WEB PROGRAMMING SCV1223

Javascript

Dr. Md Sah bin Hj Salam

En. Jumail bin Taliba

Outline

- Introduction
- Fundamental of Javascript:
  - Keywords, variables, operators
  - Control Statements
  - Functions
  - Arrays
  - Objects

Reference:
Internet & World Wide Web: How To Program, 3rd Ed., Dietel & Goldberg, Prentice Hall

Introduction to Javascript

- To make web pages more dynamic and interactive
- An Interpreter-based language
- It isn't Java
- Case-sensitive
- Must be embedded into HTML
- Browser dependent

Introduction to Javascript

Embedding Javascript into HTML:

```
<script type="text/javascript">
    <!-
     Javascript code goes here
    // -->
</script>
```

```
<script language="javascript">
    <!-
     Javascript code goes here
    // -->
</script>
```

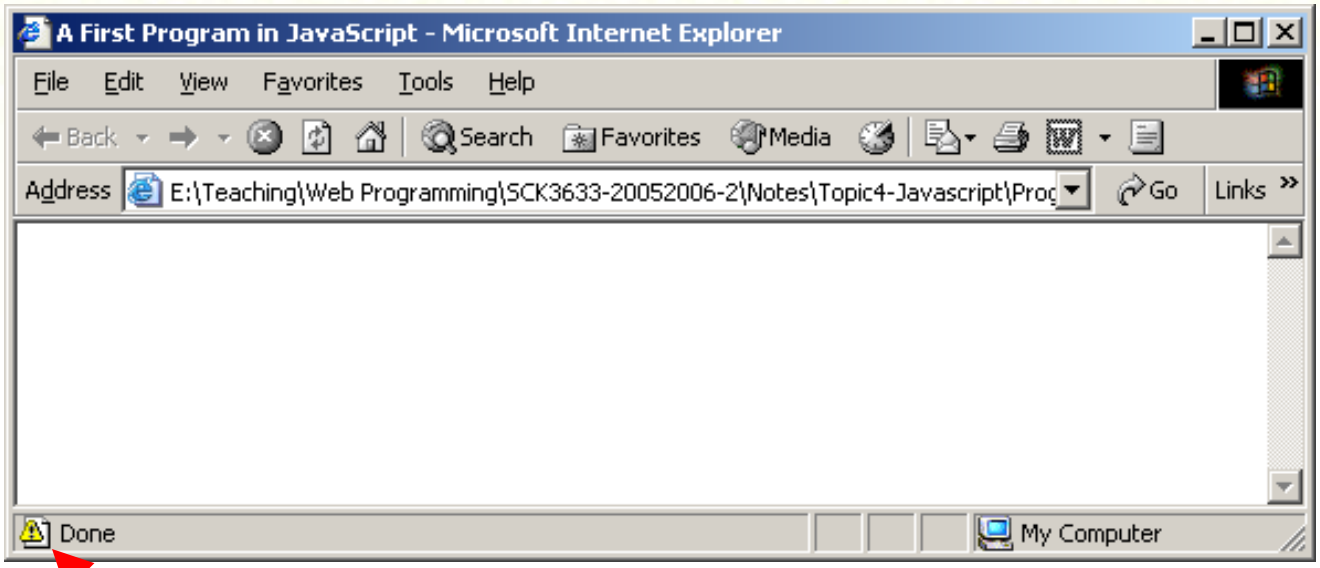
Simple Program

Internal script:

```
<html>
 <head>
     <title>A First Program in JavaScript</title>
     <script type = "text/javascript">
       <!--
         document.writeln(
            "<h1>Welcome to JavaScript Programming!</h1>" );
       // -->
     </script>
  </head>
  <body></body>
</html>
```

Output:

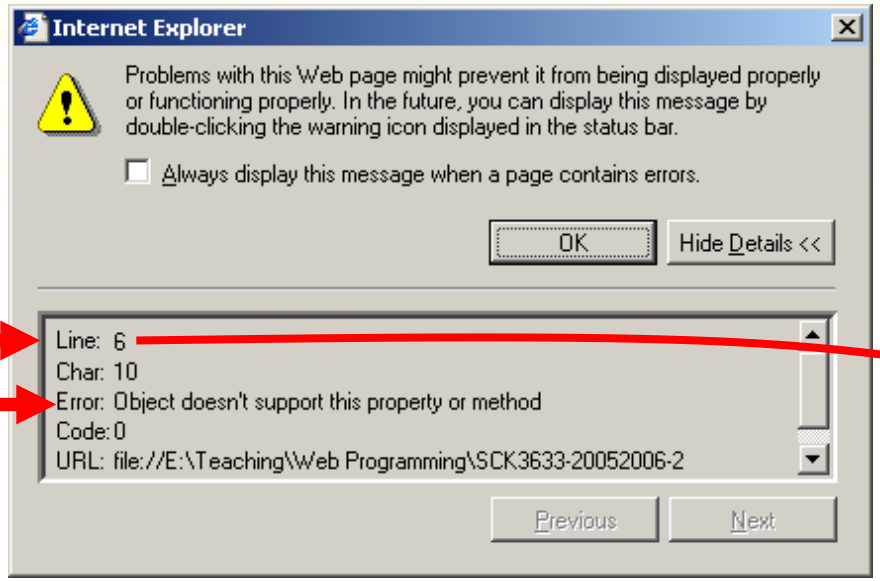
Debugging Errors:

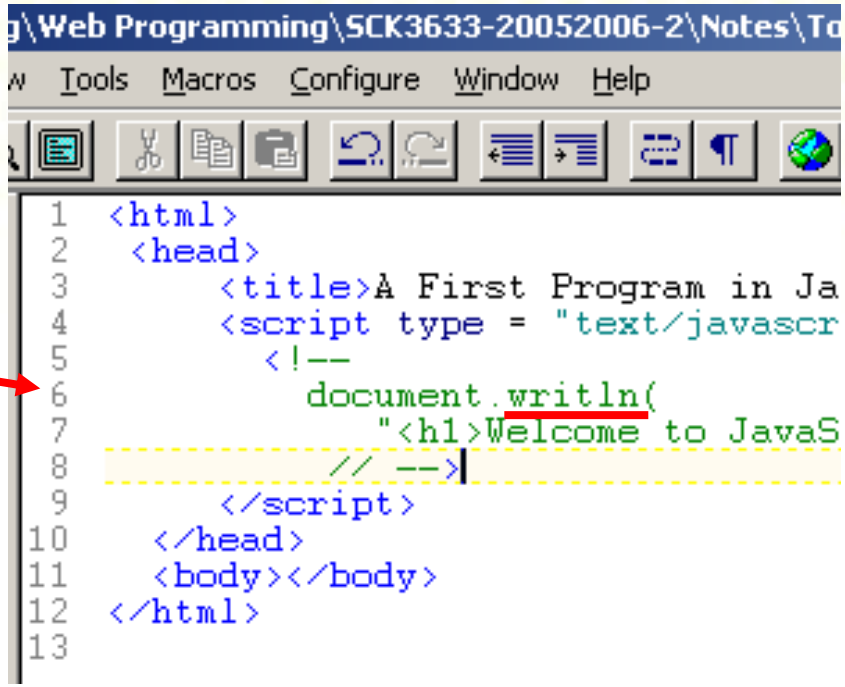


Click this.

Warning icon means there are some errors in your script

Debugging Errors:





Simple Program

External script:

HTML document (welcome.html)

```
<html>
 <head>
     <title>A First Program in JavaScript</title>
     <script type = "text/javascript" src = "welcome.js">
     </script>
  </head>
  <body></body>
</html>
```

External Javascript file (welcome.js)

```
document.writeln("<h1>Welcome to JavaScript Programming!</h1>" );
```

Output:

Simple Programs

Example 2: *displaying simple message box using alert dialog*

```
<html>
   <head>
      <title>Printing Multiple Lines in a Dialog Box</title>
      <script type = "text/javascript">
         <!--
            window.alert( "Welcome to\nJavaScript\nProgramming!" );
         // -->
      </script>
   </head>

   <body>
      <p>Click Refresh
   </body>
</html>
```

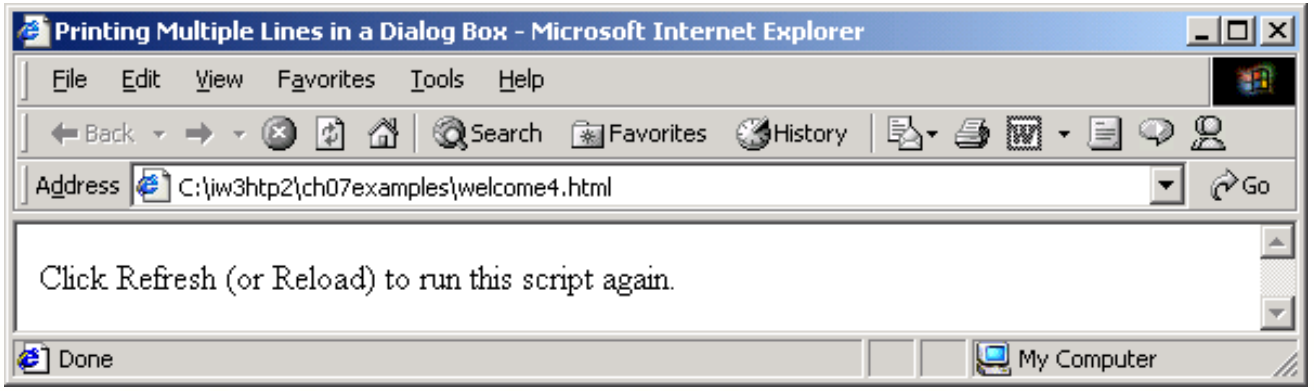
The **window** method **alert** displays an alert dialog to the user.

When the alert dialog displays, the string passed as its one argument is displayed.

The escape sequence **\n** is the newline character that places all remaining text on the next line.

Simple Programs

Output:



Escape sequence	Description
`\n`	Newline. Position the screen cursor at the beginning of the next line.
`\t`	Horizontal tab. Move the screen cursor to the next tab stop.
`\r`	Carriage return. Position the screen cursor to the beginning of the current line; do not advance to the next line. Any characters output after the carriage return overwrite the characters previously output on that line.
`\\`	Backslash. Used to represent a backslash character in a string.
`\"`	Double quote. Used to represent a double quote character in a string contained in double quotes. For example, `window.alert( "\"in quotes\"" );` displays **"in quotes"** in an **alert** dialog.
`\'`	Single quote. Used to represent a single quote character in a string. For example, `window.alert( '\'in quotes\'' );` displays **'in quotes'** in an **alert** dialog.

Fig. 7.5 Some common escape sequences.

Simple Programs

Example 3: *getting user input using prompt dialog*

```
<html>
   <head>
      <title>Using Prompt Box</title>
      <script type = "text/javascript">
        <!--
         var name; // string entered by the user

         // read the name from the prompt box as a string
         name = window.prompt ("Please enter your name", "Ali");
         document.writeln("<h1>Hello, " + name +
                  ", welcome to JavaScript Programming!</h1>" );
           // -->
      </script>
  </head>

  <body>
    <p>Click Refresh (or Reload) to run this script again.</p>
  </body>
</html>
```

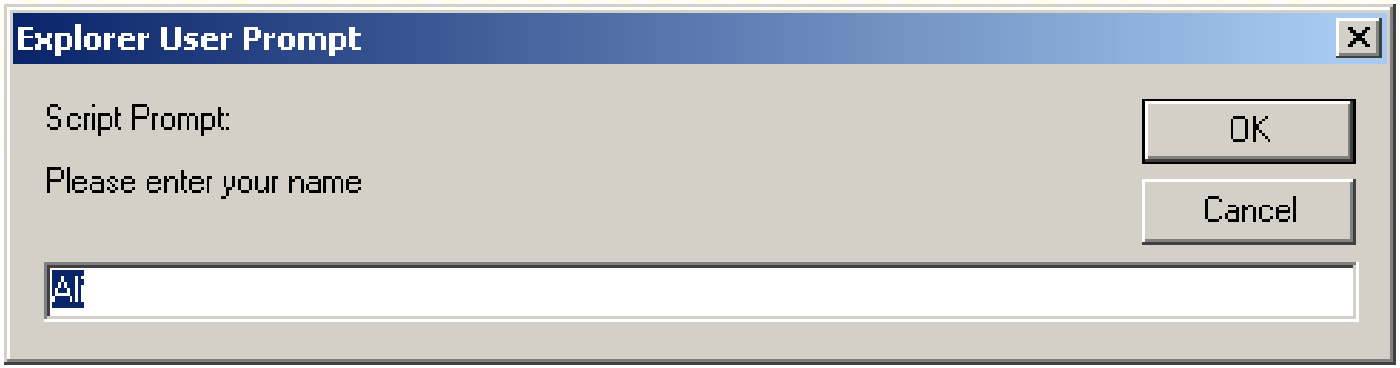
This string is used as a prompt message

default value

The **window** method **prompt** displays an input dialog to the user.

Simple Program

Output:



Keywords

JavaScript Keywords				
break	case	continue	delete	do
else	false	for	function	if
in	new	null	return	switch
this	true	typeof	var	void
while	with			
Keywords that are reserved, but not used by JavaScript				
catch	class	const	debugger	default
enum	export	extends	finally	import
super	try			

Fig. 8.2 JavaScript keywords.

Operators

Arithmetic operators:

JavaScript operation	Arithmetic operator	Algebraic expression	JavaScript expression
Addition	+	$f + 7$	`f + 7`
Subtraction	-	$p - c$	`p - c`
Multiplication	*	bm	`b * m`
Division	/	x / y or <Anchor9> or $x \quad y$	`x / y`
Modulus	%	$r \bmod s$	`r % s`

Fig. 7.11 Arithmetic operators.

Relational operators:

Standard algebraic equality operator or relational operator	JavaScript equality or relational operator	Sample JavaScript condition	Meaning of JavaScript condition
Equality operators			
=	==	`x == y`	`x` is equal to `y`
	!=	`x != y`	`x` is not equal to `y`
Relational operators			
>	>	`x > y`	`x` is greater than `y`
<	<	`x < y`	`x` is less than `y`
	>=	`x >= y`	`x` is greater than or equal to `y`
£	<=	`x <= y`	`x` is less than or equal to `y`

Fig. 7.14 Equality and relational operators.

Operators

Precedence and Associativity:

Operators	Associativity	Type
( )	left to right	parentheses
* / %	left to right	multiplicative
+ -	left to right	additive
< <= > >=	left to right	relational
== !=	left to right	equality
=	right to left	assignment

Fig. 7.16 Precedence and associativity of the operators discussed so far.

Variables

Declaring and assigning variables:

```
var variable1;          // example 1 - declaring a variable without any value
var variable2 = 100;    // example 2 - declaring and assigning a variable
variable3 = 3.823;      // example 3 - assigning without declaring first
```

Data type will affect the result of an operation:

Example:

```
num1 = 7;  // an integer number
num2 = 2.0; // a real number
num3 = 2;   // an integer number
ch   = '2'; // a character

result = num1/num2;   // result=3.5
result = num1/num3;   // result=3.5, do not be confused with C
result = num1 + num2; // result=9
result = num1 + ch;   // result='72';
result = ch + num1;   // result='2';
```

Selections

if statement

- Three forms of `if` statements are shown at the next table.
- The *condition* is always in parentheses
- All TRUE-PARTS and all FALSE-PARTS are a single statement or a block of statements (also called compound statement)

```
if(condition)
   statement;
```

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

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

if statement

Examples:

```
if ( attendance < 0.8 )
{
  document.write("FAIL");
}
```

```
if (score > 50)
  document.write("FAIL");
else
   document.write("FAIL");
```

```
if (score > 90)
       document.write("Grade A");
else if (score > 75)
       document.write("Grade B)";
else if (score > 60)
       document.write("Grade C");
else if (score > 50)
       document.write("Grade D");
else
      document.write("Grade F");
```

switch statement

```
switch (expression)
{
 case value1: statements_1;
              break;

 case value2 : statements_2;
              break;
  ...

 default : statements;
           break;
}
```

How the `switch` statement works?

1. Check the value of expression.

2. Is it equal to value1?
   - If yes, execute the statements_1 and break out of the switch.
   - If no, is it equal to value2? etc.

3. If it is not equal to any values of the above, execute the default statements and then break out of the switch.

switch statement

Example:

```
var value = 2;

switch (value)
{
 case 1: document.write("One");
         break;

 case 2: document.write("Two");
         break;

 default : document.write("Neither One nor Two");
         break;
}
```

this expression evaluates to 2

it is not equal to this case-value (i.e. 2!=1). So, skip the statements of case 1 and move to the next case.

it is equal to this case-value (i.e. 2==2). So, execute the statements of the 'case 2'.

Prints Two

break out of the switch

Output:

Two

Repetitions

for loops

```
for (initialization; condition; update)
{
      statements;
}
```

Example: Prints odd numbers between 0 -10.

```
for (var n=1; n<10; n +=2)
{
      document.write(n + " <br>");
}
```

```
Output:

1
3
5
7
9
```

while loops

```
while (condition)
{
    statements;
}
```

Example: Prints odd numbers between 0 -10.

```
var n=1;

while (n<10)
{
  document.write(n + " <br>");
  n +=2;
}
```

```
Output:

1
3
5
7
9
```

do-while loops

```
do
{
   statements;
} while (condition)
```

Example: Prints odd numbers from 1 – 9.

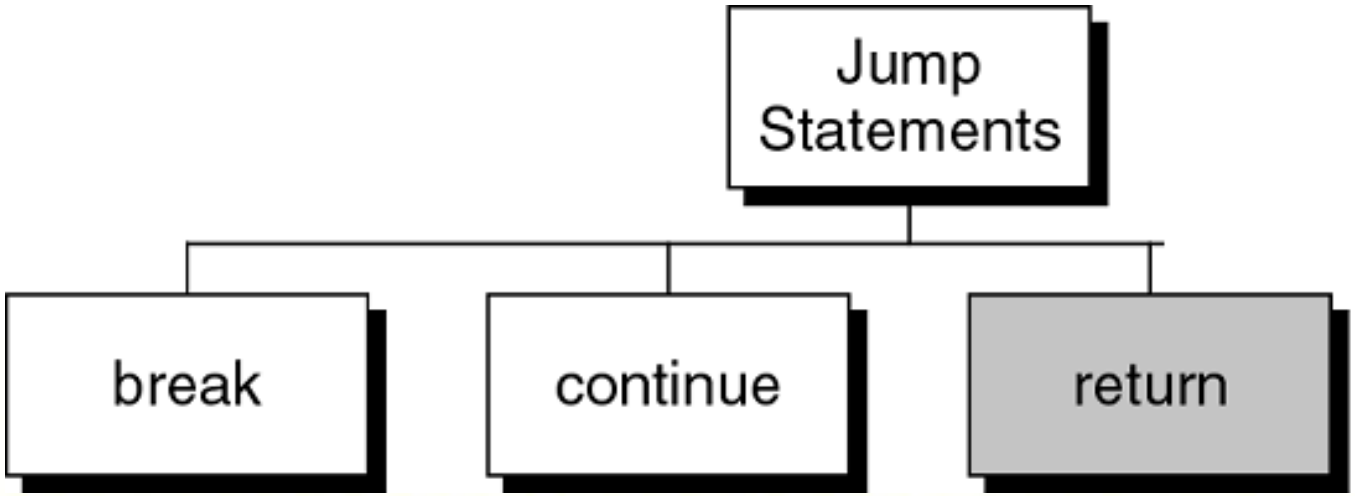
```
var n=1;

do
{
   document.write(n + " <br>");
   n +=2;
} while (n<10)
```

```
Output:

   1
   3
   5
   7
   9
```

Jump Statements

Jump statements

- Repetition of a loop is controlled by the loop condition.
- We can alter the flow of control by using using jump statements.
- Javascript provides three jump statements:



break statement

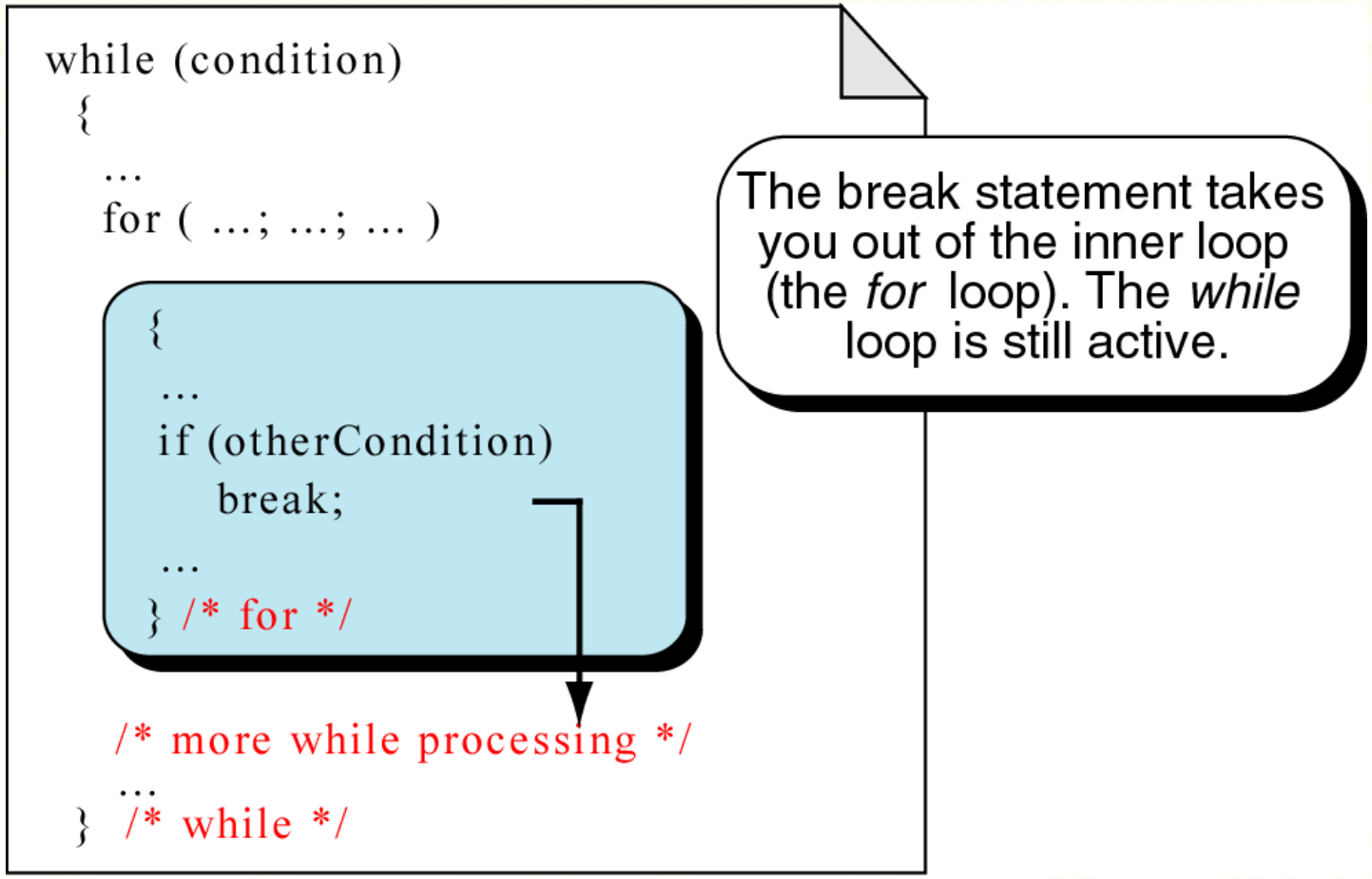
- It causes a loop to terminate

Example:

```
for (n=10; n>0; n=n-1)
{
 if (n<8) break;
 document.write(n, " ");
}
```

```
Output:


10 9 8
```

`break` statement



`break` statement with label

Example: with break statement

```
outer: { // labeled block - outer block
    for (var i=5; i>0; i--)
    { // inner block
      for (var j=0; j<i; j++)
      {
       if (j==2) break;
       if (i==3) break outer;
       document.writeln(i," ");
      }
      document.writeln("<br>");
   }
}
```

This break (without label) terminates the inner for loop.

This break (with label) terminates the outer for loop

Output:

5 5
4 4

`continue` statement

- In `while` and `do…while` loops, the `continue` statement transfers the flow of control to the loop condition.
- In `for` loop, the `continue` statement transfers the flow of control to the updating part.

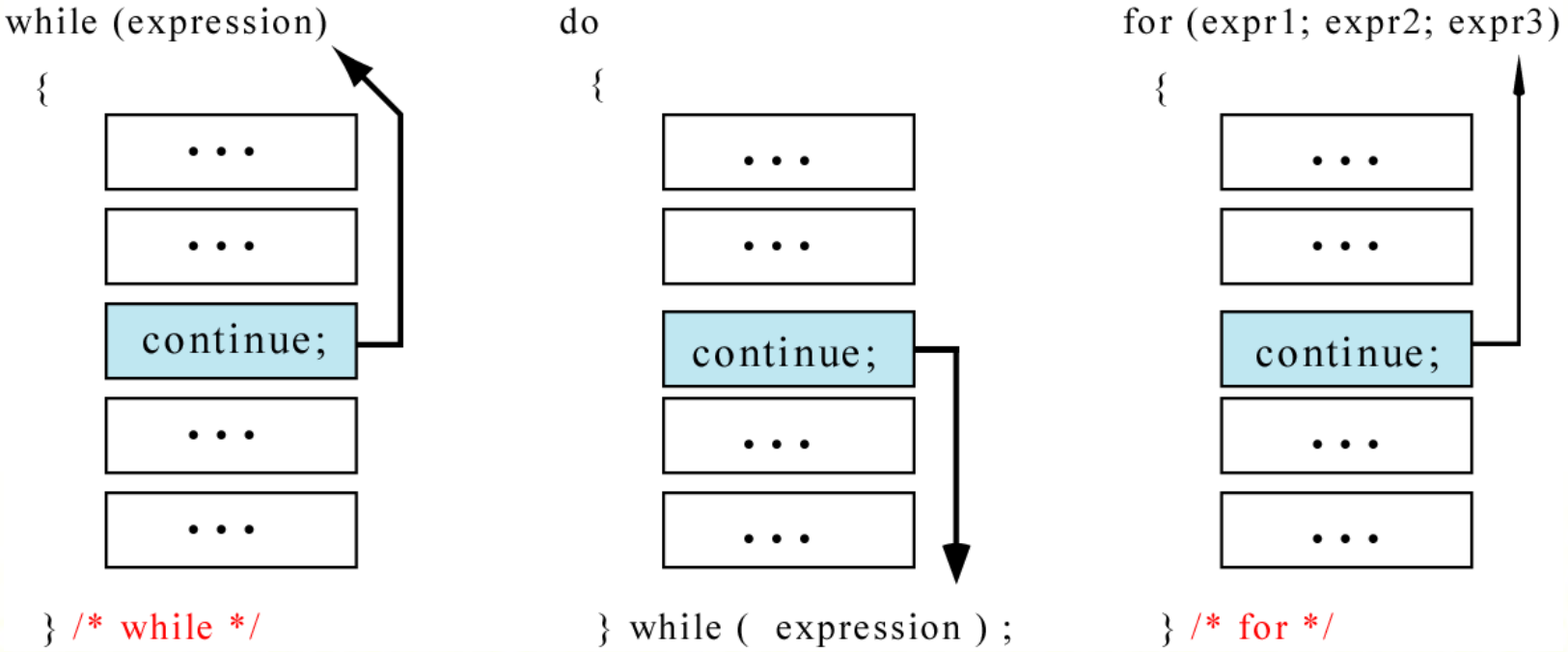


Figure 6-22: The *continue* statement

`continue` statement

Example:

```
for (n=10; n>0; n=n-1)
{
 if (n%2==1) continue;
 document.write(n," ");
}
```

```
Output:


10 8 6 4 2
```

`continue` statement with label

Example: with continue statement

```
outer: { // labeled loop - outer loop
    for (var i=5; i>0; i--)
    { // inner block
      for (var j=0; j<i; j++)
      {
       if (j%2 == 1) continue;
       if (i%2 == 0) continue outer;
       document.writeln(i," ");
      }
      document.writeln("<br>");
   }
}
```

This continue (without label) skips the remaining statements in the same loop and goes to the updating part (j++)

This continue (with label) skips the remaining statements and goes to the updating part of the outer loop (i--)

Output:

```
0 2 4
0 2
0
```

`return` statement

- It causes a function to terminate.

Example:

```
function print_numbers()
{ var n=10;
  var i;

  while (n>0)
  {
     for (i=n;i>0; i--)
     {
       if (i%2==1) continue;

       if (i%4==0) break;

       if (n==6) return;

       document.write(i, " ");
     }
     document.writeln("<br>");
     n=n-1;
   }
}
```

The continue statement transfers control to the updating part (i--)

The break statement terminates the for loop.

The return statement terminates the function and returns to the caller.

```
Output:

10



6
```

`return` statement

- When to use `return`?
- *Example:* the following functions are equivalent

```
function calc_point(grade)
{
 var result;

 if (grade=='A') result = 4.0;
 else if (grade=='B') result = 3.0;
 else if (grade=='C') result = 2.5;
 else if (grade=='D') result = 2.0;
 else result = 0.0;

 return result;
}
```

```
function calc_point(grade)
{
 if (grade=='A') return 4.0;
 if (grade=='B') return 3.0;
 if (grade=='C') return 2.5;
 if (grade=='D') return 2.0;
 return 0.0;
}
```

The *else* part of each *if* statement may be omitted. It has never been reached.

return statement

```
function calc_point3(grade)
{
 var result;

 switch (grade)
 {
   case 'A': result = 4.0;
             break;

   case 'B': result = 3.0;
             break;

   case 'C': result = 2.5;
             break;

   case 'D': result = 2.0;
             break;

   default:  result =0.0;
 }

 return result;
}
```

```
function calc_point4(grade)
{
 switch (grade)
 {
   case 'A': return 4.0;
   case 'B': return 3.0;
   case 'C': return 2.5;
   case 'D': return 2.0;
 }
 return 0.0;
}
```

The *break* statement of each *case* may be omitted. It has never been reached.

Functions

Creating functions

Involves two steps:

- Define: *to define what processes should be taken*
- Call/Invoke: *to execute the functions*

Syntax of function definition:

```
function function_name (param1, param2, .., param_n)
                                        //parameters are optional
{
    //function's code goes here

    return value_or_object; //optional
}
```

Creating functions (cont.)

Example:

```
<html>
<head>
     <title> simple function </title>
     <script language="Javascript">

          //function definition
          function hello()
          { alert("Hello World");
          }
      </script>
</head>

<body onLoad="javascript:hello();"> <!-- function invocation-->

</body>

</html>
```

Arrays

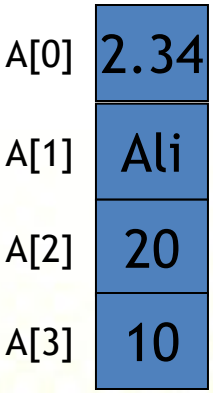
Creating arrays

```
var a = new Array(12);// create 12-element array
var b = new Array();   // create an empty array

var c = new Array(12,10,11); // create 3-element array
                  // and initialize its elements with specified values

var d = [12,10,11]; // same as array 'c'

var e = [1,,,10]; // only the first and last elements are
                  // initilized
```

Inserting values into array

```
var A =new Array();
A.push(10);
A.push(20);
A.push("Ali");
A.push(2.34);
```

Result:



Inserting values into array (cont.)

```
var A =new Array();
A[0]= 10;
A[1]= 20;
A[2]="Ali";
A[3]=2.34;
```

Result:

A[0]	10
A[1]	20
A[2]	Ali
A[3]	2.34

```
// the following is better and more flexible
var A =new Array();
A[0]= 10;
A[A.length]= 20;
A[A.length]="Ali";
A[A.length]=2.34;
```

Traversing the elements of array

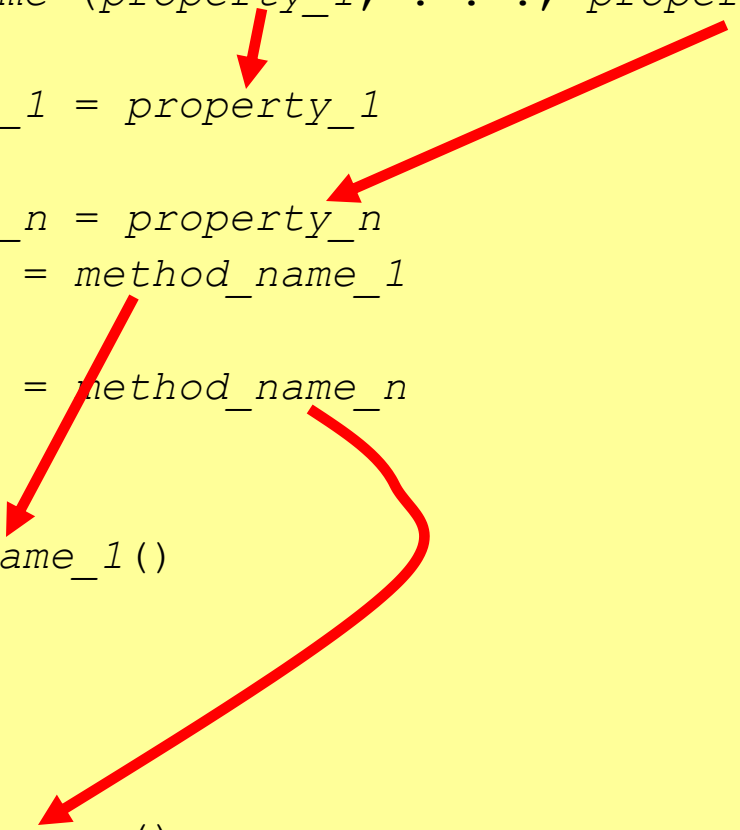
```
// Example: summing up the elements of array A
sum =0;
for (var i=0; i<A.length; i++)
   sum += A[i];
```

```
// Another way, using for-in loop
sum =0;
for (var i in A)
   sum += A[i];
```

Objects

Creating classes

```
function class_name (property_1, . . ., property_n)
{
    this.property_1 = property_1
    . . .
    this.property_n = property_n
    this.method_1 = method_name_1
    . . .
    this.method_n = method_name_n
}

function method_name_1()
{

}

function method_name_n()
{
    . . .
}
```



Creating classes (cont.)

Example:

```
function Person(aName,age)
{
    this.name = aName;
    this.age  = age;
    this.displayInfo = print;
}

function print()
{
  window.alert("Name= "   + this.name +
               "\nAge= " + this.age);
}
```

Creating object and accessing its members

```
object_name = new class_name (property_1, property_2, …..);
```

Example:

```
// creating an object of class Person
person1 = new Person("Ali",20);

// displaying info of person1
person1.displayInfo();

// changing property
person1.age=23;
```

Array of objects

Example:

```
var person_list = new Array(); // creating array
// inserting objects into the array elements
person_list[0]= new Person("Ali",20);
person_list[1]= new Person("Aminah",24);
person_list[2]= new Person("Bakar",19);
```

```
// displaying info of all persons
for (var i=0; i<person_list.length; i++)
    person_list[i].displayInfo();
```

```
// calculating the average age of all persons
sum=0;
for (var i=0; i<person_list.length; i++)
    sum += person_list[i].age;
average = sum / person_list.length;
```

Reference

- Sebesta, R. W., Programming the World Wide Web, (2009), 5th Edition, Pearson.
- Deitel P. J, Deitel H. M., Internet & World Wide Web: How To Program, (2007), 5th Edition, Prentice Hall.
- Anderson-Freed S., (2001), Weaving A Website: Programming in HTML, JavaScript, PHP and Java. Prentice Hall