>delete a[2] = " + (delete a[2]) + "</p>");
15      document.write("<p>a.length = " + a.length + "</p>");
16      for(var i = 0 ; i < a.length ; i++){
17        document.write(i + " = " + a[i] + "<br>");
18      }
19    </script>
20  </body>
```

Lab3 : Result


delete Operators - Mozilla Firefox
File Edit View History Bookmarks Tools Help
delete Operators
delete o.x = true
"x" in o = false
delete a[2] = true
a.length = 3
0 = 1
1 = 2
2 = undefined

Binary Operators – Arithmetic Operators

Purpose	Operator	Example
Addition	+	sum = num1 + num2
Subtraction	-	diff = num1 - num2
Multiplication	*	prod = num1 * num2
Division	/	quot = num1 / num2
Modulus	%	mod = num1 % num2

Binary Operators – Shift Operators

Purpose	Operator	Example
Left shift	<<	64 << 3
Right shift	>>	128 >> 1 256 >> 4 -256 >> 4
Unsigned right shift	>>>	-256 >>> 4

```
<script type="text/javascript">
    document.write((64 << 3) + "<br />");     //print 512
    document.write((128 >> 1) + "<br />");    //print 64
    document.write((256 >> 4) + "<br />");    //print 16
    document.write((-256 >> 4) + "<br />");   //print -16
    document.write(-256 >>> 4);               //print 268435440
</script>
```

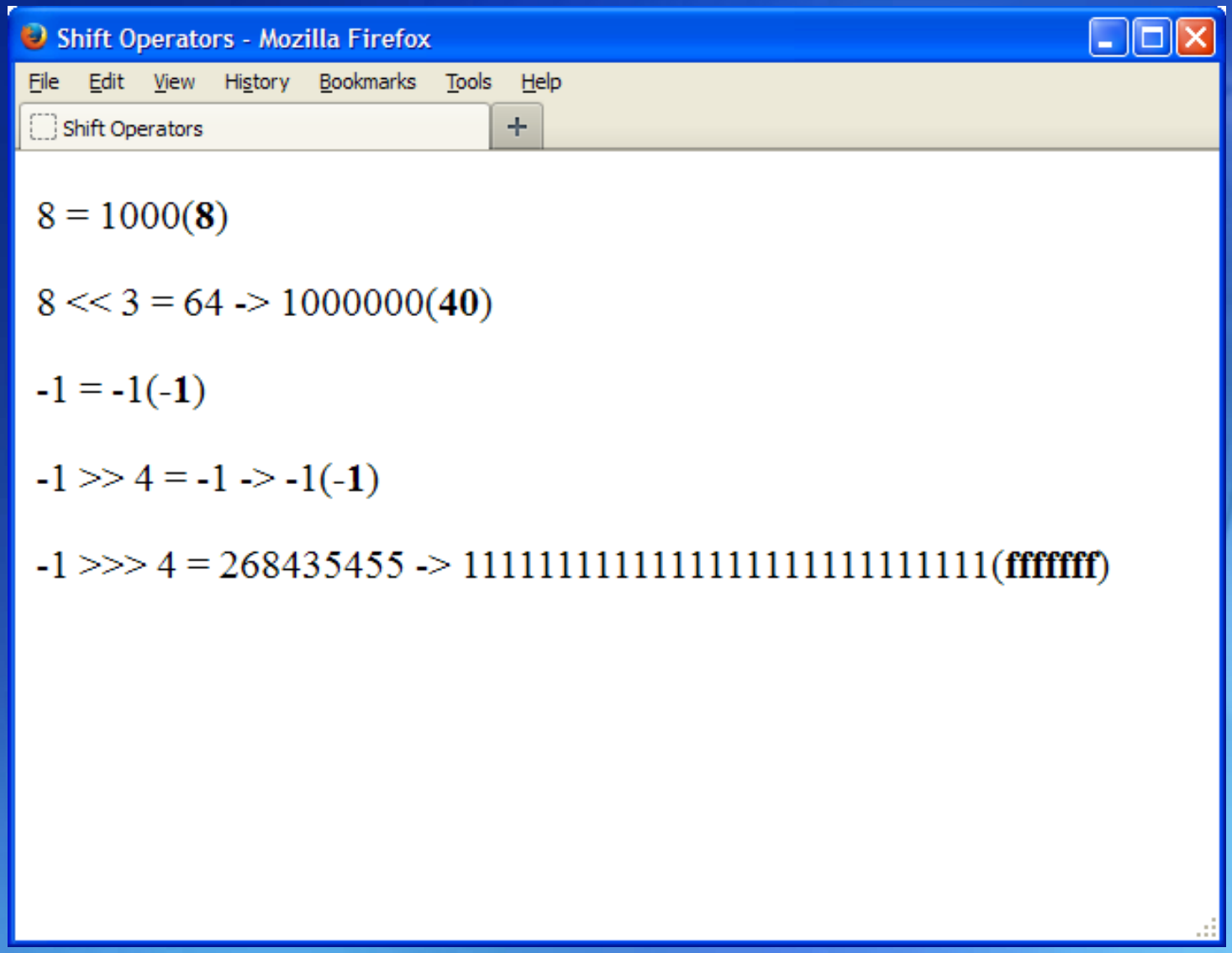
Lab4 : Shift Operators

- Web Browsers
  - IE10, Firefox, Google Chrome, Opera, Safari
- Text Editors
  - Notepad++ or Editplus
- Files
  - shiftop.html

Lab4 : shiftop.html

```
8   <script>
9       su = 8;
10      document.write("<p>8 = " + su.toString(2));
11      document.write("(<strong>" + su.toString(16) + "</strong>)</p>");
12      su1 = su << 3;
13      document.write("<p>8 << 3 = " + su1 + " -> " + su1.toString(2));
14      document.write("(<strong>" + su1.toString(16) + "</strong>)</p>");
15
16      su = -1;
17      document.write("<p>-1 = " + su.toString(2));
18      document.write("(<strong>" + su.toString(16) + "</strong>)</p>");
19      su1 = su >> 4;
20      document.write("<p>-1 >> 4 = " + su1 + " -> " + su1.toString(2));
21      document.write("(<strong>" + su1.toString(16) + "</strong>)</p>");
22      su2 = su >>> 4;
23      document.write("<p>-1 >>> 4 = " + su2 + " -> " + su2.toString(2));
24      document.write("(<strong>" + su2.toString(16) + "</strong>)</p>");
25  </script>
```

Lab4 : Result


Shift Operators - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Shift Operators
8 = 1000(8)
8 << 3 = 64 -> 1000000(40)
-1 = -1(-1)
-1 >> 4 = -1 -> -1(-1)
-1 >>> 4 = 268435455 -> 11111111111111111111111111111(fffffff)

Binary Operators – Relational Operators

Purpose	Operator	Example
Less than	<	4 < 5 return true
Less than or equal to	<=	5 <= 5 return true
Greater than	>	4 > 5 return false
Greater than or equal to	>=	4 >= 4 return true

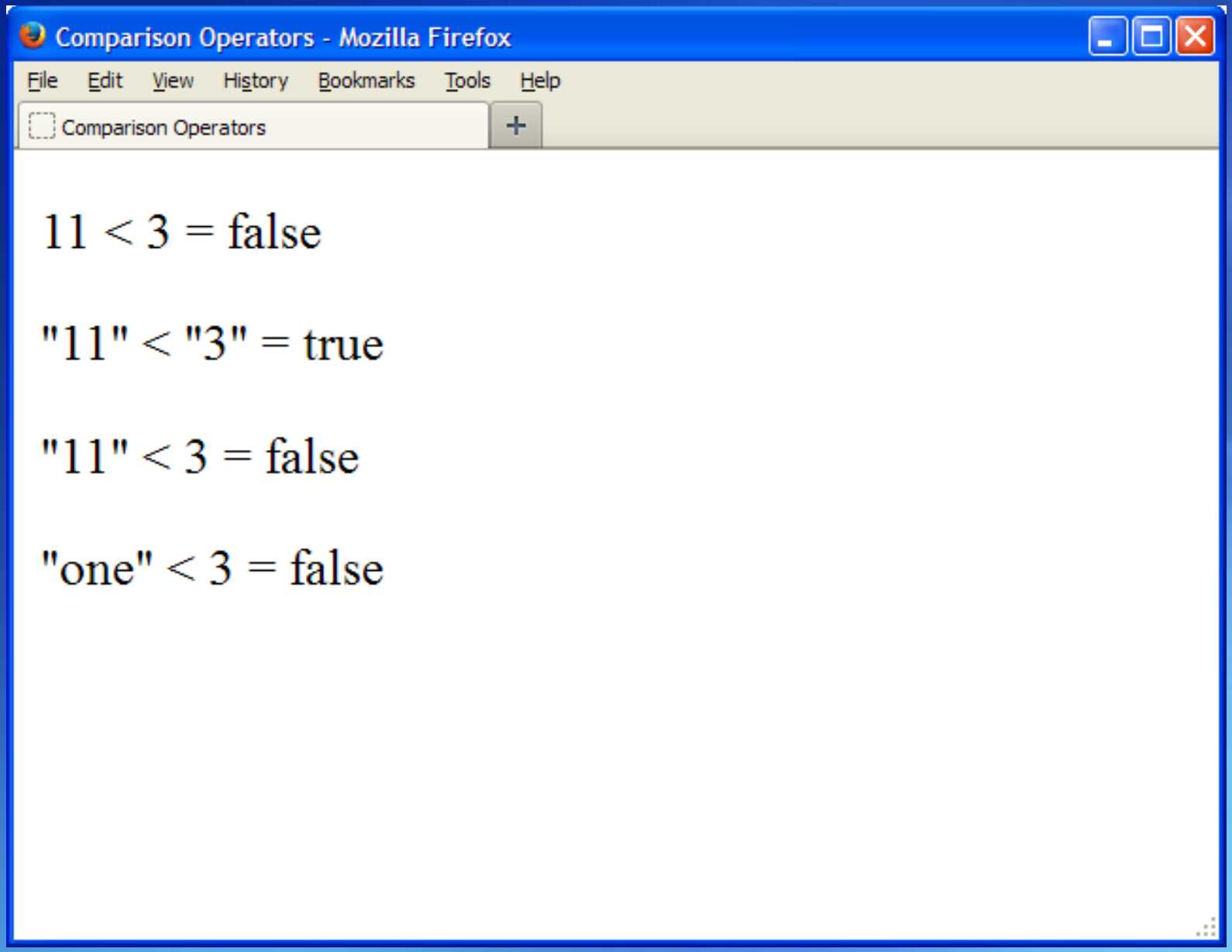
Lab5 : Comparison Operators

- Web Browsers
  - IE10, Firefox, Google Chrome, Opera, Safari
- Text Editors
  - Notepad++ or Editplus
- Files
  - comparisonop.html

Lab5 : comparisonop.html

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <title> Comparison Operators </title>
5          <meta charset="utf-8">
6      </head>
7      <body>
8          <script>
9              document.write("<p>11 < 3 = " + (11 < 3) + "</p>");
10             document.write("<p>\"11\" < \"3\" = " + ("11" < "3") + "</p>");
11             document.write("<p>\"11\" < 3 = " + ("11" < 3) + "</p>");
12             document.write("<p>\"one\" < 3 = " + ("one" < 3) + "</p>");
13         </script>
14     </body>
15 </html>
```

Lab5 : Result


Comparison Operators - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Comparison Operators
11 < 3 = false
"11" < "3" = true
"11" < 3 = false
"one" < 3 = false

Binary Operators – `in`

- Returns **`true`** if the specified property is in the specified object.
- Syntax

```
prop in expression
```

```
<script type="text/javascript">
   var trees = new Array("redwood", "bay", "cedar", "oak", "maple");
   document.write((3 in trees) + "<br />");       //print true
   document.write((6 in trees) + "<br />");       //print false
   document.write(("bay" in trees) + "<br />");  //print false
   document.write(("PI" in Math) + "<br />");    //print true
   var mycar = {make : "Honda", model : "Accord", year : 1998};
   document.write(("make" in mycar) + "<br />");   //print true
   document.write(("color" in mycar) + "<br />");   //print false
   delete mycar.make;
   document.write("make" in mycar);                //print false
</script>
```

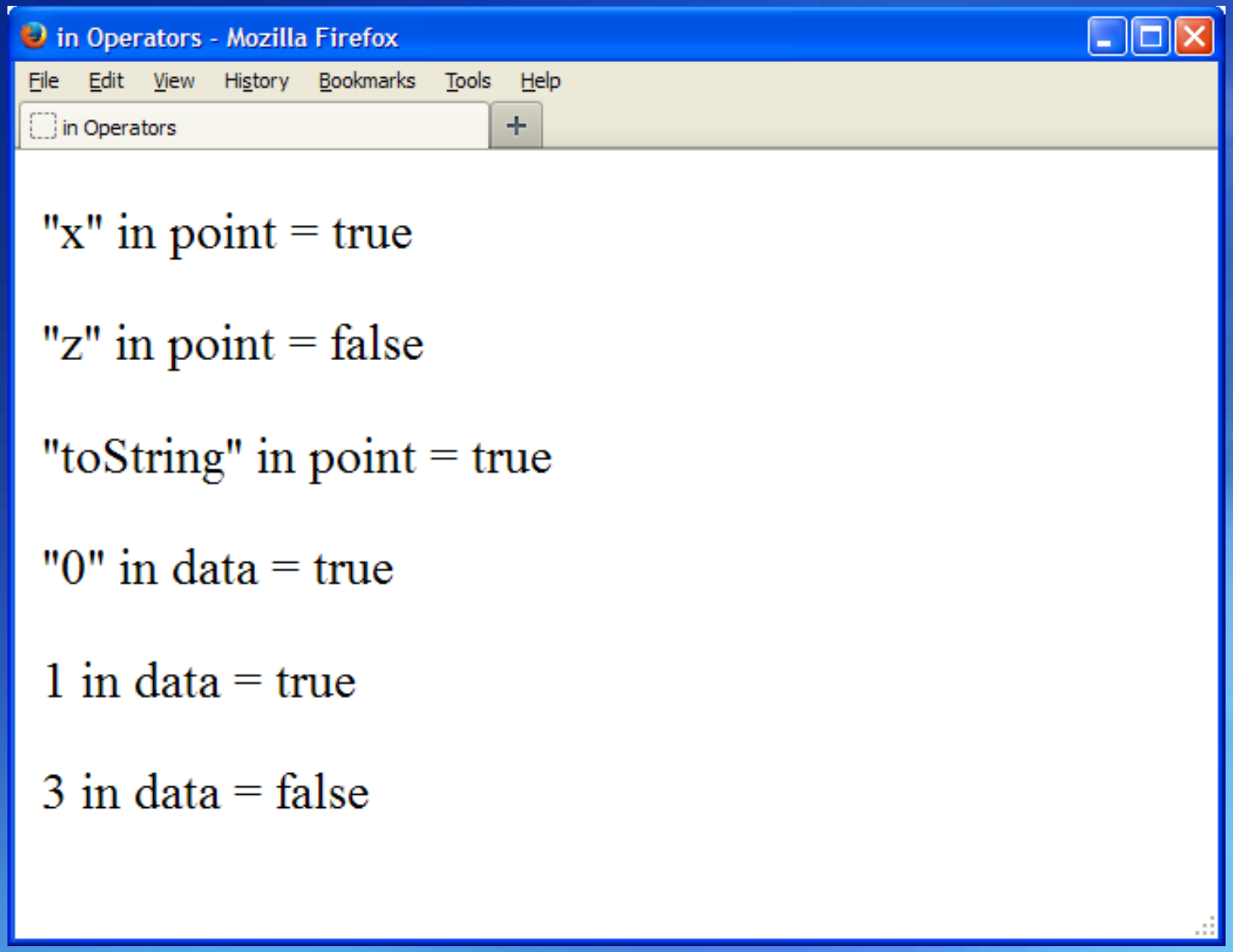
Lab6 : in Operators

- Web Browsers
  - IE10, Firefox, Google Chrome, Opera, Safari
- Text Editors
  - Notepad++ or Editplus
- Files
  - inop.html

Lab6 : inop.html

```
3    <head>
4      <title> in Operators </title>
5      <meta charset="utf-8">
6    </head>
7    <body>
8      <script>
9        var point = { x : 1, y : 1};
10       document.write("<p>\"x\" in point = " + ("x" in point) + "</p>");
11       document.write("<p>\"z\" in point = " + ("z" in point) + "</p>");
12       document.write("<p>\"toString\" in point = " + ("toString" in point) + "</p>");
13
14       var data = [7,8,9];
15       document.write("<p>\"0\" in data = " + ("0" in data) + "</p>");
16       document.write("<p>1 in data = " + (1 in data) + "</p>");
17       document.write("<p>3 in data = " + (3 in data) + "</p>");
18     </script>
19   </body>
20 </html>
```

Lab6 : Result


in Operators - Mozilla Firefox
File Edit View History Bookmarks Tools Help
in Operators
"x" in point = true
"z" in point = false
"toString" in point = true
"0" in data = true
1 in data = true
3 in data = false

Binary Operators – instanceof

- Tests whether an object has in its prototype chain the **prototype** property of a constructor.
- Syntax

```
object instanceof constructor
```

```
<script type="text/javascript">
   function C(){}    //defining a constructor
   function D(){}    //defining another constructor
   var obj = new C();
   document.write((obj instanceof C) + "<br />"); //print true
   document.write((obj instanceof D) + "<br />"); //print false
   document.write(C.prototype instanceof Object);  //print true
</script>
```

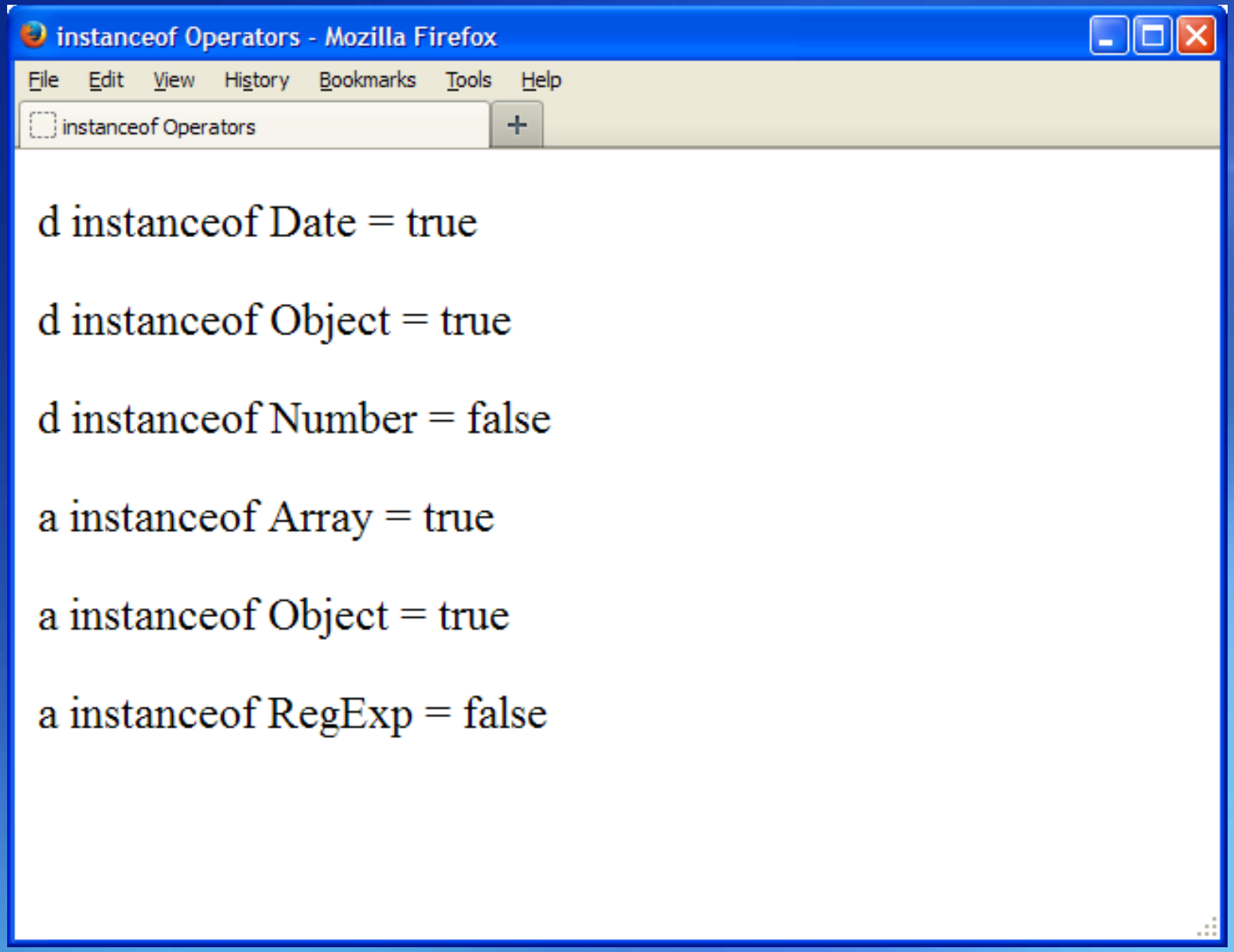
Lab7 : `instanceof` Operators

- Web Browsers
  - IE10, Firefox, Google Chrome, Opera, Safari
- Text Editors
  - Notepad++ or Editplus
- Files
  - instanceofop.html

Lab7 : instanceofop.html

```
3   <head>
4     <title> instanceof Operators </title>
5     <meta charset="utf-8">
6   </head>
7   <body>
8     <script>
9       var d = new Date();
10      document.write("<p>d instanceof Date = " + (d instanceof Date) + "</p>");
11      document.write("<p>d instanceof Object = " + (d instanceof Object) + "</p>");
12      document.write("<p>d instanceof Number = " + (d instanceof Number) + "</p>");
13
14      var a = [1,2,3];
15      document.write("<p>a instanceof Array = " + (a instanceof Array) + "</p>");
16      document.write("<p>a instanceof Object = " + (a instanceof Object) + "</p>");
17      document.write("<p>a instanceof RegExp = " + (a instanceof RegExp) + "</p>");
18    </script>
19  </body>
20 </html>
```

Lab7 : Result


instanceof Operators - Mozilla Firefox
File Edit View History Bookmarks Tools Help
instanceof Operators
d instanceof Date = true
d instanceof Object = true
d instanceof Number = false
a instanceof Array = true
a instanceof Object = true
a instanceof RegExp = false

Binary Operators – Equality Operators

Purpose	Operator
Equal	==
Not equal	!=
Strict equal	===
Strict not equal	!==

```
<script type="text/javascript">
    document.write((3 == "3") + "<br />");      //print true
    document.write((3 != "3") + "<br />");      //print false
    document.write((3 === "3") + "<br />");     //print false
    document.write((3 !== "3") + "<br />");     //print true
</script>
```

Binary Operators – Bitwise Operators

		& (Bitwise AND)	^ (Bitwise XOR)	| (Bitwise OR)
1	1	1	0	1
1	0	0	1	1
0	1	0	1	1
0	0	0	0	0

Binary Operators – Short-Circuit Logical Operators

		&& (Conditional AND)	|| (Conditional OR)
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

Ternary Operator

- The ? : operator
- (Condition) ? true : false

```
<script type="text/javascript">
    var bStop = false, nAge = 23;
    nAge > 18 ? (
        alert("OK, you can go."),
        location.assign("adult.html")
    ) : (
        bStop = true,
        alert("Sorry, your age too much young!")
    );
</script>
```

Assignment Operators

- The = operator

```
var su = 5;
```

- The +=, -=, *=, /=, %= operator

```
var su = 5;          su += 8;
```

- The &=, ^=, |= operator

```
var su = 5;          su |= 3;
```

- The <<=, >>=, >>>= operator

```
var su = -128;     su >>= 3;
```

Simple Programming Constructs

- Conditions - Decide at runtime whether to perform certain statements.
- Loops - Decide at runtime how many times to perform certain statements.
- Branches

Conditional Statements

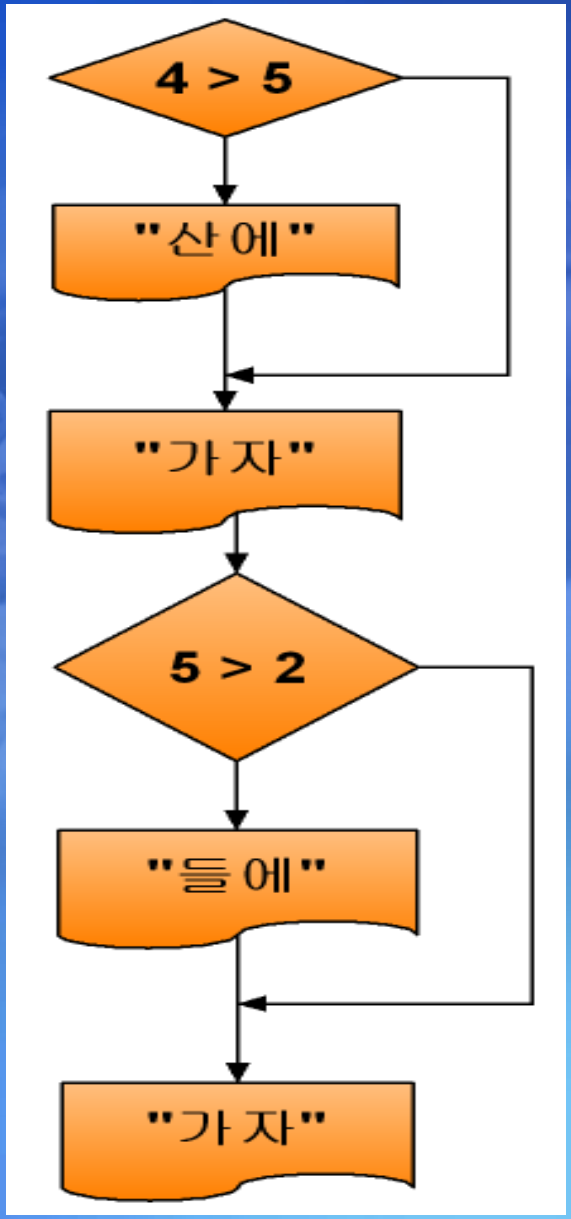
- Performs different actions for different decisions.
- `if` statement
- `if…else` statement
- `if…else if….else` statement
- `switch` statement

The `if` construct

- Use to execute some code only if a specified condition is `true`.
- Can use `if` with code blocks.

Syntax - if

```
if (condition)
{
    statement ;
}
```



Syntax – `if` (Cont.)

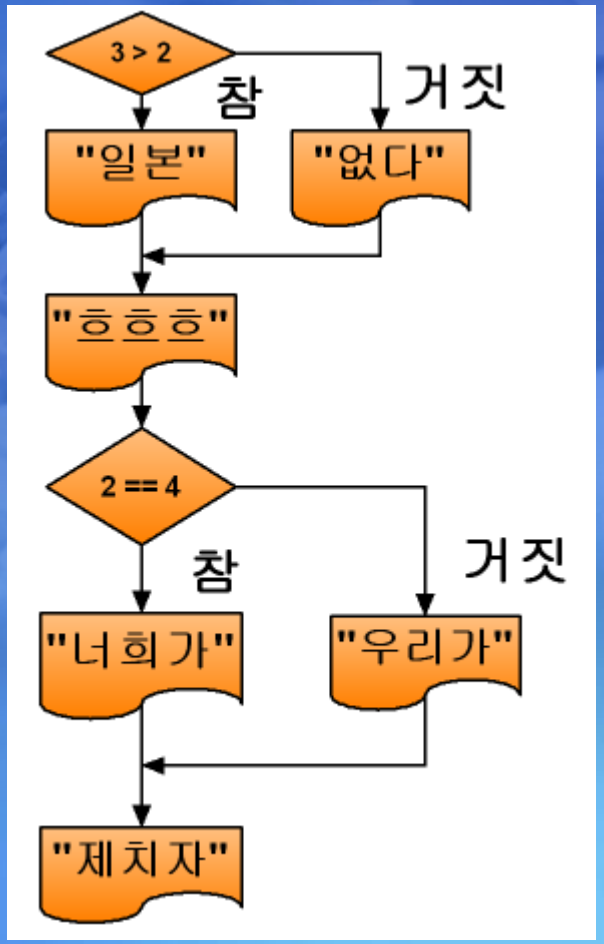
```
<script type="text/javascript">
  function choices(){
      var prefChoice = 1;
      var stateChoice = 'OR';
      var genderChoice = 'F';

      if(prefChoice === 1){
          alert("You've picked option 1. Here is what will happen...");

          if(genderChoice === 'M'){
              alert("You've picked 1 and you're from Oregon and you're a man.");
          }
      }
  }
</script>
```

Syntax – if (Cont.)

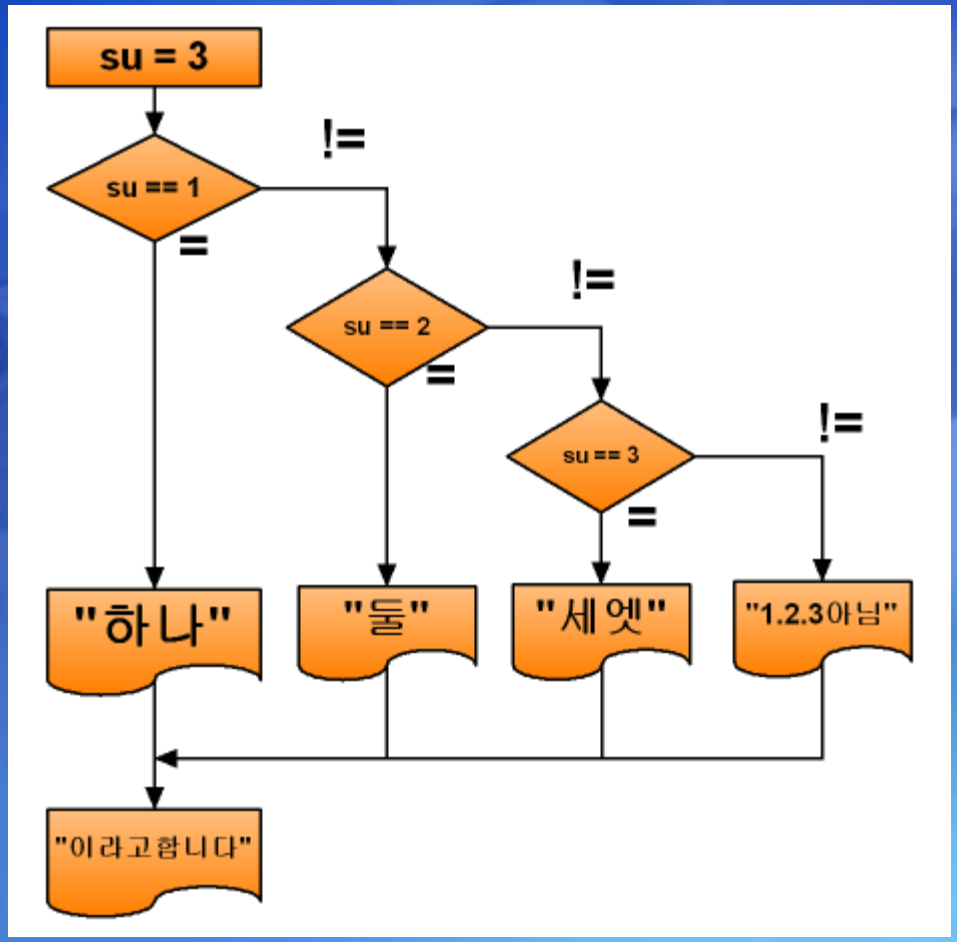
```
if (condition)
{
    statement ;
} else {
    statement ;
}
```



일본
흐흐흐
우리가
제치자

Syntax – if (Cont.)

```
if (condition)
{
    statement ;
}
else if (condition)
{
    statement ;
}
else
{
    statement ;
}
```



Syntax – `if` (Cont.)

```
<script type="text/javascript">
  function choices(){
      var stateCode = 'MO';
      var taxPercentage = 0.0;

      if(stateCode === 'OR') taxPercentage = 3.5;
      else if(stateCode === 'CA') taxPercentage = 5.0;
      else if(stateCode === 'MO') taxPercentage = 1.0;
      else taxPercentage = 2.0;

      alert(taxPercentage);
  }
</script>
```

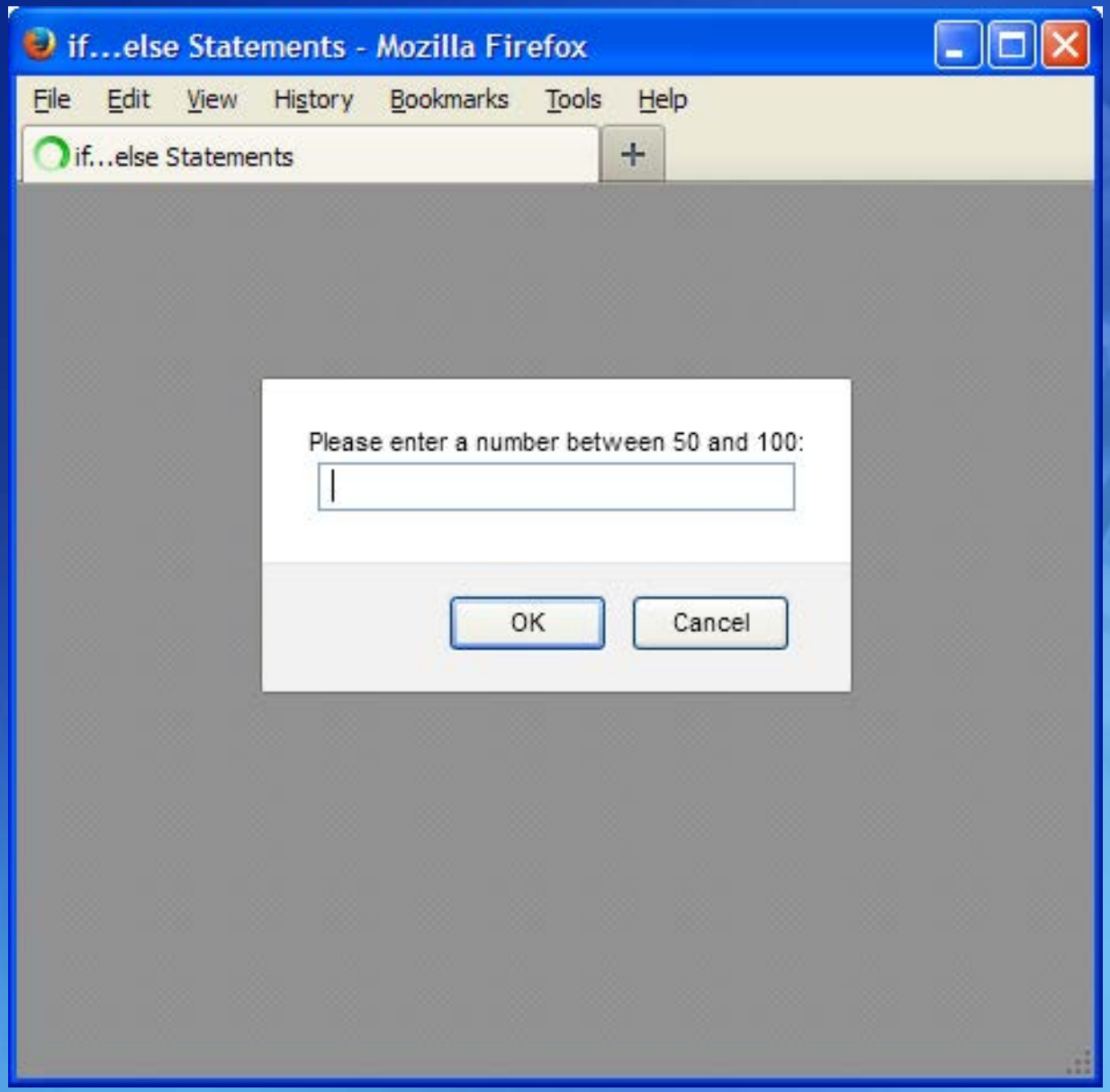
Lab8 : if…else if… Statements

- Web Browsers
  - IE10, Firefox, Google Chrome, Opera, Safari
- Text Editors
  - Notepad++ or Editplus
- Files
  - ifelse.html

Lab8 : ifelse.html

```
3   <head>
4      <title> if...else Statements </title>
5      <meta charset="utf-8">
6   </head>
7   <body>
8      <script>
9         var inputNum = prompt("Please enter a number between 50 and 100:");
10
11        if(isNaN(inputNum)){
12           alert(inputNum + " doesn't appear to be a number.");
13        } else if((inputNum > 99) || (inputNum < 51)){
14           alert("That number, " + inputNum + ", is not between 50 and 100.");
15        } else {
16           alert(inputNum + "is a valid number.");
17        }
18
19     </script>
20  </body>
```

Lab8 : Result

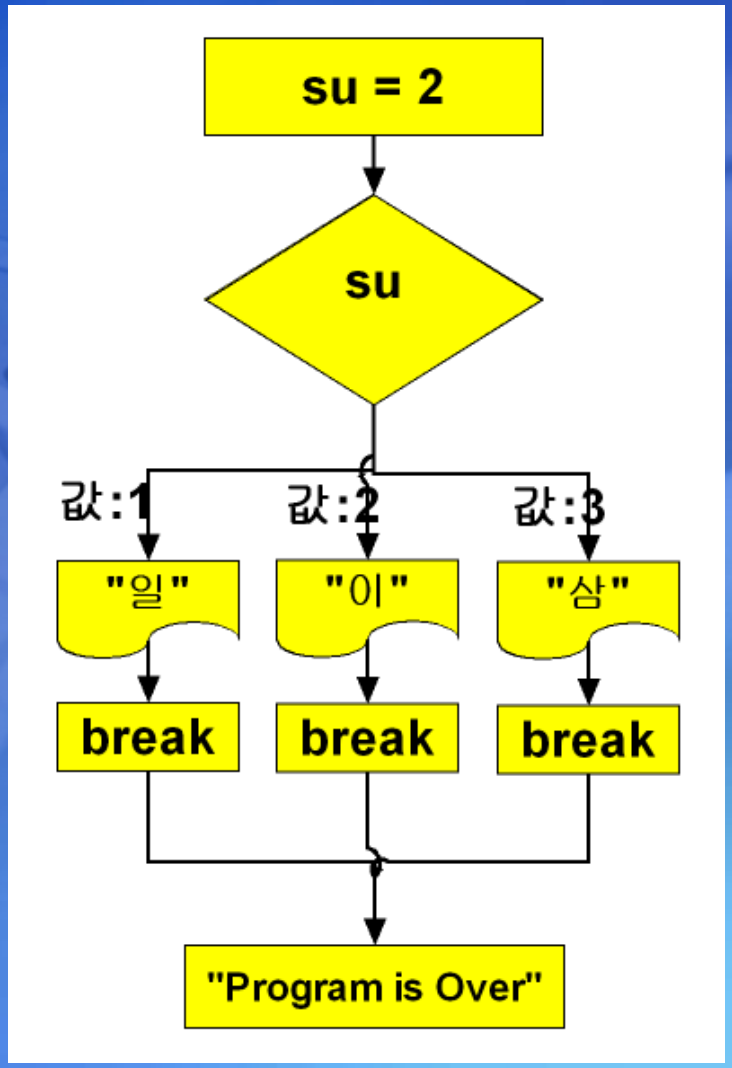

if...else Statements - Mozilla Firefox
File Edit View History Bookmarks Tools Help
if...else Statements
Please enter a number between 50 and 100:
OK
Cancel

The `switch` Construct

- Use to select one of many blocks of code to be executed.
- The `case` labels must be literals.
- The `default:` case is the same as the `else` in an `if` construct.
- The `break` statement is used to exit out of a `switch` statement.
- If a `case` statement does not contain a `break`, the line of code after the completion of the `case` will be executed.

Syntax – switch

```
switch (expression)
{
    case constant1 :
          statement ;
          [break ;]
    case constant2 :
          statement ;
          [break ;]
    default :
          statement ;
}
```



The switch Construct (Cont.)

```
<script type="text/javascript">
    var day = new Date().getDay();
    switch(day){
        case 0 : x = "Today it's Sunday"; break;
        case 1 : x = "Today it's Monday"; break;
        case 2 : x = "Today it's Tuesday"; break;
        case 3 : x = "Today it's Wednesday"; break;
        case 4 : x = "Today it's Thursday"; break;
        case 5 : x = "Today it's Friday"; break;
        default : x = "Today it's Saturday";
    }
    document.write(x);
</script>
```

The `switch` Construct (Cont.)

```
function choices(){
    var stateCode = 'MO';
    var taxPercentage = 0.0;
    switch (stateCode){
        case 'OR' :
        case 'MA' :
        case 'WI' :
            statePercentage = 0.5;
            taxPercentage = 3.5;
            break;
        case 'MO' :
            taxPercentage = 1.0;
            statePercentage = 1.5;
        case 'CA' :
        case 'NY' :
        case 'VT' :
            statePercentage = 2.6;
            taxPercentage = 4.5;
            break;
        case 'TX' :
            taxPercentage = 3.0;
            break;
        default :
            taxPertage = 2.0;
            statePercentage = 2.3;
    }
    alert("Tax is " + taxPercentage + " and State is " + statePercentage);
}
```

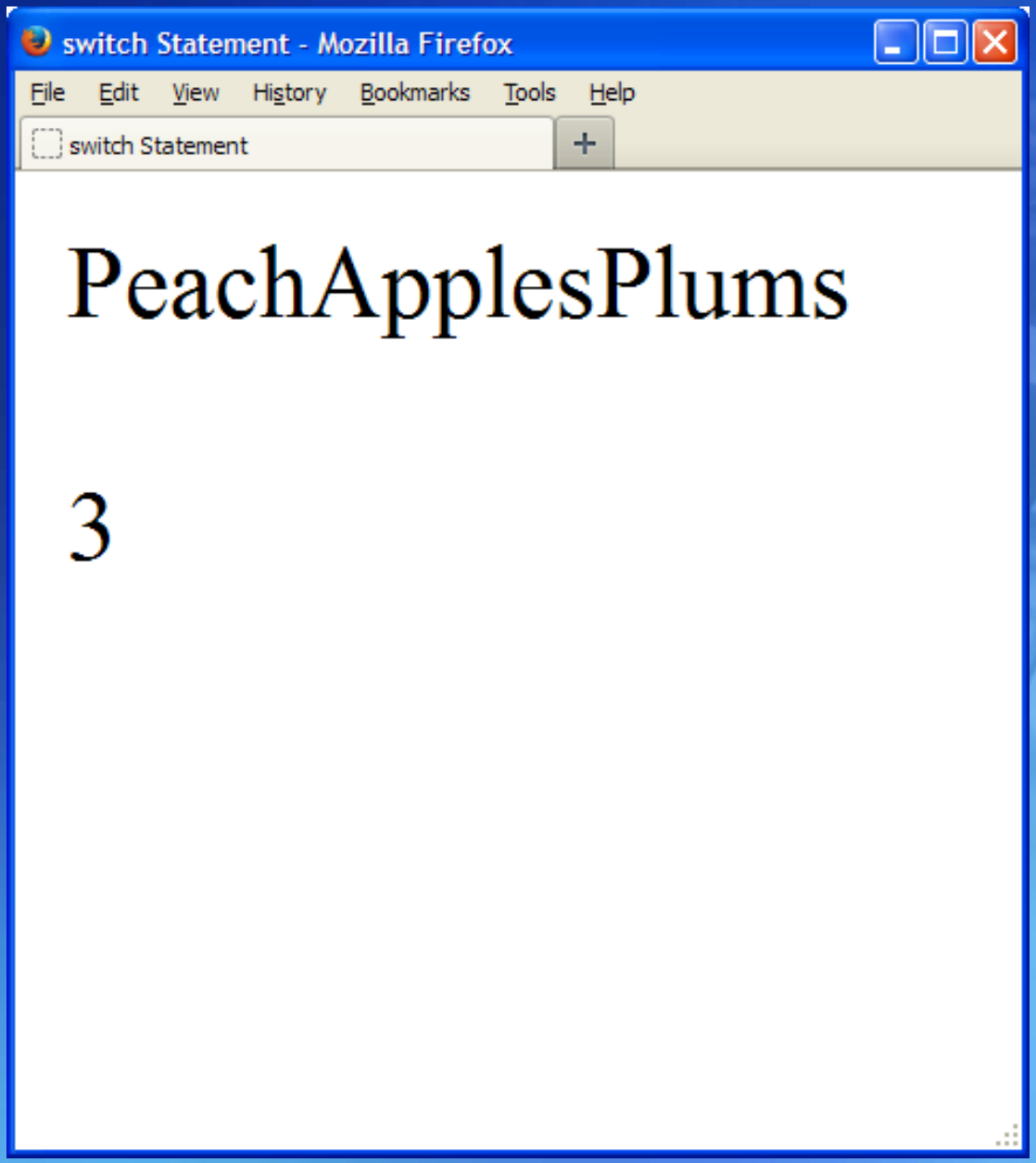
Lab9 : `switch` Statements

- Web Browsers
  - IE10, Firefox, Google Chrome, Opera, Safari
- Text Editors
  - Notepad++ or Editplus
- Files
  - switch.html

Lab9 : switch.html

```
3   <head>
4       <title> switch Statement </title>
5       <meta charset="utf-8">
6   </head>
7   <body>
8       <script>
9           var su = parseInt(Math.random() * 10 + 1);
10          switch(su){
11              case 1 : document.write("Banana"); break;
12              case 2 : document.write("Orange"); break;
13              case 3 : document.write("Peach");
14              case 4 : document.write("Apples");
15              case 5 : document.write("Plums"); break;
16              case 6 : document.write("Pineapples"); break;
17              case 7 : document.write(""); break;
18              default : document.write("Nuts");
19          }
20          document.write("<br><br>" + su);
21      </script>
22  </body>
```

Lab9 : Result


switch Statement - Mozilla Firefox
File Edit View History Bookmarks Tools Help
switch Statement
PeachApplesPlums
3

Loop Statements

- Want the same block of code to run over and over again in a row.
- Instead, can use loops to perform a task like this.
- `for` statement
- `while` statement
- `do...while` statement
- `for...in` statement
- `for each...in` statement

The `for` Loop

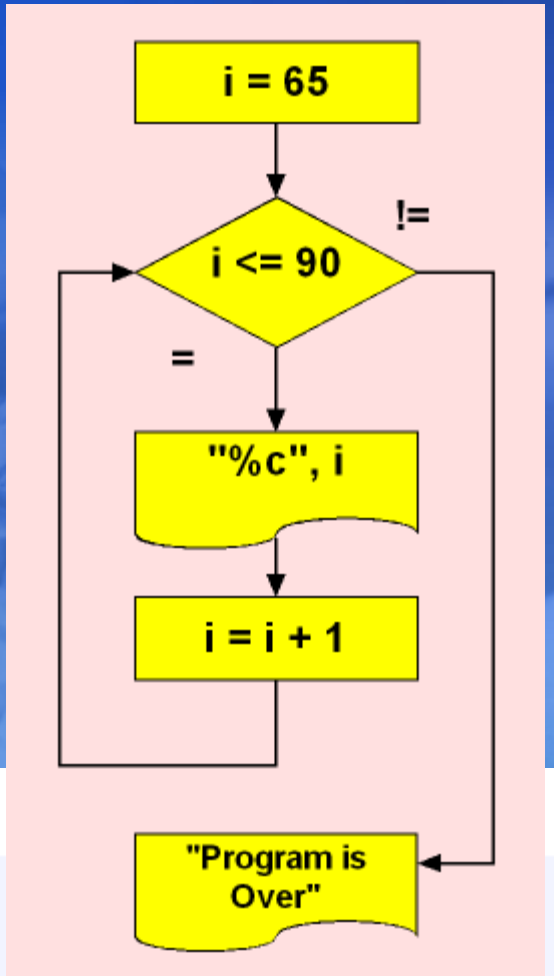
- Provides a compact way to iterate over a range of values.
- Initialize - Is the section that is processed *once*, before any other part of the loop.
- Condition - Is the section that is processed just before each iteration of the loop.
- Statement - Is the statement or code block which is processed with every loop iteration.
- Update - Is the section that is processed after the body but before each subsequent retest of the condition.

Syntax - `for`

```
for ( [initialize];[condition];[update] )
{
    statement ;
}
```

- Multiple initializations must be separated with commas(`,`) not semi-colons(`;`).
- Condition must be a *boolean* expression.

Syntax – for (Cont.)




```
<script type="text/javascript">
   for(var i = 65; i <= 90 ; i++)
   {
      document.write(String.fromCharCode(i) + "  ");
   }
</script>
```

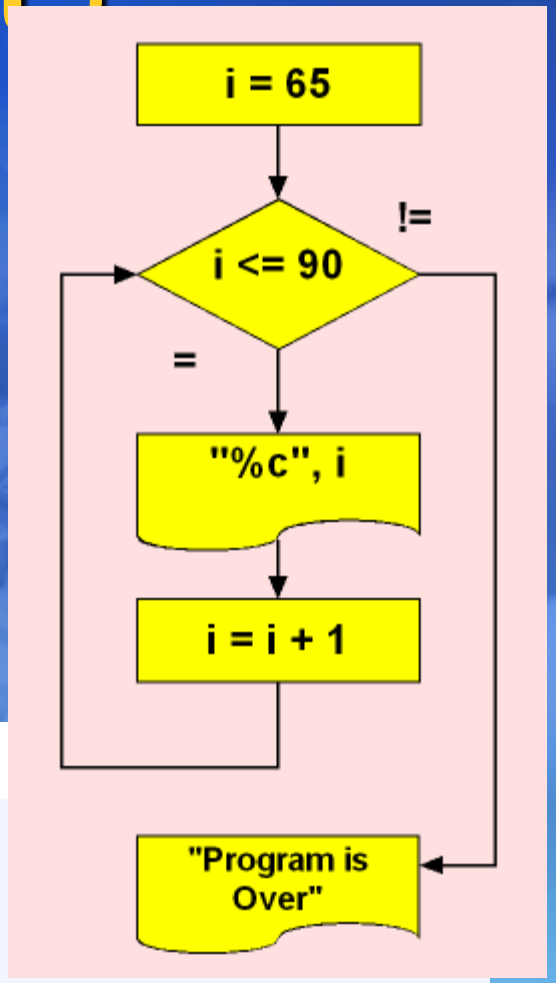
The `while` loop

- Continually execute a block of statements while a condition remains *true*.
- Can perform more than one statement by using a code block.

Syntax – while

```
initialization ;
while (condition) {
      statement ;
      update ;
}
```

Syntax – while (Cont.)




```
<script type="text/javascript">
    var i = 65;
    while (i <= 90)
    {
        document.write(String.fromCharCode(i) + "  ");
        i++;
    }
</script>
```

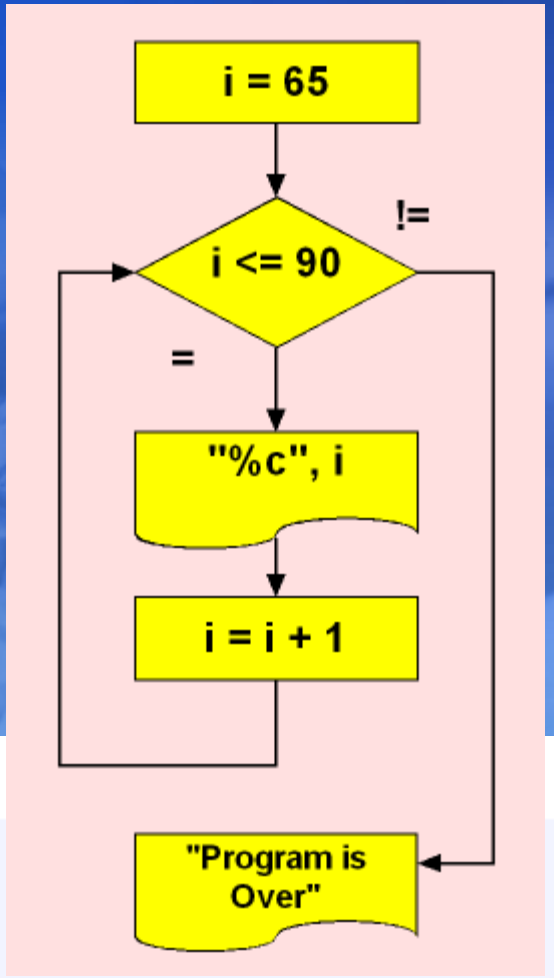
The do Loop

- `while` and `for` loops are used for *zero/many* iterative loops.
- `do` is used for *one/many* iterative loops.
- Condition at the bottom of the loop is processed after the body.
- Body of loop is processed at least *once*.

Syntax - do

```
initialization ;
do {
    statement ;
    update;
} while (condition) ;
```

Syntax – do (Cont.)




```
<script type="text/javascript">
    var i = 65;
    do
    {
        document.write(String.fromCharCode(i) + "  ");
        i++;
    }while(i <= 90);
</script>
```

Comparing Loop Constructs

- `while` – Iterates indefinitely through statements and performs the statements zero or more times.
- `do` – Iterates indefinitely through statements and performs the statements *one* or more times.
- `for` – Steps through statements a predefined number of times

The `for…in` Loop

- Iterates over the enumerable properties of an object, in arbitrary order.
- For each distinct property, statements can be executed.
- Syntax

```
for (variable in object){}
```

The for…in Loop (Cont.)

```
<script type="text/javascript">
    var array = {a : 1, b:2, c:3};
    for(var i in array){
        document.write(i + " = " + array[i] + "<br />");
    }
</script>
```

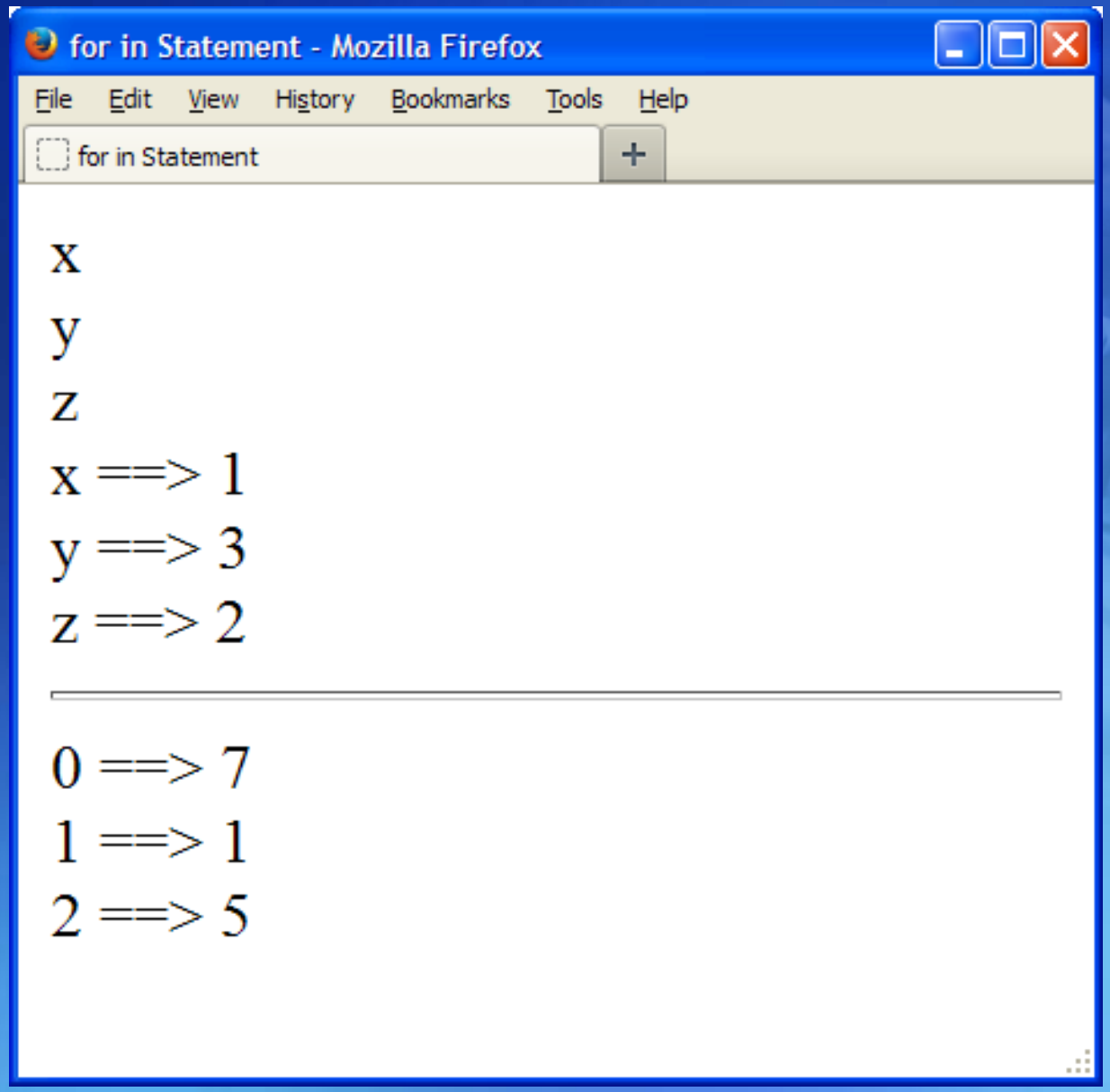
Lab10 : `for in` Statements

- Web Browsers
  - IE10, Firefox, Google Chrome, Opera, Safari
- Text Editors
  - Notepad++ or Editplus
- Files
  - forin.html

Lab10 : forin.html

```
3   <head>
4     <title> for in Statement </title>
5     <meta charset="utf-8">
6   </head>
7   <body>
8     <script>
9       var obj = {x : 1, y : 3, z : 2 };
10      for(var a in obj){
11        document.write(a + "<br>");
12      }
13
14      for(var a in obj){
15        document.write(a + " ==> " + obj[a] + "<br>");
16      }
17      document.write("<hr>");
18
19      var array = [7,1,5];
20      for(var n in array){
21        document.write(n + " ==> " + array[n] + "<br>");
22      }
23    </script>
24  </body>
```

Lab10 : Result



The `for each…in` Loop

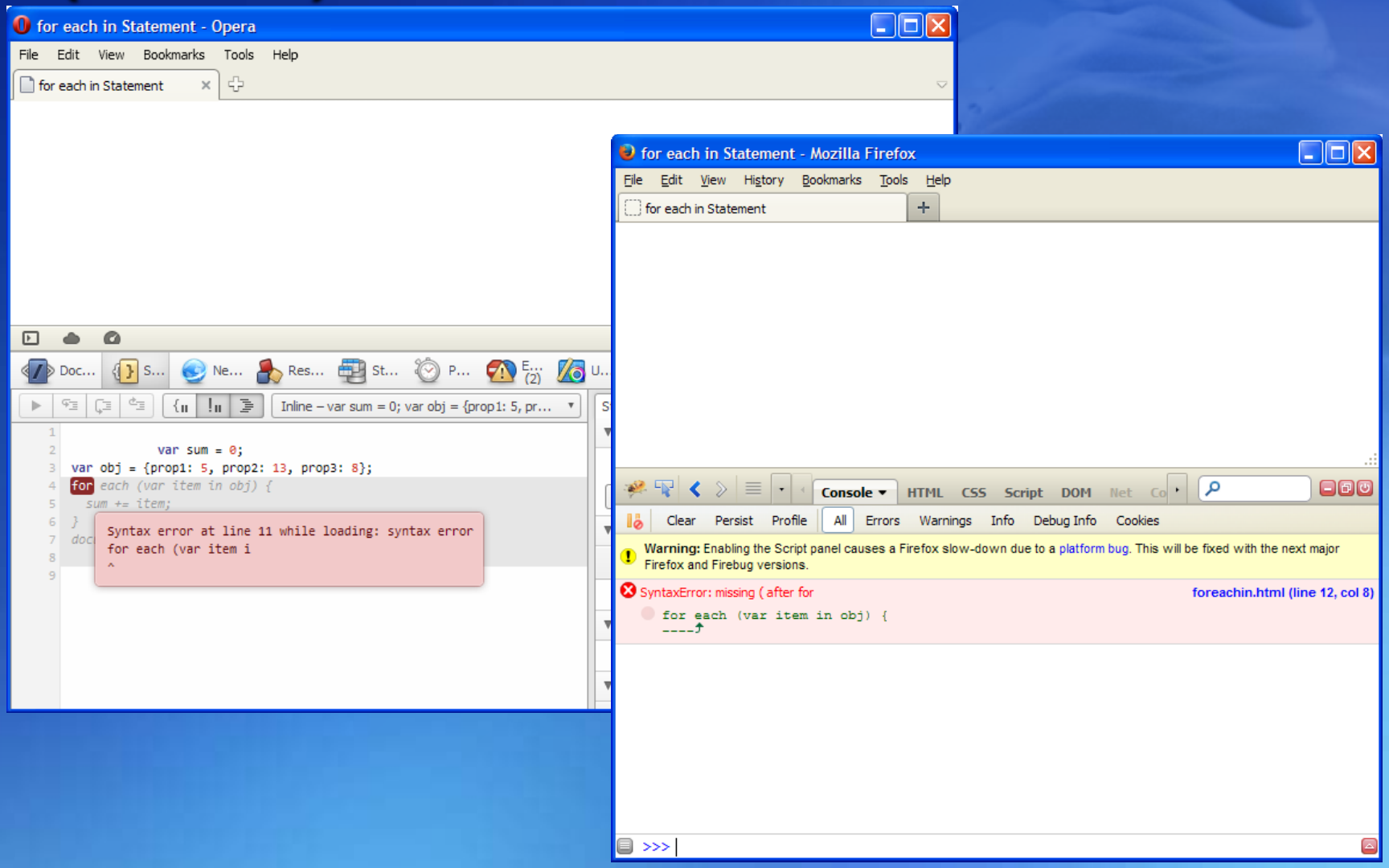
- Iterates a specified variable over all values of object's properties.
- For each distinct property, a specified statement is executed.
- Is part of the ECMA-357 standard.
- Is *not* widely supported by non-Mozilla browsers, *not* part of the ECMAScript standard.
- Since JavaScript 1.6
- Syntax

```
for each (variable in object) { }
```

The for each…in Loop (Cont.)

```
<script type="text/javascript">
    var sum = 0;
    var array = {a : 5, b:13, c:8};
    for each (var i in array){
        sum += i;
    }
    document.write("sum = " + sum);
</script>
```

The for each…in Loop (Cont.)



The `continue` Statement

- Permits you to end a loop iteration
- Used inside **`while`**, **`for`**, and **`do`** loops only
- Should be used only when the alternative code is much more complex

```
<script type="text/javascript">
   for(var i = 0 ; i < 10 ; i++)
   {
      if (i == 3) continue;
      document.write("The number is " + i + "<br />");
   }
</script>
```

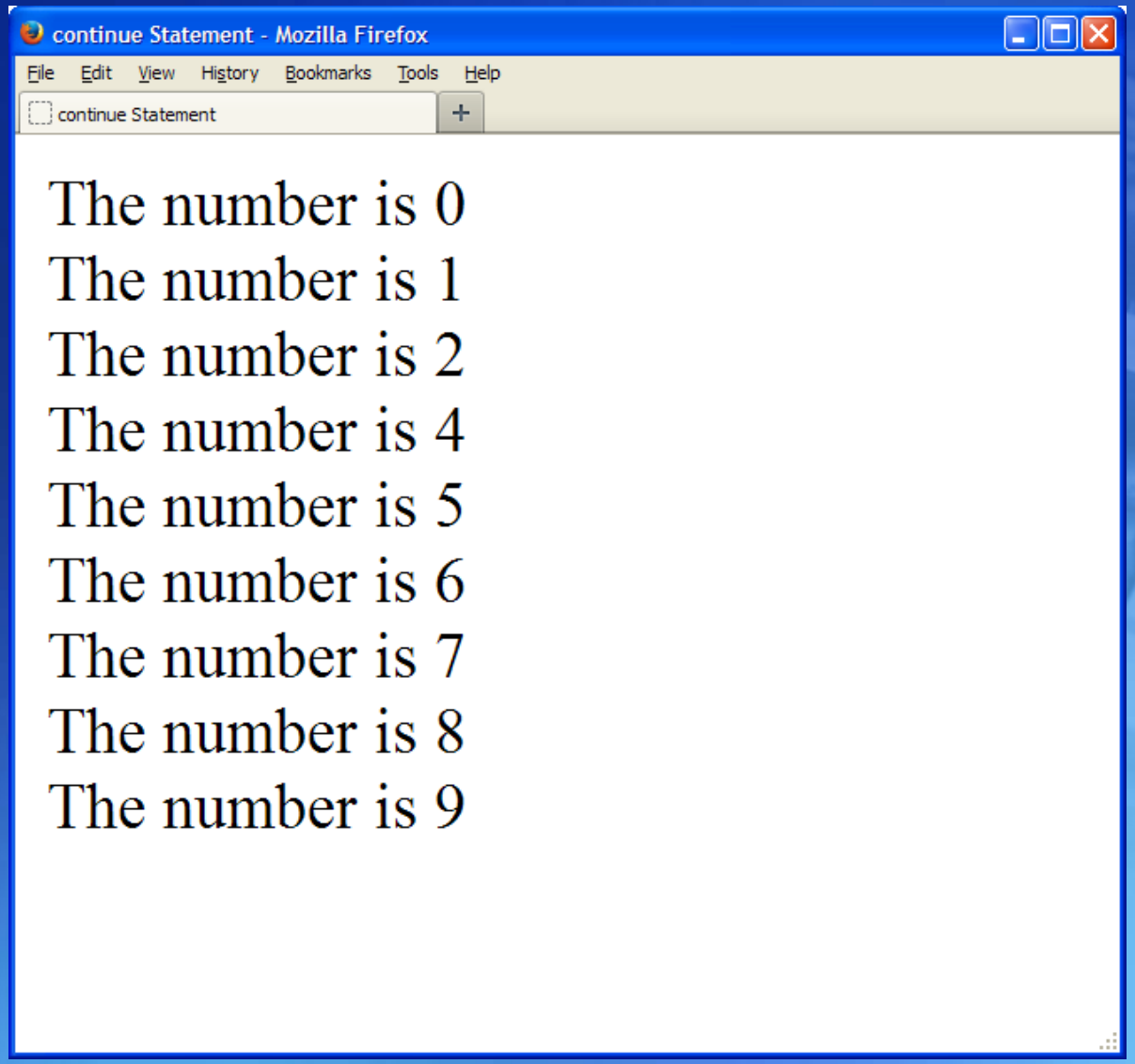
Lab11 : `continue` with label Statements

- Web Browsers
  - IE10, Firefox, Google Chrome, Opera, Safari
- Text Editors
  - Notepad++ or Editplus
- Files
  - continue.html

Lab11 : continue.html

```
3   <head>
4       <title> continue Statement </title>
5       <meta charset="utf-8">
6   </head>
7   <body>
8       <script>
9           mainloop: for(var i = 0 ; i < 10 ; i++){
10              if(i == 3) continue mainloop;
11              document.write("The number is " + i + "<br>");
12          }
13      </script>
14  </body>
15  </html>
```

Lab11 : Result



The `break` Statement

- The **`break`** statement permits the controlled and immediate termination of a loop or **`switch`** statement.
- It can be used to prevent fall thru.
- The **`break`** statement is valid inside **`while`**, **`for`**, **`do`**, and **`switch`** constructs only.

```
<script type="text/javascript">
   for(var i = 0 ; i < 10 ; i++)
   {
      if (i == 3) break;
      document.write("The number is " + i + "<br />");
   }
</script>
```

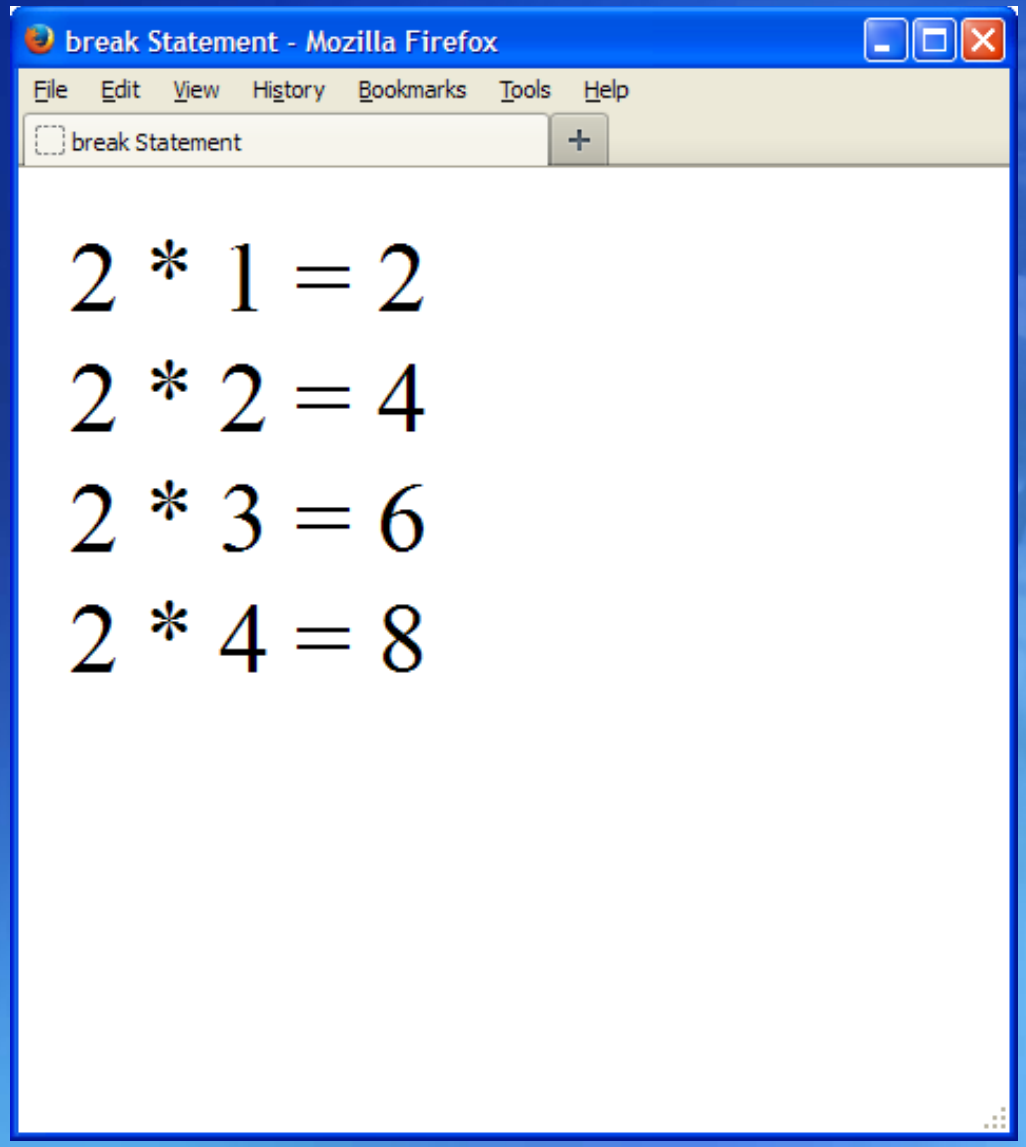
Lab12 : break with label Statements

- Web Browsers
  - IE10, Firefox, Google Chrome, Opera, Safari
- Text Editors
  - Notepad++ or Editplus
- Files
  - break.html

Lab12 : break.html

```
3   <head>
4       <title> break Statement </title>
5       <meta charset="utf-8">
6   </head>
7   <body>
8       <script>
9           Loop1 :
10              for(var i = 2; i <= 9 ; i++){
11                  for(var j = 1 ; j <= 9; j++){
12                      if(j == 5) break Loop1;
13                      //break;
14                      //continue Loop1;
15                      //continue;
16                      document.write(i + " * " + j + " = " + i * j + "<br>");
17                  }
18              }
19      </script>
20  </body>
```

Lab12 : Result



try catch finally

- Is JavaScript's exception handling mechanism.
- The `try` statement lets you test a block of code for errors.
- The try block is followed by a `catch` clause.
- The `catch` statement lets you handle the error.

`try` `catch` `finally` (Cont.)

- The `catch` clause is followed by a `finally` block.
- The `finally` clause is *guaranteed* to be executed, regardless of what happens in the `try` block.
- Both the `catch` and `finally` blocks are optional.
- *But*, a `try` block must be accompanied by at least one of these blocks.
- The `try`, `catch`, and `finally` blocks all begin and end with curly braces.

try catch finally (Cont.)

```
try {
   ……
}
catch(e) {
   ……
}
finally {
   ……
}
```

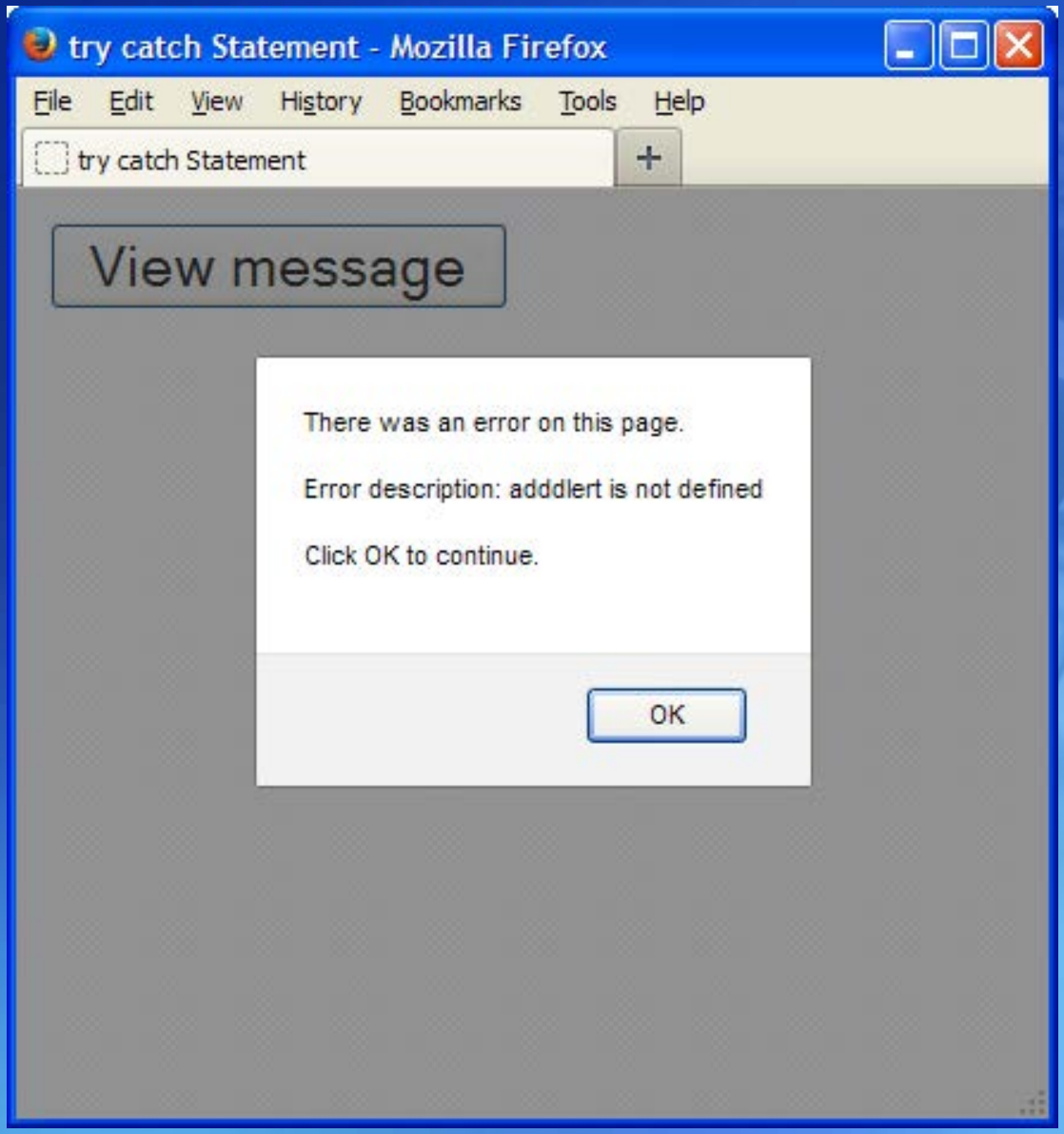
Lab13 : `try catch` Statements

- Web Browsers
  - IE10, Firefox, Google Chrome, Opera, Safari
- Text Editors
  - Notepad++ or Editplus
- Files
  - trycatch.html

Lab13 : trycatch.html

```
3   <head>
4       <title> try catch Statement </title>
5       <meta charset="utf-8">
6       <script>
7           var txt="";
8           function message() {
9               try {
10                adddlert("Welcome guest!");
11              } catch(err) {
12                txt="There was an error on this page.\n\n";
13                txt+="Error description: " + err.message + "\n\n";
14                txt+="Click OK to continue.\n\n";
15                alert(txt);
16              }
17          }
18      </script>
19  </head>
20  <body>
21      <input type="button" value="View message" onclick="message()" />
22  </body>
```

Lab13 : Result



`throw` Statement

- The `throw` statement lets you create custom errors.
- If you use the `throw` statement together with `try` and `catch`, you can control program flow and generate custom error messages.

```
throw exception;
```

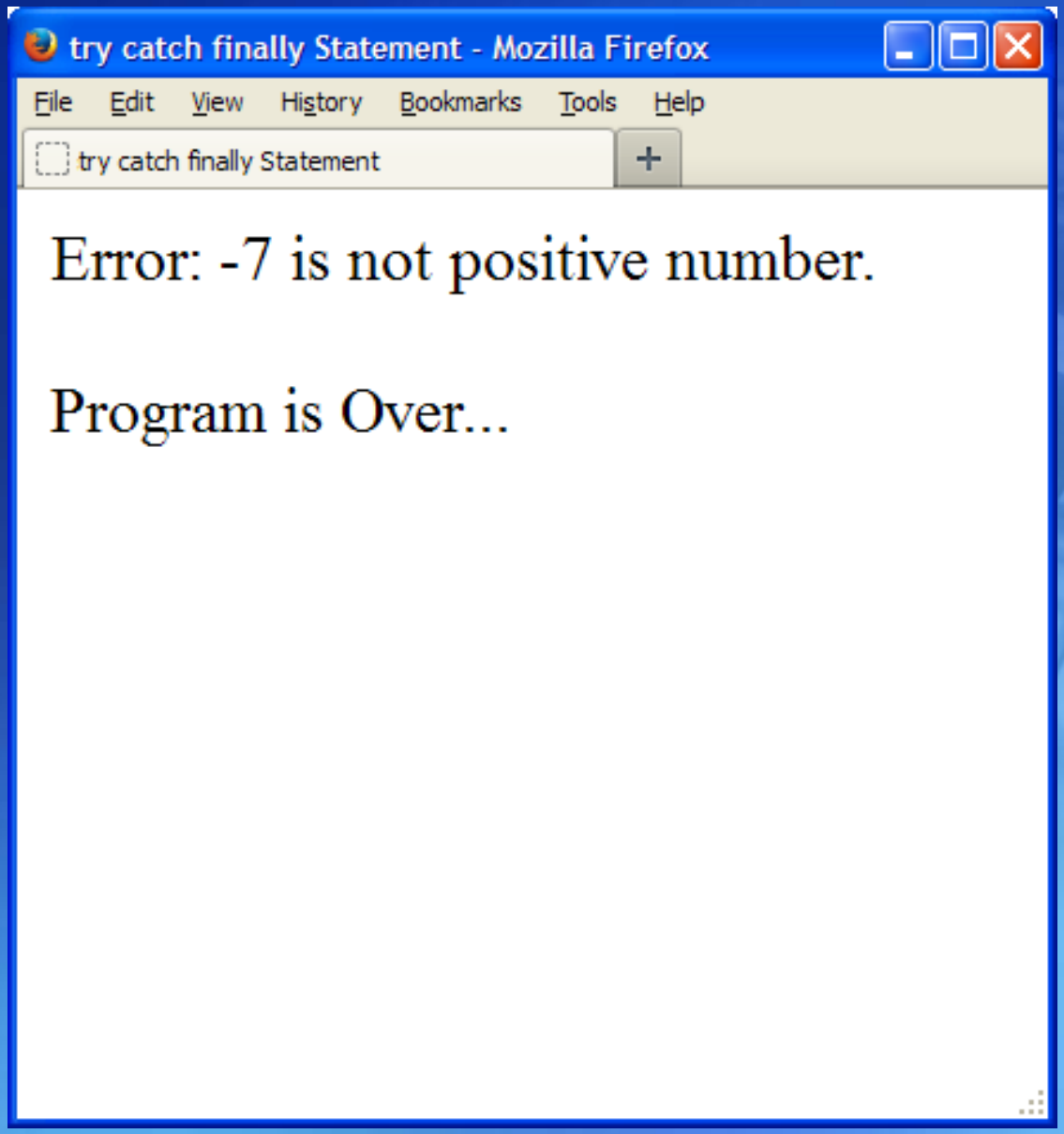
Lab14 : `try catch finally` Statements

- Web Browsers
  - IE10, Firefox, Google Chrome, Opera, Safari
- Text Editors
  - Notepad++ or Editplus
- Files
  - trycatch1.html

Lab14 : trycatch1.html

```
3   <head>
4      <title> try catch finally Statement </title>
5      <meta charset="utf-8">
6   </head>
7   <body>
8      <script>
9         var su = prompt("Input a positive number : ");
10        su = parseInt(su);
11        try{
12           if(su < 0) throw new Error(su + " is not positive number.");
13           document.write("su = " + su);
14        }catch(ex){
15           document.write(ex);
16        } finally{
17           document.write("<br><br>Program is Over...");
18        }
19     </script>
20  </body>
```

Lab14 : Result



Lab15 : try catch finally throw Statements

- Web Browsers
  - IE10, Firefox, Google Chrome, Opera, Safari
- Text Editors
  - Notepad++ or Editplus
- Files
  - trycatchthrow.html

Lab15 : trycatchthrow.html

```
7   <body>
8     <script>
9       function myFunction() {
10        var y = document.getElementById("mess");
11        y.innerHTML = "";
12        try {
13          var x = document.getElementById("demo").value;
14          if(x == "")    throw "empty";
15          if(isNaN(x)) throw "not a number";
16          if(x > 10)     throw "too high";
17          if(x < 5)      throw "too low";
18        } catch(err) {
19          y.innerHTML = "Error: " + err + ".";
20        }
21      }
22    </script>
23
24    <h1>try catch finally with throw Statement</h1>
25    <p>Please input a number between 5 and 10:</p>
26      <input id="demo" type="text">
27      <button type="button" onclick="myFunction()">Test Input</button>
28    <p id="mess"></p>
29  </body>
```

Lab15 : Result

with Statement

- The `with` statement is used to temporarily extend the scope chain.
- It has the following syntax.

```
with (object)
    statement
```

```
<script>
   var a, x, y;
   var r = 10;

   with(Math){
      a = PI * r * r;
      x = r * cos(PI);
      y = r * sin(PI / 2);
   }
   document.write(a + ", " + x + ", " + y);
</script>
```

Quiz

1. In the following, and parentheses to the expression so that it evaluates to **8**:

```
var valA = 37;
var valB = 3;
var valC = 18;
var resultOfComp = valA – valB % 3 / 2 * 4 + valC – 3;
```

Quiz

1. In the following, and parentheses to the expression so that it evaluates to **8**:

```
var valA = 37;
var valB = 3;
var valC = 18;
var resultOfComp = valA – valB % 3 / 2 * 4 + valC – 3;
```

```
var resultOfComp =
              ( valA – valB ) % 3 / 2 * ( 4 + valC ) - 3;
```

Quiz (Cont.)

2. Using a switch statement, test an expression for a value of '`one`', '`two`', or '`three`', and set a variable to '`OK`' if the expression is '`one`' or '`two`'; '`OK2`' if the expression is '`three`'; and '`NONE`' if it doesn't match any of these.

Quiz (Cont.)

2. Using a switch statement, test an expression for a value of '**one**', '**two**', or '**three**', and set a variable to '**OK**' if the expression is '**one**' or '**two**'; '**OK2**' if the expressio
doesn't m

```
switch(val) {
        case 'one' :
        case 'two' :
                result = 'OK';
                break;
        case 'three' :
                result = 'OK2';
                break;
        default :
                result =
'NONE';
}
```

Quiz (Cont.)

3. You have three variables: `varOne`, `varTwo`, and `varThree`. How would you test all three such that a block of code is processed only if `varOne` is `33`, `varTwo` is less than or equal to `100`, but `varThree` is greater than `0`?

Quiz (Cont.)

3. You have three variables: `varOne`, `varTwo`, and `varThree`. How would you test all three such that a block of code is processed only if `varOne` is `33`, `varTwo` is less than or equal to `100`, but `varThree` is greater than `0`?

```
if ((varOne == 33) && (varTwo <= 100) && (varThree > 0))
```

Quiz (Cont.)

4. In what three ways can you process a block of code exactly six times? In what conditions would you use one technique over another?

Quiz (Cont.)

4. In what three ways can you process a block of code exactly six times? In what conditions would you use one technique over another?

```
for (var i = 0; i < 6; i++) {
...
}
```

```
i = 0;
while (i < 6)
{
      ...
      i++;
}
```

```
i = 0;
do {
      i++;
      ...
}while (i < 6);
```

Quiz (Cont.)

5. Would you alter the following conditional statement, and if so, why?

```
if (valTest1 == valTest2) ...
```

Quiz (Cont.)

5. Would you alter the following conditional statement, and if so, why?

```
if (valTest1 == valTest2) ...
```

```
if (valTest1 === valTest2)
```