

<script>



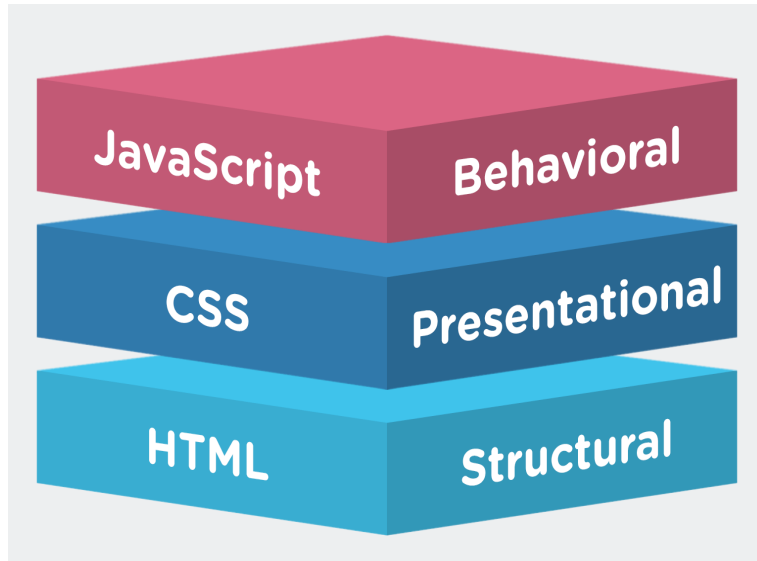
JavaScript

</script>

Client Side technologies

Introduction to JavaScript

JavaScript...



▪ HTML

- Hypertext Markup Language
- Structure of Page



▪ JavaScript

- Interactivity with User
- Dynamic Updates in a Web Page



▪ CSS

- Cascading Style Sheets
- Presentation/Styling



JavaScript...



**JavaScript is a web client side
scripting language.**

Web programming



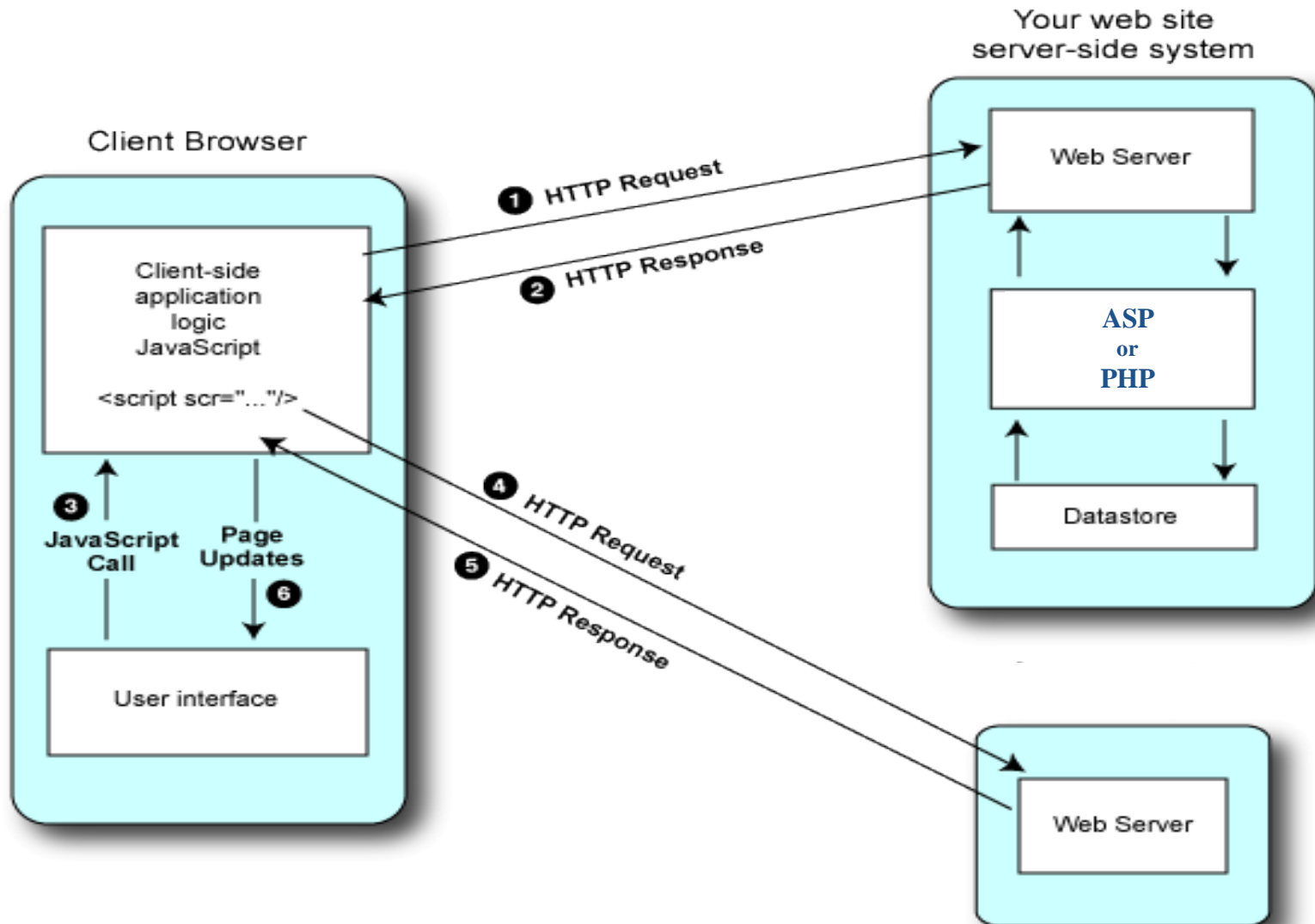
- ❑ **Programming for the World Wide Web involves both:**
 - server-side programming.
 - client-side (browser-side) programming.

Web programming (Cont.)



- ☐ Servers make web documents, which are specified in HTML, available on request to browsers.
- ☐ Browsers display, to users, web documents which have been received from servers.
- ☐ Client side Scripting language is used to write interactive web documents that can be displayed to user in browser when received from servers.

Web programming (Cont.)



HTML



- ❑ **HTML is the markup language used for specifying WWW documents.**
- ❑ **HTML is used for creating static web pages.**
 - i.e. can view or print (no interaction).
- ❑ **HTML's role on the web is to tell the browser how a document should appear.**
- ❑ **HTML was the only language one could use to create Web pages.**

JavaScript



- ❑ JavaScript (JS) is a **simple, flexible, lightweight, interpreted, programming language** with first-class functions. While it is most well-known as the **scripting language for Web pages**.
- ❑ Designed to add **interactivity** to HTML pages.
- ❑ Designed to create dynamic web sites.
 - i.e. Change contents of document, provide forms and controls, animation, control web browser window, etc.
- ❑ JavaScript statements embedded in an HTML page can recognize and respond to User Events.
- ❑ You can use JavaScript without buying a license.
- ❑ You only need a web browser & a text editor.
- ❑ JavaScript is an **object-Based language** (or prototype-based), and we can consider it as **object-oriented language**, but is **not a class-based object-oriented language** like Java, C++, C#.
- ❑ Class-based OOP languages are a subset of the larger family of OOP languages which also include prototype-based languages like JavaScript.
- ❑ Related to Java in name only (Name was part of a marketing deal).

Client Side Scripting Languages



- JavaScript (1.x)
- VBScript
- Jscript (Microsoft standardized version of JavaScript)
- TypeScript (New scripting language developed by Microsoft and based on JavaScript).
- Action script (Used with Flash)

Markup vs. Scripting vs. Programming Languages



Markup Language	Scripting Language	Programming Language
A text-formatting language designed to transform raw text into structured documents, by inserting procedural and descriptive markup into the raw text	Interpreted command by command, and remain in their original form.	Compiled, converted permanently into binary executable files (i.e., zeros and ones) before they are run.
	Output isn't a standalone program or application, it runs inside another program	Produce a standalone program or application
Very simple to learn and use.	Simple, but has some programming logic	More complicated and has advanced logic and structure.
Example: HTML, XHTML.	Example: JavaScript, VBScript and Action Script	Example: C,C++, Java

JavaScript vs. Java



JavaScript	Java
Interpreted (not compiled) by client.	Compiled on server before execution on client.
Object-based. Code uses built-in, extensible objects, and you can also make classes and inheritance.	Object-oriented. Consist of object classes with inheritance.
Cannot stand alone. Embedded in and requires HTML (Standalone in some cases like: NodeJS).	Stands alone. Java is a complete development environment.
Developed by Netscape.	Developed by Sun Microsystems.
Loose typing: Variable data types not declared.	Strong typing: Variable data types must be declared.

JavaScript History



- ❑ Developed by **Brendan Eich** at **Netscape** in 1995 under the name of **LiveScript**.
- ❑ In Navigator 2.0, name changed to **JavaScript** as a result of an agreement with Sun, the developer of Java.
- ❑ Netscape introduced an implementation of the language for **server-side scripting** with Netscape Enterprise Server in December 1995, soon after releasing JavaScript for browsers. Since the mid-2000s, there has been a resurgence of **server-side JavaScript implementations, such as Node.js**
- ❑ In November 1996, Netscape submitted JavaScript to **ECMA International** to carve out a standard specification, which other browser vendors could then implement based on the work done at Netscape.
- ❑ In 1997, **ECMAScript (Official name of JavaScript)** was introduced by **ECMA International** as an attempt at standardization.
- ❑ In 1997, the standard approved as an ISO standard.
- ❑ Microsoft recognized the importance of JavaScript and entered the market with two creations, **JScript** and **VBscript**.
- ❑ JavaScript isn't W3C standard till now.

JavaScript History (Cont.)



- ❑ The standard for JavaScript is **ECMAScript**. As of **2012**, all modern browsers fully support **ECMAScript 5.1**
- ❑ On June 17, **2015**, ECMA International published the sixth major version of ECMAScript, which is officially called **ECMAScript 2015**, and is more commonly referred to as **ECMAScript 6** or **ES6**.
- ❑ Since then ECMAScript standards are on yearly release cycles. The latest draft version is currently ECMAScript 2017.

Server-side JavaScript...



Using NodeJS (server-side JavaScript environment) we can create Server-side, standalone diverse variety of tools and applications.

JavaScript characteristics



- ☐ **Case sensitive**
- ☐ **Object-oriented**
- ☐ **Event-Driven**
- ☐ **Interpreted language**
- ☐ **Browser-Dependent**
- ☐ **Platform independent**
- ☐ **Dynamic**

What JavaScript can do?



- ☐ Giving the user more control over the browser.
- ☐ Detecting the user's browser, OS, screen size, etc.
- ☐ Performing simple computations on the client side.
- ☐ Validating the user's input. It validates the data on the user's machine before it is forwarded to the server.
- ☐ Can handle events.
- ☐ Add dynamic text into an HTML page
 - i.e. open and close new browser windows and you can control the appearance of the new windows you create. You can control their size, their location, and the toolbars they have available.
- ☐ Can create cookies.

What JavaScript can't do?



- ☐ **Directly access files on the user's system or the client-side LAN ; the only exception is the access to the browser's cookie files.**
- ☐ **Directly access files on the Web server.**

JavaScript Strength & weakness



Strength	Weakness
Quick Development	Limited Range of Built-in Methods
Easy to Learn	No Code Hiding
Platform Independence	Browser Dependant
Small Overhead	Altering the text on an HTML page will reload the entire document

How to embed JavaScript code?



❑ We can Write JavaScript:

1. Anywhere in the html file between `<script>` `</script>` tags.

```
<html>
  <head>
    <title>A Simple Document</title>
    <script >
      document.write ("Hello world");
    </ script >
  </head>
  <body>
    <p>Page content</p>
    < script >
      document.write (" welcome to JavaScript world");
    </ script >
  </body>
</html>
```

How to embed JavaScript code? (Cont.)

A circular icon with a blue border containing a white document with the letters 'JS' in purple.

2. As the value of the event handler attributes.

```
<html>
  <head>
    <title>A Simple Document</title>
  </head>
  <body>
    We can write it at the event handlers
    <input type="button" value="Click Me" onClick="alert('Hello')"/>
  </body>
</html>
```

How to embed JavaScript code? (Cont.)

A circular icon with a blue border containing a white document with the letters 'JS' in purple.

3. In an external file and refer to it using the src attribute.

myWebPage.html

```
<HEAD>
  <TITLE>A Simple Document</TITLE>
  <script src= "myJSFile.js" > </script>
</HEAD>

<BODY>
  We can refer to JavaScript in another file.
  < script >
    dosomething();
  </ script >

</BODY>
```

myJSFile.js

```
function dosomething()
{
  alert ("Hello ");
}
.
.
.
.
```

Outputting text with JavaScript



❑ `document.write()`

❑ **Example:**

```
<script>  
    document.write("Hello There!");  
</script>
```

Let's Try it....

Variables



- Naming
 - In JavaScript, the first character must be a letter, an underscore (_), or a dollar sign (\$).
 - Subsequent characters may be letters, digits, underscores, or dollar signs.
 - Don't use spaces inside names. FirstName NOT First Name.
 - Avoid reserved words, words that are used for other purposes in JavaScript.
 - Case-sensitive - FirstName different than firstName.
 - Variables should describe what they are.
 - JavaScript programmers tend to use camel case that starts with a lowercase letter:
firstName, lastName, masterCard, interCity.
- Types (specified at the runtime based on the variable value):
 - String.
 - Numeric.
 - Boolean (true or false).
 - null (special keyword, that is treated as an "empty" variable).
 - Undefined (A special keyword means that a value hasn't even been assigned yet).

Variables (cont.)



❑ Declaration:

- use keyword **var** to declare a variable.
- Variables are case sensitive.
- Variables are **loosly typed**, initial value is **undefined**.

Example:

```
var count;  
alert (count); //undefined  
count=3;  
//or  
var count=3;  
// Case Sensitive  
var Count=3; //Is a new variable
```

- If you re-declare a JavaScript variable, it will not lose its value.

Example:

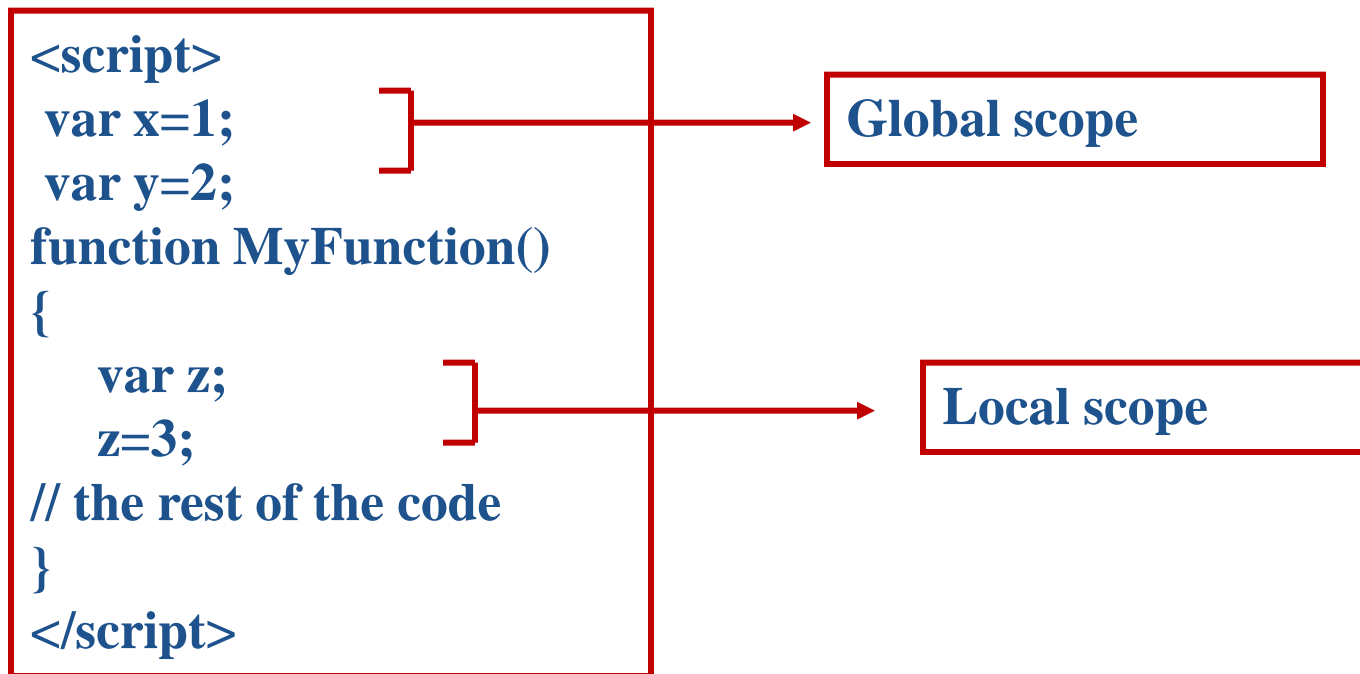
```
var carName = "Volvo";  
var carName;  
alert (carName); //volvo
```


Variables – Life time & scope



□ Lifetime & Scope:

- Local Scope
- Global Scope



Variables – Life time & scope (Cont.)



□ Lifetime & Scope (cont.):

- Variables declared:
 - inside a function are local variable
 - outside any function are global variable (available to any other code in the current document).
- All variables and methods are in the public global scope, they are properties of the global object “window”
- You can access global variable in a function using:
`window.varName;`
- When you declare a variable without **var**, it'll be considered **global variable**!

Example:

```
function foo() {  
    var x;  
    x = 5;  
    y = 6; //will consider it global variable, as if you write: window.y = 6;  
    return x + y;}  

```

Hoisting



❑ Variables:

- Hoisting is JavaScript's default behavior of **moving all declarations to the top of the current scope** (to the top of the current script or the current function).
- Variables declared with **let and const** aren't hoisted.
- We can refer to a variable declared later without getting any exception or error (a variable can be used before it has been declared).

```
var y;  
y=10; // Assign 10 to y, that's declared before.  
x = 5; // Assign 5 to x, that's defined later, because it was hoisted.  
document.write(x,y); //5,10  
var x; // Declare x (Hoisted). The same as if declared it at code beginning with x.
```

- JavaScript **only hoists declarations, not initializations..**

```
var x = 5; //x hoisted, but initialization not hoisted.  
document.write(x,y); // 5, undefined  
var y = 7; // Initialize y (y is hoisted, but initialization is not hoisted)
```

- It's recommended to declare all variables at the beginning of every scope to avoid any unwanted bugs.

❑ Functions

- Function statements are hoisted too.
- Functions are available even before its declaration

Strict mode



- ❑ Strict mode is declared by adding **"use strict";** to the beginning of a script or a function.
 - Declared at the beginning of a script, it has global scope (all code in the script will execute in strict mode).
 - Declared inside a function, it has local scope (only the code inside the function is in strict mode).

❑ Why Strict Mode?

- Strict mode makes it easier to write "secure" JavaScript.
- Strict mode changes previously accepted "bad syntax" into real errors.

- ❑ JavaScript in strict mode does not allow variables to be used if they are not declared.

```
"use strict";
```

```
x = 3.14;           // This will cause a syntax error
```

```
y = 10;             // This will not cause an error, as y is hoisted.
```

```
var y;
```

Strict mode (Cont.)



Variables – block scope with let



❑ Block variable declaration: let (New ES6 feature):

- There was no Block Scope before ES6, only function scope, let declaration introduced in ES6 allowing block scope
- Variables declared by **let** have as **their scope the block in which they are defined**, as well as in **any contained sub-blocks**.
- let variables are block-scoped. **The scope of a variable declared with let is just the enclosing block, not the whole enclosing function.**

```
1  function varTest() {  
2      var x = 1;  
3      if (true) {  
4          var x = 2; // same variable!  
5          console.log(x); // 2  
6      }  
7      console.log(x); // 2  
8  }  
9  
10 function letTest() {  
11     let x = 1;  
12     if (true) {  
13         let x = 2; // different variable  
14         console.log(x); // 2  
15     }  
16     console.log(x); // 1  
17 }
```

Variables – block scope with let (Cont.)



❑ Block variable declaration: let (Cont.):

- Loops of the form for (let x...) create a fresh binding for x in each iteration, and the scope of the variable will be inside the for loop only.

```
function test(){
    .....
    for (let i = 0; i < messages.length; i++) {
        ... //let scope inside loop only, not whole function.
    }
}
```

- Global let variables **are not properties on the global object**. That is, you won't access them by writing `window.variableName`. Instead, they live in the scope of an invisible block that notionally encloses all JS code that runs in a web page.
- It's an error to try to use a let variable before its declaration is reached (**as variables declared using let aren't hoisted**).

```
function update() {
    document.write("your name:", t); // ReferenceError
    ...
    let t = "test";}
```

Variables - Constants



❑ JavaScript Constants (new ES6 Feature):

- Variables declared with **const** are constant variables, you can't assign to them, except at the point where they're declared.

```
const MAX_CAT_SIZE_KG = 3000;
```

```
MAX_CAT_SIZE_KG = 5000; // SyntaxError
```

```
MAX_CAT_SIZE_KG++; // SyntaxError
```

```
const theFairest; // SyntaxError, you can't declare const variable without assigning it a value
```

- A constant can be global or local to a function where it is declared.
- Constants also share a feature with variables declared using **let** in that they are **block-scoped instead of function-scoped** (and thus they are not hoisted)

JavaScript Guidelines



1. JavaScript is Case Sensitive.
→ myVar is different than myvar
2. JavaScript ignores extra spaces.
→ name="Ali" is the same as name = "Ali"
3. Multiple line comments preceded by /* and ended by */
→ /* This is a comment block.
It contains several lines */
document.write("Hello World!")
4. Single line Comment are preceded by a double-slash (//).
→ //this is one line comment
document.write("Hello World!")
5. JavaScript uses the Unicode character set. Unicode covers (almost) all the characters, punctuations, and symbols in the world.
6. Special characters are preceded by \
→ document.write ("You \\ me are singing!")

JavaScript Guidelines (Cont.)



❑ JavaScript special characters:

Character	Meaning
<code>\b</code>	Backspace
<code>\f</code>	Form feed
<code>\t</code>	Horizontal tab
<code>\n</code>	New line
<code>\r</code>	Carriage return
<code>\\</code>	Backslash
<code>\'</code>	Single quote
<code>\"</code>	Double quote

Operators



❑ JavaScript supports:

1- Unary operators:

Requires one operand such as `x++`

2-Binary operators:

Require two operands in the expression such as `x+2`

3- Ternary operators:

Requires three operands such as Conditional (`? :`) operator.

Operators (Cont.)



❑ Arithmetic Operators:

(y=5)

Operator	Description	Example	Result
+	Addition	$x=y+2$	$x=7$
-	Subtraction	$x=y-2$	$x=3$
*	Multiplication	$x=y*2$	$x=10$
/	Division	$x=y/2$	$x=2.5$
%	Modulus (division remainder)	$x=y\%2$	$x=1$
++	Increment	$x=++y$	$x=6$
--	Decrement	$x=--y$	$x=4$

Operators (Cont.)



❑ Assignment Operators:

(x=10, y=5)

Operator	Example	Same As	Result
=	x=y		x=5
+=	x+=y	x=x+y	x=15
-=	x-=y	x=x-y	x=5
=	x=y	x=x*y	x=50
/=	x/=y	x=x/y	x=2
%=	x%=y	x=x%y	x=0

Operators (Cont.)



❑ Comparison Operators:

Operator	Description
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equality
!=	Inequality
===	Strict Equality
!==	Strict Inequality

Operators (Cont.)



❑ Bitwise Operators:

Operator	Description
&	Bitwise AND
 	Bitwise OR
^	Bitwise XOR
~	Bitwise NOT
<<	Bitwise Left Shift
>>	Bitwise Right Shift
>>>	Unsigned Right Shift

Operators (Cont.)



❑ Logical Operators:

Operator	Description
&&	Logical "AND" – returns true when both operands are true; otherwise it returns false
 	Logical "OR" – returns true if either operand is true. It only returns false when both operands are false
!	Logical "NOT"—returns true if the operand is false and false if the operand is true. This is a unary operator and precedes the operand

Operators (Cont.)



❑ String concatenation:

- + operator , used in sting concatenation.
- Example:

```
<script>  
    var A="Welcome "  
    var B="Ahmed"  
    var C=A+B  
    document.write(c)  
    // the result will be "WelcomeAhmed"  
</script>
```

Operators (Cont.)



❑ Special Operators:

- **Conditional Operator: (Condition)?if true:if false**
- **Example:**

```
<script>
```

```
var temp=120;  
x=(temp>100) ? "red" : "blue";  
// the value of x will be "red"
```

```
var temp=20;  
y=(temp>100) ? "red" : "blue";  
// the value of y will be "blue"
```

```
</script>
```

Operators (Cont.)



❑ Comma Operator:

- The (, operator) cause two expressions to be executed sequentially.
- The (, operator) causes the expressions on either side of it to be executed in left-to-right order, and obtains the value of the expression on the right.
- It is commonly used when
 - naming variables,
 - in the increment expression of a for loop,
 - in function calls, arrays and object declarations.
- Example:
`var k=0, i, j=0;`

Operators (Cont.)



❑ **typeof Operator:**

- A unary operator returns a string that represents the data type.
- The return values of using typeof can be one of the following: "number", "string", "boolean", "undefined", "object", or "function".
- Example:

```
var myName = "javascript";  
typeof myName;    //string
```

Coercion



- ❑ Coercion is **converting** from one data type to another when expression is executed giving a result without causing any error.
 - ❑ Sometimes gives surprising results from human perspective.
 - ❑ JavaScript engine coerce :
 - Number to string
- $1 + "2" \rightarrow 12$

Operators Precedence



- ❑ **Operator precedence:** Determines the order in which operators are evaluated. Operators with higher precedence are evaluated first.
- ❑ **Operator Associativity:** Determines the order in which operators of the same precedence are processed.
- ❑ **The operators that you have learned are evaluated in the following order (from highest precedence to lowest):**
 1. Parentheses()
 2. Multiply/divide/modulus (*, /, %)
 3. Addition/Subtraction (+, -)
 4. Comparison (<, <=, >=, >)
 5. Equality (==, !=)
 6. Logical and (&&)
 7. Logical or (||)
 8. Conditional (?:)
 9. Assignment operators (=, +=, -=, *=, /=, %=)

Example:

$5 + 3 * 2 = 11 \rightarrow 3*2=6, \text{ then } 6+5 = 11.$

BUT $(5 + 3) * 2 = 16 \rightarrow 5+3 = 8, \text{ then } 8*2 = 16.$

Controlling Program Flow



❑ Control Statements that can be used are:

1. Conditional Statements

- a. ifelse
- b. switch/case

2. Loop Statements

- a. for
- b. while
- c. do...while

Controlling Statements (Cont.)



1. Conditional Statements

a) if....else

```
if (condition)
{
    do something;
}
else if (Condition)
{
    do something else;
}
else
{
    do something else;
}
```

b) switch / case

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

    case value2:
        statements
        break;

    default :
        statements
}
```


Controlling Statements (Cont.)



2. Loop Statements

a) **for**

```
for ( var i=0 ;i<10;i++)  
{  
    document.write(" number" + i)  
}
```

b) **while**

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

c) **do...while**

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

c) **for(...in...)**

```
for (variablename in object)  
{  
    statement  
}
```

Controlling Statements (Cont.)



❑ Breaking Loops :

- **break statement** : The break statement will break the loop and continue executing the code that follows after the loop (if any).
- **continue statement**: The continue statement will break the current loop and continue with the next value.

Dialogue Boxes



❑ Alert:

- The simplest way to direct output to a dialog box
- Example:

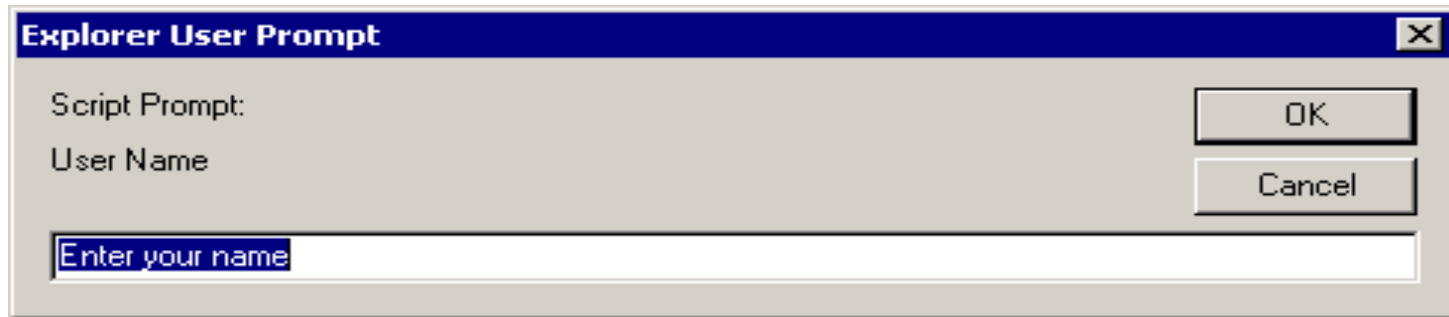
```
<script>  
    alert("Click Ok to continue.");  
</script>
```

Dialogue Boxes (Cont.)



□ Prompt:

- The simplest way to interact with the user.



- Example:

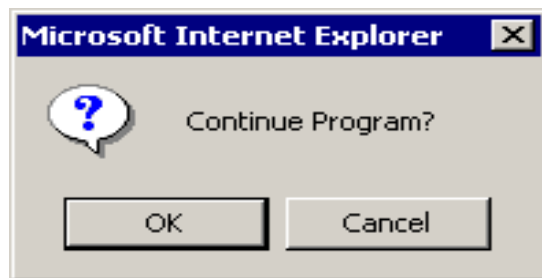
```
<script>  
    var Name = prompt('User Name' , 'Enter your name');  
</script>
```

Dialogue Boxes (Cont.)



❑ Confirm:

- displays a dialog box with two buttons: OK and Cancel.
 - If the user clicks on OK it will return true.
 - If the user clicks on the Cancel it will return false.



- Example:

```
<script>  
    var response = confirm('Are you sure you want to continue?');  
</script>
```

JavaScript Built-in Functions



Name	Description	Example
parseInt()	Convert string to int	<pre>parseInt("3") //returns 3 parseInt("3a") //returns 3 parseInt("a3") //returns NaN</pre>
parseFloat()	Convert string to float	<pre>parseFloat("3.55") //returns 3.55 parseFloat("3.55a") //returns 3.55 parseFloat("a3.55") //returns NaN</pre>
Number()	<p>The Number() function converts the object argument to a number that represents the object's value.</p> <p>if the value cannot be converted to a legal number, NaN is returned .</p>	<pre>var x1 = false, x2 = "999", x3 = "999 888"; document.write(Number(x1), Number(x2), Number(x3)); // returns 0, 999, Nan</pre> <p>Note:</p> <pre>parseInt("123hui"); //returns 123 Number("123hui"); //returns NaN</pre>
String()	Convert different objects to strings.	<pre>var x1 = New Boolean(0);, x2 = 999, x3 = "999 888"; document.write(String(x1), String(x2), String(Sx3)); // returns false, 999, 999 888</pre>

JavaScript Built-in Functions (Cont.)



Name	Description	Example
isFinite(num) (used to test number)	returns true if the string contains numbers only, else false	<pre>document.write(isFinite(33)) //returns true document.write(isFinite("Hello")) //returns false document.write(isFinite("33a")) //returns false</pre>
isNaN(val) (used to test string)	validate the argument for a number and returns true if the given value is not a number else returns false.	<pre>document.write(isNaN(0/0)) //returns true document.write(isNaN("348a")) //returns true document.write(isNaN("abc")) //returns true document.write(isNaN("348")) //returns false</pre>
eval(expression)	evaluates an expression and returns the result.	<pre>a=999; b=777; document.write(eval(b + a)); // returns 1776</pre>

JavaScript Built-in Functions (Cont.)



Name	Description	Example
<code>escape(string)</code>	method converts the special characters like space, colon etc. of the given string in to escape sequences.	<pre>escape("test val"); //test%20val</pre>
<code>unescape(string)</code>	function replaces the escape sequences with original values. e.g. %20 is the escape sequence for space " ".	<pre>unescape("test%20val"); //test val</pre>

JavaScript User-defined Functions

A circular icon with a blue border containing a white document with the letters 'JS' in purple.

```
function dosomething(x)
{
  statements
}
```

Function parameters

```
dosomething("hello")
```

Function call

```
function sayHi()
{
  statements
  return "hi"
}
z= sayHi()
```

The value of z is "hi"

JavaScript User-defined Functions (Cont.)



- ❑ Function can be called before its declaration block.
- ❑ Functions are not required to return a value and will return undefined implicitly if it is to set explicitly.
- ❑ When calling function it is not obligatory to specify all of its arguments.
 - The function has access to all of its passed parameters via arguments collection
 - We can specify default value using || operator or ES6 default function parameters

```
function dosomething (x)
{
    x = x || "nothing was sent";
    console.log ("value is :" + x);
}
```

```
dosomething("hello")
// value is : hello

dosomething("")
// value is : nothing was sent
```

```
// ES6
function dosomething (x = "nothing was sent")
{
    //x = x || "nothing was sent";
    console.log ("value is :" + x);
}
```

JavaScript Debugging Errors



Types of Errors

Syntax

Inaccurate capitalization, or forgetting to close quotation marks or parentheses).

HTML

Forgetting a fundamental HTML step will cause your JavaScript code to fail.

Runtime

Technically correct but performs an invalid function, such as dividing by zero, generate script that calls a non-existent function

Logical

Code that may not return an error but does not produce the result you expect).

JavaScript Debugging



- ❑ Modern browsers have JavaScript console within developer tool (F12) where errors in scripts are reported
 - Errors may differ across browsers
 - You can debug and add breakpoints and watches.
 - More details about debugging in JavaScript:
http://www.w3schools.com/js/js_debugging.asp

- ❑ Debugging tool in Firefox:
 - Download and use firefox add-on Firebug

- ❑ Adding **debugger;** (like breakpoints) statement in your code, will cause the debugger to stop on its position while in debugging mode.

Debugging Errors on FF(Cont.)



❑ Using Firebug Addon:

- **Installation:** (Open Firefox, open Tools menu → Add-ons → Get Add-ons → Browse all Addons, and then search for firebug, then choose add to firefox)

- **How to use:**

http://www.webmonkey.com/tutorial/Build_Better_Pages_With_Firebug

<http://www.evotech.net/blog/2007/06/introduction-to-firebug/>

<http://code.google.com/intl/ar-EG/apis/ajaxsearch/articles/firebug.html>

console object



- ❑ The Console object provides access to the browser's debugging console (e.g., the Web Console in Firefox).
- ❑ The specifics of how it works vary from browser to browser.
- ❑ In your browser, you can use the `console.log()` method to display data in the browser console (click F12 to open browser console).
- ❑ **Syntax:**

```
console.log("test message");
```

- ❑ Do not use it on production!
- ❑ More details: <https://developer.mozilla.org/en/docs/Web/API/console>

Self Study



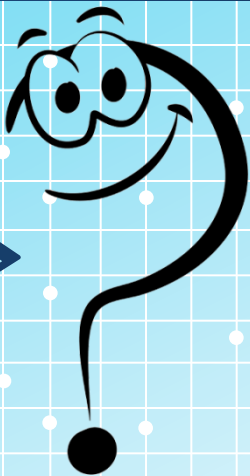
- ☐ What's New in ECMA script standard later versions (ES.next)?
- ☐ What's XSS (Cross-site scripting)?
- ☐ What's TypeScript? And what are its features?
- ☐ What's CoffeeScript?
- ☐ What's Dart?

<script>



JavaScript

</script>

<SCRIPT>  </SCRIPT>

<script>document.writeln("Thank
You!")</script>