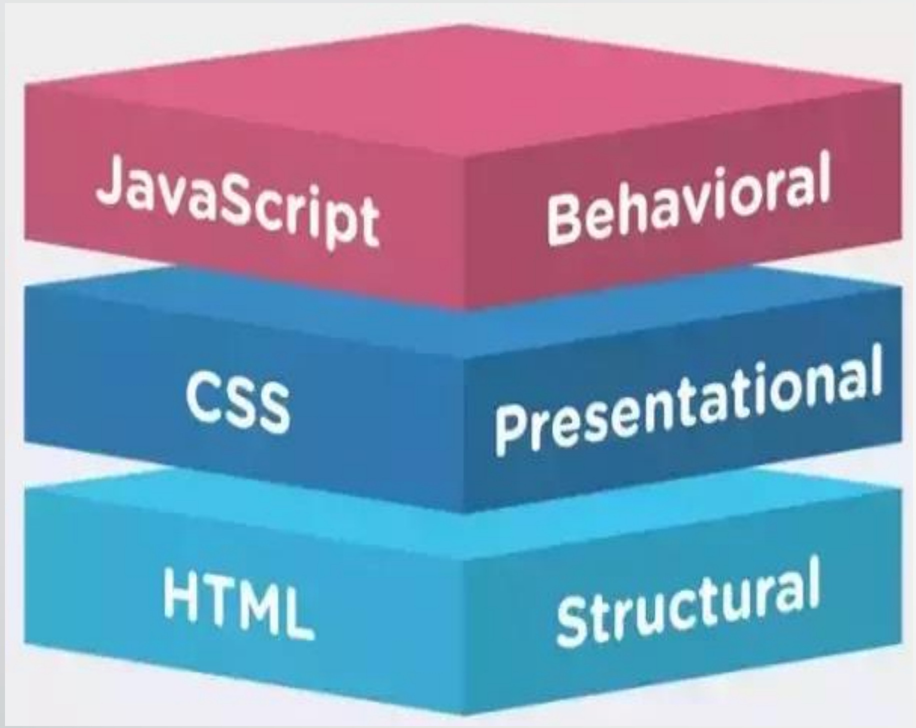
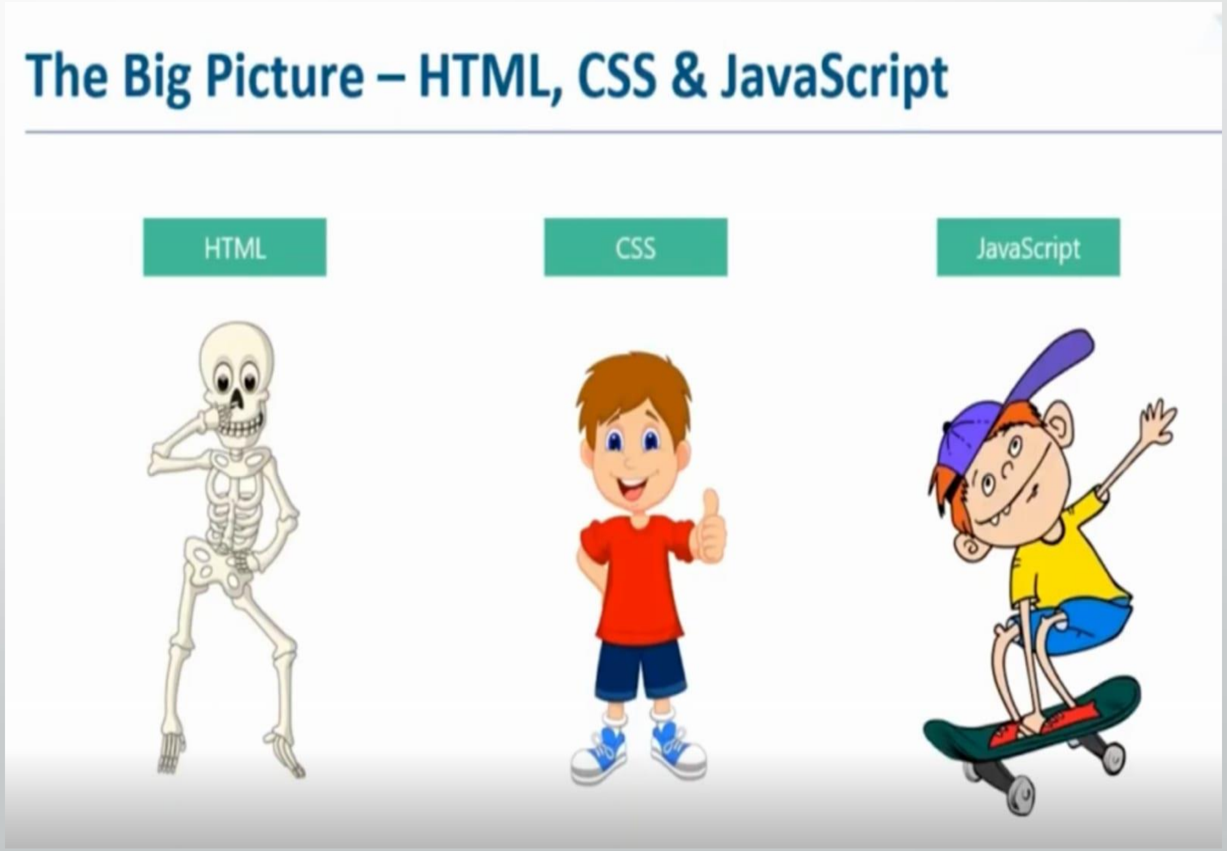
Introduction to JavaScript

ITI - Assiut Branch

Eng. Sara Ali

JavaScript


JavaScript
Behavioral
CSS
Presentational
HTML
Structural


The Big Picture – HTML, CSS & JavaScript
HTML
CSS
JavaScript

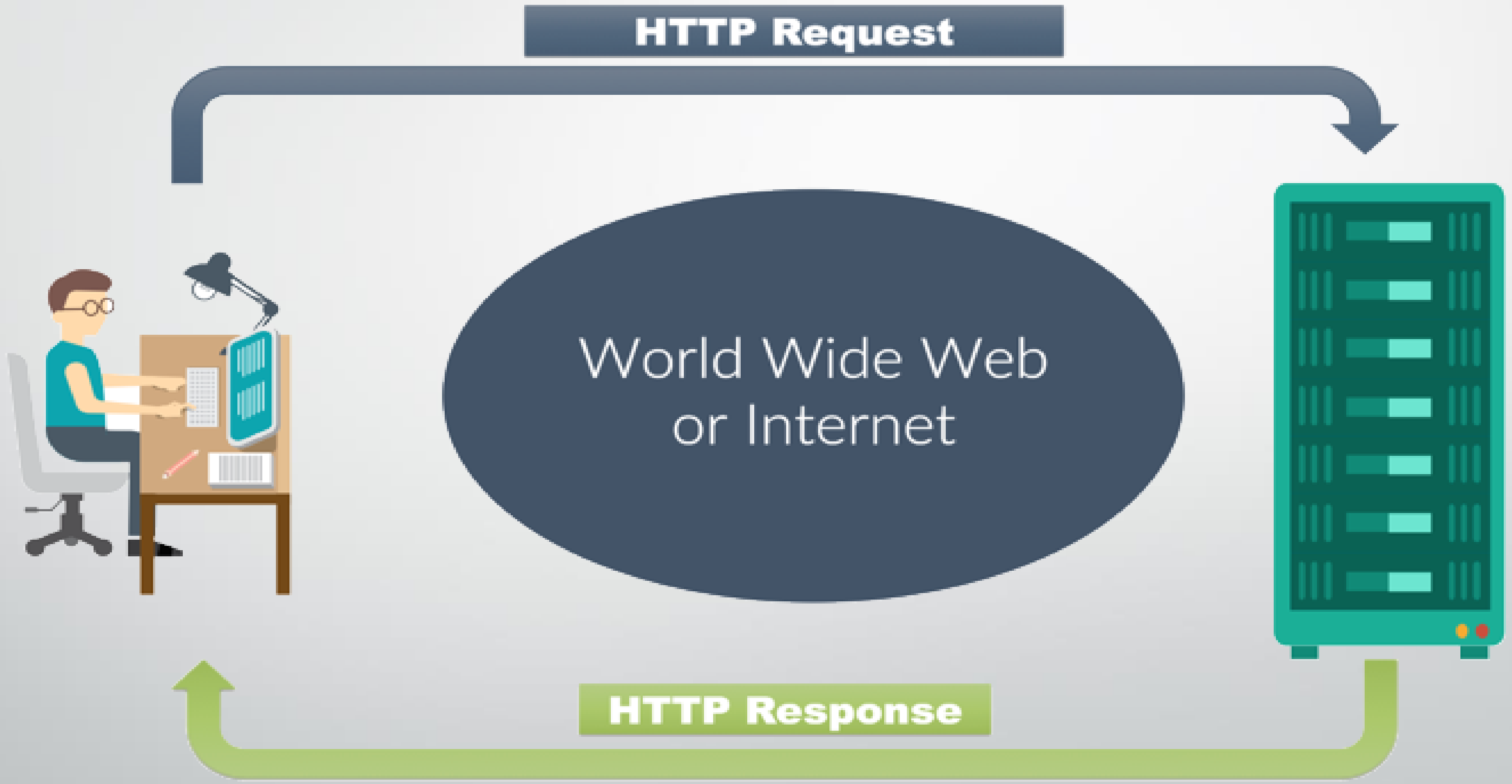
JavaScript…

JavaScript is a web client side scripting language.

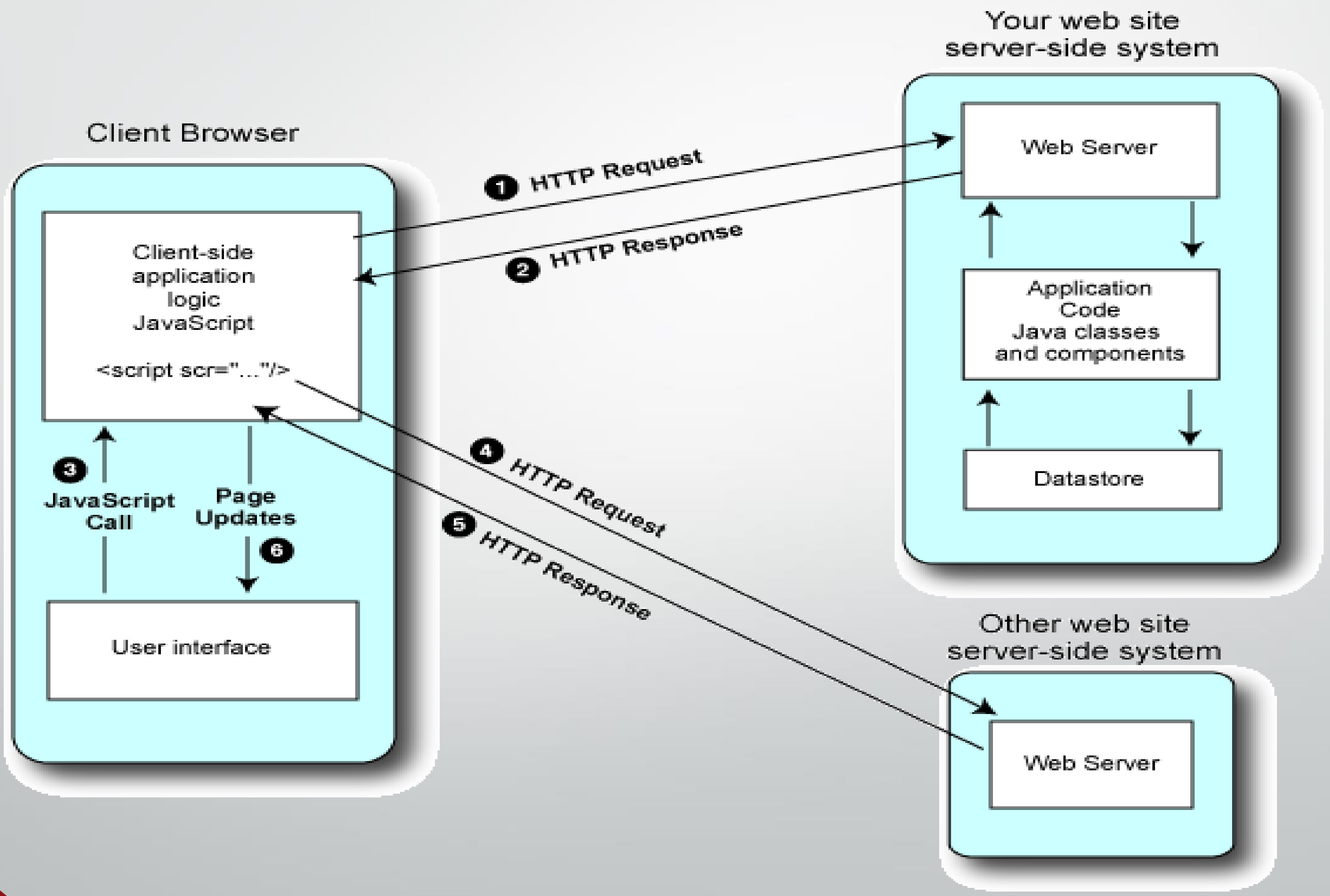
Web programming

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

Web programming (Cont.)



Web programming (Cont.)



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.

JavaScript

- JavaScript (JS) is a simple, flexible, lightweight, interpreted, programming language.
- 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#.
- Related to Java in name only (Name was part of a marketing deal).

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 History

- Created by Brendan Eich at Netscape in 1995.
- Before JavaScript take this name its name was Mocha.
- Then became LiveScript then with the name that we know now JavaScript.
- 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 European Computer Manufacturers Association (ECMA) International to carve out a standard specification.
- In 1997, ECMAScript (Official name of JavaScript) was introduced by ECMA International as an attempt at standardization.

JavaScript characteristics

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

What JavaScript can do?

- JavaScript Can Change HTML Content
- JavaScript Can Change HTML Attribute Values
- JavaScript Can Change HTML Styles (CSS)
- JavaScript Can Hide HTML Elements
- JavaScript Can Show HTML Elements
- Detecting the user's browser, OS, screen size, etc.
- Validating the user's input. It validates the data on the user’s machine before it is forwarded to the 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.)

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.)

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 “);
}
.
.
.
.
```

External JavaScript Advantages

Placing scripts in external files has some advantages:

- It separates HTML and code
- It makes HTML and JavaScript easier to read and maintain
- Cached JavaScript files can speed up page loads
- To add several script files to one page - use several script tags:

```
<script src= “myScript1.js” ></script>
<script src= “myScript2.js” ></script>
<script src="https://www.w3schools.com/js/myScript.js"></script>
```

JavaScript Output

JavaScript can "display" data in different ways:

- Writing into an HTML element, using innerHTML.
- Writing into the HTML output, using document.write().
- Writing into an alert box, using window.alert().
- Writing into the browser console, using console.log().

JavaScript Output (Cont.)

- **Using document.write()**

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

JavaScript Output (Cont.)

- **Using Window.alert()**
  - The simplest way to direct output to a dialog box

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

You can skip the window keyword.

JavaScript Output (Cont.)

- **Using Console.log()**
  - In your browser, you can use the console.log() method to display data in the browser console (click F12 to open browser console).

```
<script>
        console.log(“test message”);
</script>
```

Do not use it on production!

Variables

- **Naming:**

  The general rules for constructing names for variables (unique identifiers) are:

  - Names can contain letters, digits, underscores, and dollar signs.
  - Names must begin with a letter
  - Names can also begin with $ and _
  - Names are case sensitive (y and Y are different variables)
  - Reserved words (like JavaScript keywords) cannot be used as names
  - Don’t use spaces inside names. FirstName NOT First Name.

Variables (cont.)

- **Types:**
  - 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:

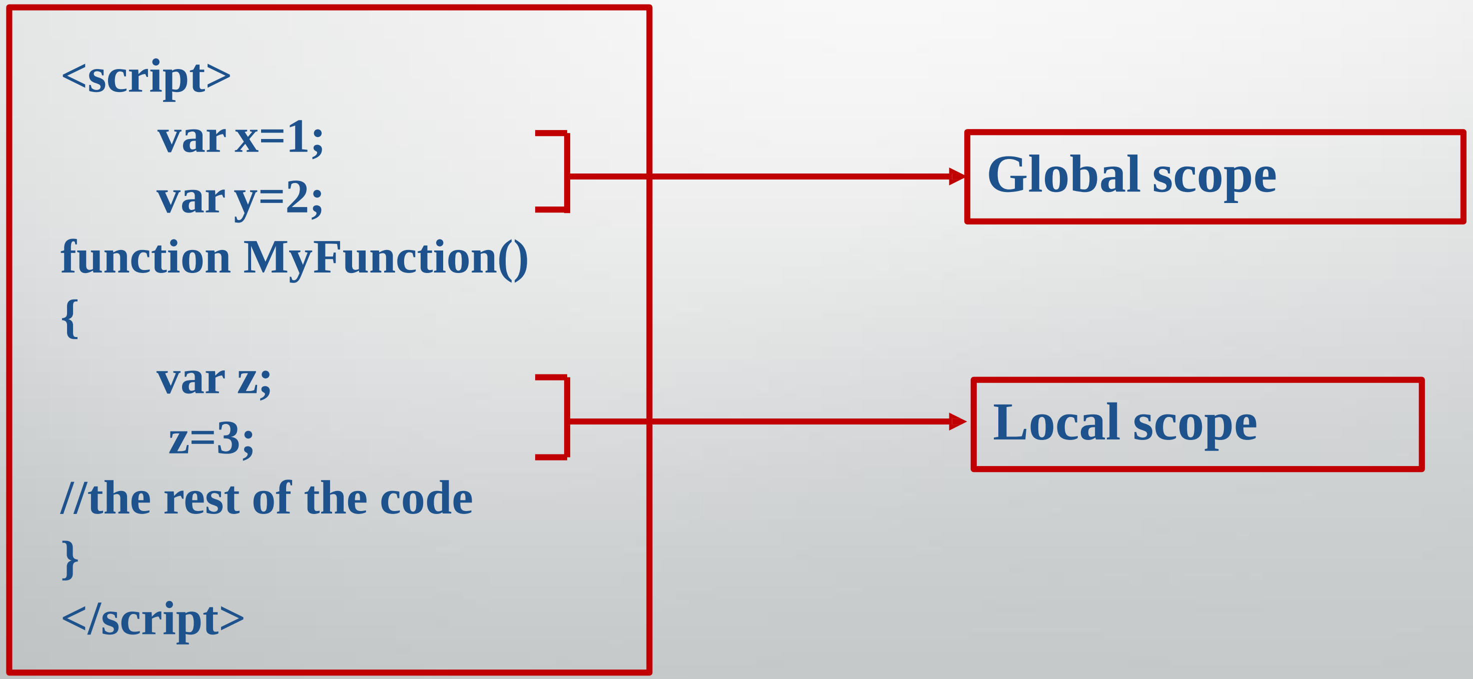
```
var count;
alert (count); //undefined  count=3;
//or
var count=3;
// Case Sensitive
var Count=3; //Is a new variable
```

  - 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

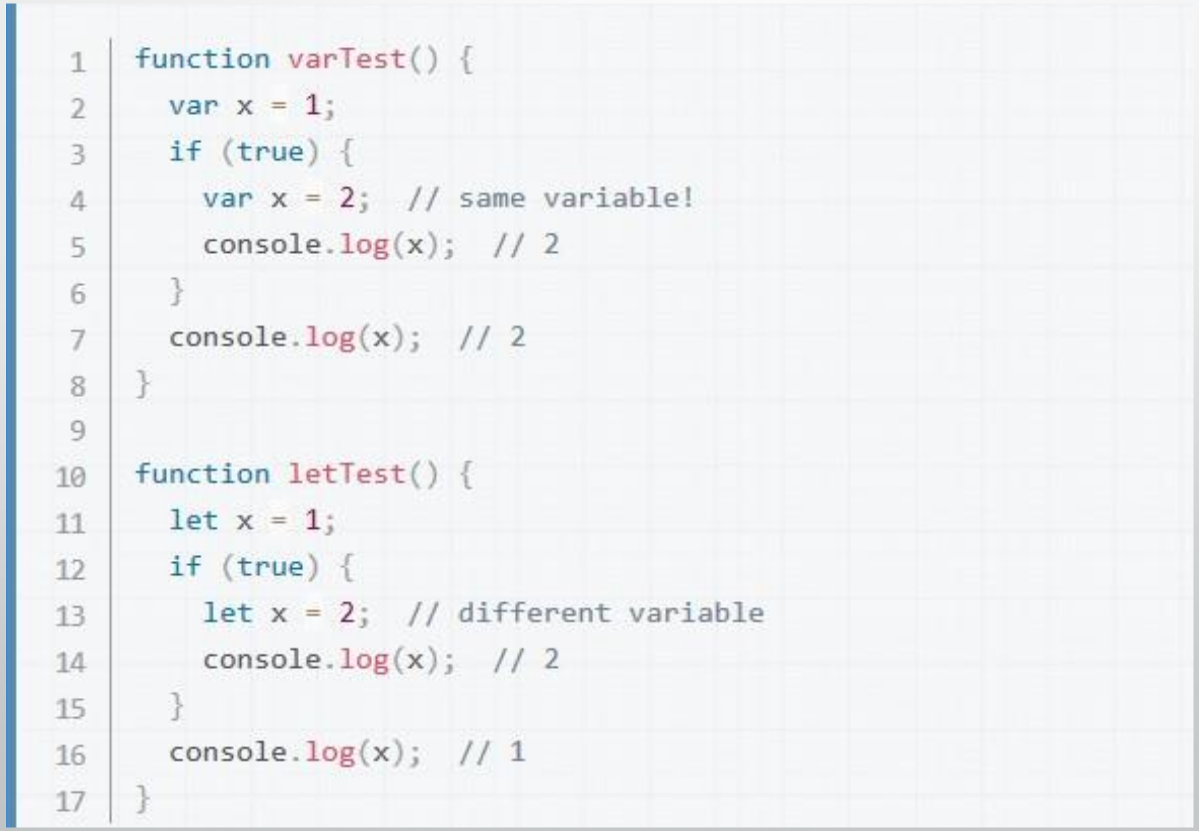


```
<script>
        var x=1;
        var y=2;
function MyFunction()
{
        var z;
         z=3;
//the rest of the code
}
</script>
```

Variables – Life time & scope

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.
- let variables are block-scoped. The scope of a variable declared with let is just the enclosing block, not the whole enclosing function.



```
function varTest() {
  var x = 1;
  if (true) {
    var x = 2;  // same variable!
    console.log(x);  // 2
  }
  console.log(x);  // 2
}

function letTest() {
  let x = 1;
  if (true) {
    let x = 2;  // different variable
    console.log(x);  // 2
  }
  console.log(x);  // 1
}
```

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.
        }
}
```

  - 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 Comments

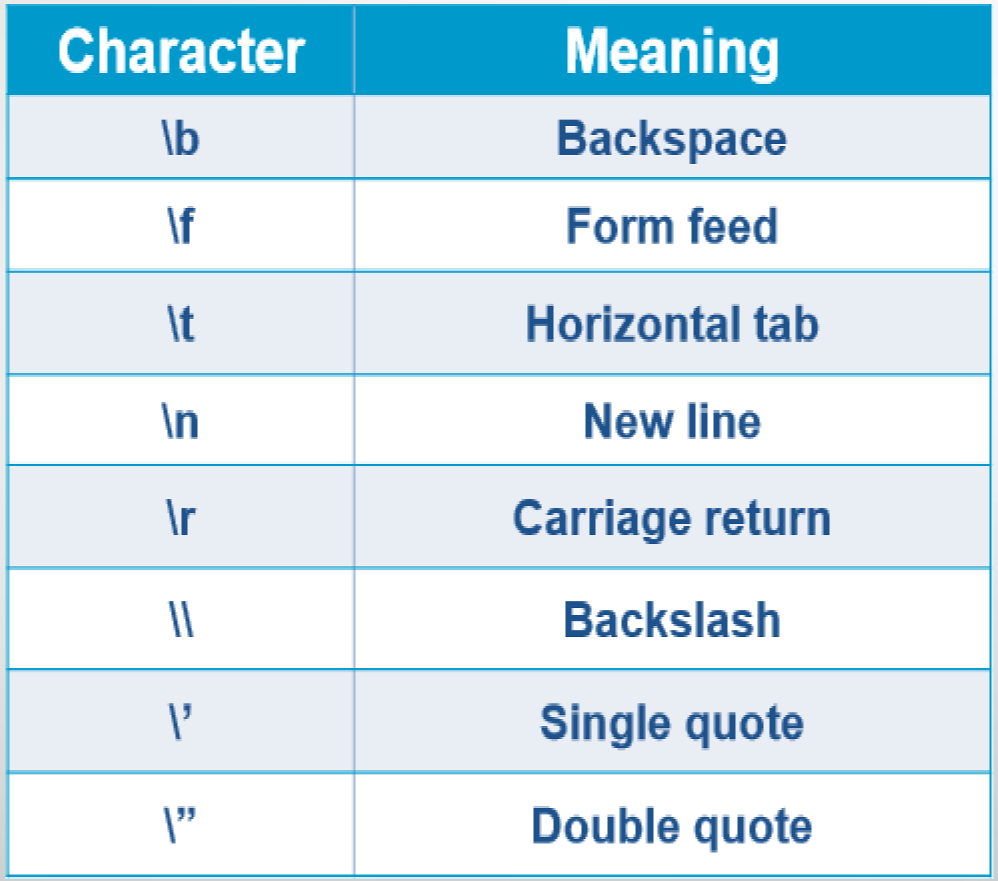
- Multiple line comments preceded by /* and ended by */

```
/* This is a comment block.
 It contains several lines */

 document.write("Hello World!")
```

- Single line Comment are preceded by a double-slash (//).

```
//this is one line comment

 document.write("Hello World!")
```

JavaScript special characters



Character	Meaning
\b	Backspace
\f	Form feed
\t	Horizontal tab
\n	New line
\r	Carriage return
\\	Backslash
\’	Single quote
\”	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.
  - 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", "undefned", "object", or "function".
  - ➢ Example:

```
var myName = “javascript”;
typeof myName;      //string
```

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 → 3*2=6 , then 6+5 = 11.

BUT (5 + 3) * 2 = 16 → 5+3 = 8 , then 8*2 = 16.

Controlling Program Flow

- **Control Statements that can be used are:**

1. **Conditional Statements**
    a. if ….else
    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…)

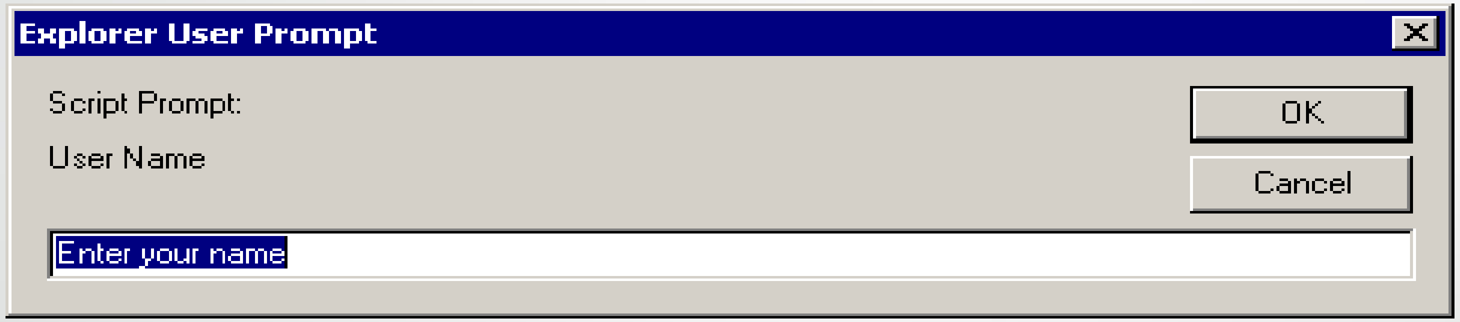
```
for (variablename in object)
{
            statement
}
```

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

- **Prompt:**
  - The simplest way to interact with the user.

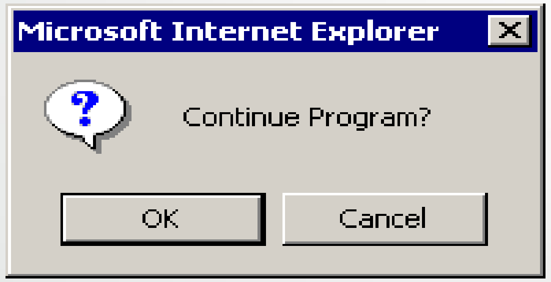


  - Example:

```
<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.


Microsoft Internet Explorer

Continue Program?

  - 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**	parseInt("3") //returns 3
		parseInt(“3a”) //returns 3
		parseInt(“a3”) //returns NaN

parseFloat()	**Convert string to float**	parseFloat("3.55") //returns 3.55
		parseFloat(“3.55a”) //returns 3.55
		parseFloat(“a3.55”) //returns NaN

Number()	**The Number() function converts the object argument to a number that represents the object's value.**
	
	if the value cannot be converted to a legal number, NaN is returned .
		var x1 = false, x2 = "999“, x3 = "999 888";
		document.write(Number(x1), Number(x2), Number(x3));
		// returns 0, 999, Nan
		Note:
		parseInt("123hui"); //returns 123
		Number("123hui"); //returns NaN

String()	**Convert different objects to strings.**	var x1 = New Boolean(0);, x2 = 999, x3 = "999 888";
		document.write(String(x1), String(x2), String(Sx3));
		// returns false, 999, 999 888

JavaScript Built-in Functions (Cont.)

Name	Description	Example

isFinite(num)
(used to test number)
	returns true if the string contains numbers only, else false	**document.write(isFinite(33))**
		//returns true
		document.write(isFinite(“Hello”))
		//returns false
		document.write(isFinite(“33a”))
		//returns false

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.	**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

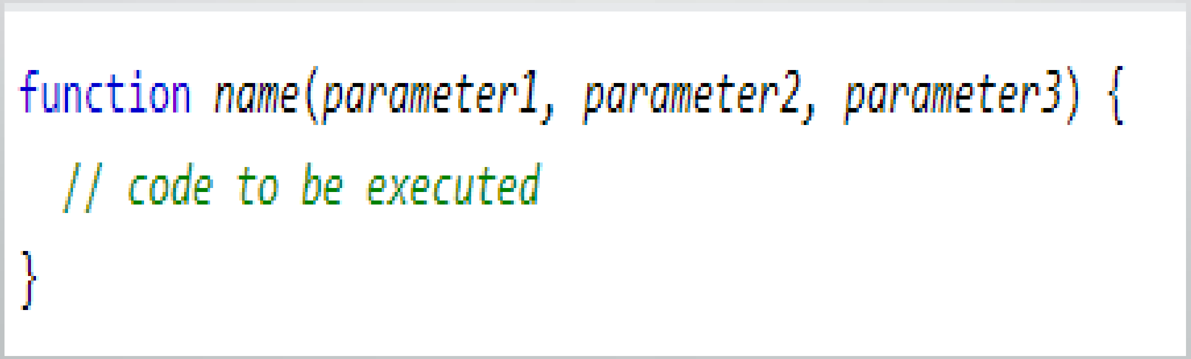
eval(expression)	evaluates an expression and returns the result.	**a=999; b=777;**
		document.write(eval(b + a));
		// returns 1776

JavaScript Built-in Functions (Cont.)

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

JavaScript User-defined Functions

- A JavaScript function is defined with the function keyword, followed by a name, followed by parentheses ().
- Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).
- The parentheses may include parameter names separated by commas:
- (parameter1, parameter2, ...)
- The code to be executed, by the function, is placed inside curly brackets: {}



```
function name(parameter1, parameter2, parameter3) {
  // code to be executed
}
```

JavaScript User-defined Functions (Cont.)

- **Function Invocation**

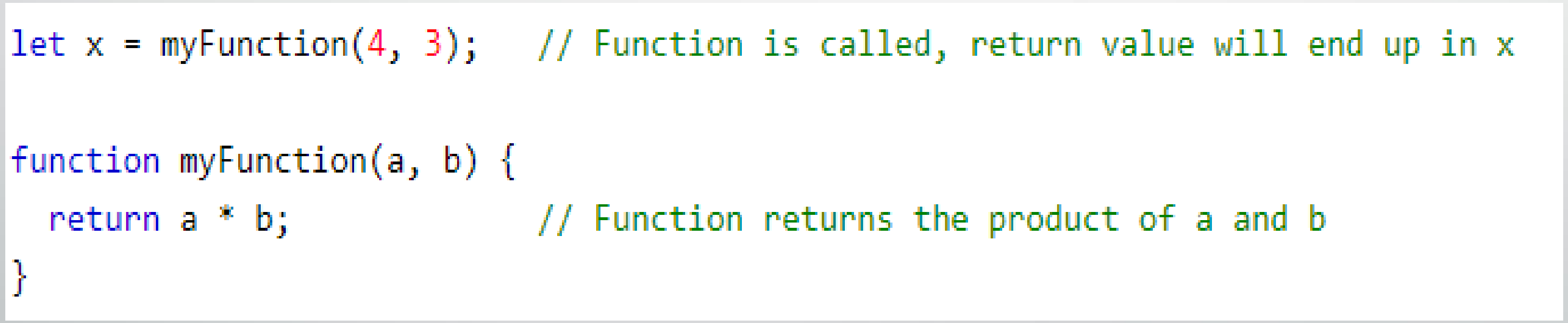
The code inside the function will execute when "something" **invokes** (calls) the function:

- When an event occurs (when a user clicks a button)
- When it is invoked (called) from JavaScript code
- Automatically (self invoked)

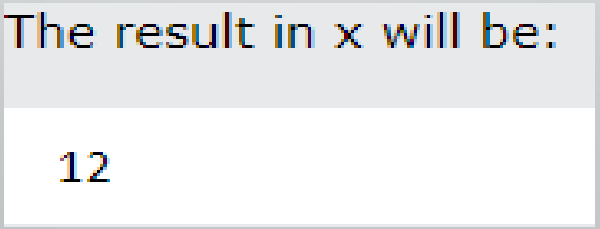
```
<script>
        //self invoked function
        ( function() {
                //code
        } () );
</script>
```

JavaScript User-defined Functions (Cont.)

- **Function Return**
  - Functions are not required to return a value and will return undefined implicitly if it is to set explicitly.
  - When JavaScript reaches a return statement, the function will stop executing.
  - The return value is "returned" back to the "caller"



```
let x = myFunction(4, 3);   // Function is called, return value will end up in x

function myFunction(a, b) {
  return a * b;             // Function returns the product of a and b
}
```

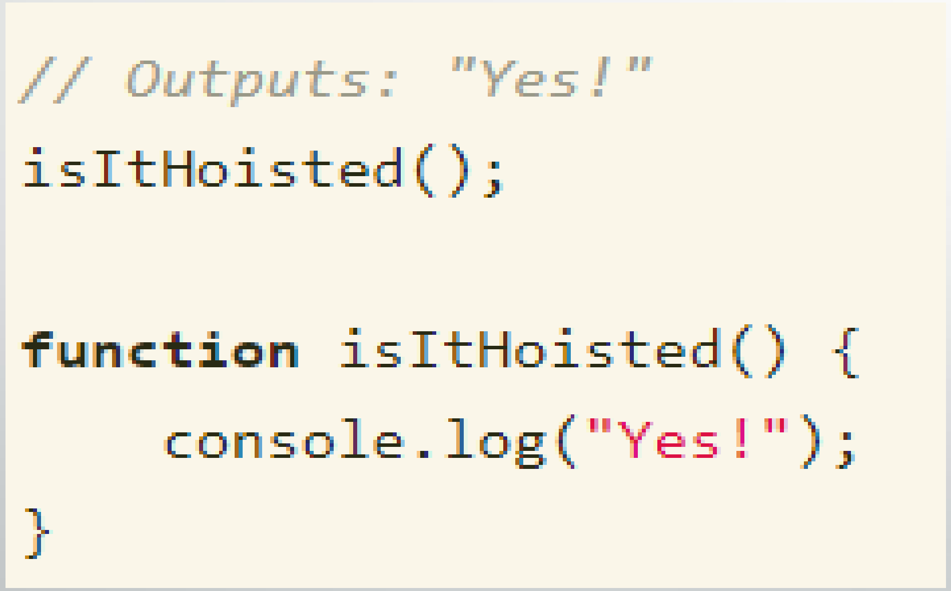


```
The result in x will be:

12
```

JavaScript User-defined Functions (Cont.)

- **Function Hoisting**

  Function can be called before its declaration block.



```
// Outputs: "Yes!"
isItHoisted();

function isItHoisted() {
    console.log("Yes!");
}
```

JavaScript

THANK YOU