

# **Introduction to Java script**

ICS 2203

# What is JavaScript?

- a dynamic computer programming language. lightweight and most commonly used as a part of web pages, whose implementations allow client-side script to interact with the user and make dynamic pages.
- It is an interpreted programming language with object-oriented capabilities.
- made its first appearance in Netscape 2.0 in 1995 with the name **LiveScript** as a way to add programs to web pages in the Netscape Navigator browser.
- Netscape changed name to JavaScript, possibly because of the excitement being generated by Java.
- has since been adopted by all other major graphical web browsers. With the general-purpose core of the language being embedded in the web browsers.

- The ECMAScript standard, done by the Ecma International organization -describes the way the JavaScript language should work.
- In practice, the terms ECMAScript and JavaScript can be used interchangeably
- The ECMA-262 Specification defined a standard version of the core JavaScript language.
  - JavaScript is a lightweight, interpreted programming language.
  - Designed for creating network-centric applications.
  - Complementary to and integrated with Java.
  - Complementary to and integrated with HTML.
  - Open and cross-platform.

## Advantages of JavaScript

- **Less server interaction:** You can validate user input before sending the page off to the server. This saves server traffic, which means less load on your server.
- **Immediate feedback to the visitors:** They don't have to wait for a page reload to see if they have forgotten to enter something.
- **Increased interactivity:** You can create interfaces that react when the user hovers over them with a mouse or activates them via the keyboard.
- **Richer interfaces:** You can use JavaScript to include such items as drag-and-drop components and sliders to give a Rich Interface to your site visitors.

## Limitations of JavaScript

- We cannot treat JavaScript as a full-fledged programming language. It lacks the following important features:
  - Client-side JavaScript does not allow the reading or writing of files. This has been kept for security reason.
  - JavaScript cannot be used for networking applications because there is no such support available.
  - JavaScript doesn't have any multithreading or multiprocessor capabilities.

=>Once again, JavaScript is a lightweight, interpreted programming language that allows you to build interactivity into otherwise static HTML pages.

# SYNTAX

- Implemented using JavaScript statements placed within the **<script>...</script>** HTML tags in a web page.
- **<script>** tags, containing JavaScript can be placed, anywhere within a web page, but it is recommended within the **<head>** tags.
- The **<script>** tag alerts the browser program to start interpreting all the text between these tags as a script. A simple syntax is as follows;

```
<script ...>  
JavaScript code  
</script>
```

- The script tag takes two important attributes:
  - **Language:** specifies what scripting language you are using. Typically, its value will be javascript. Recent versions of HTML (and XHTML) have phased out this.
  - **Type:** it is what is now recommended to indicate the scripting language in use and its value should be set to "text/javascript".
- So your JavaScript syntax will look as follows.

```
<script language="javascript" type="text/javascript">  
JavaScript code  
</script>
```

# Your First JavaScriptCode

- a sample example to print out "Hello World".

```
<html>
<body>
<script language="javascript"
type="text/javascript">
<!--
document.write ("Hello World!")
//-->
</script>
</body>
</html>
```

- An optional HTML comment surrounds the JavaScript code to save the code from a browser that does not support JavaScript. "//" signifies a comment in JavaScript added to prevent a browser from reading the end of the HTML comment as a piece of JavaScript code.
- a function **document.write** which writes a string into our HTML document is called. This function can be used to write text, HTML, or both.

## Whitespace and Line Breaks

- JavaScript ignores spaces, tabs, and newlines that appear in JavaScript programs. You can use spaces, tabs, and newlines freely in your program and you are free to format and indent your programs in a neat and consistent way that makes the code easy to read and understand.

**Semicolons are Optional:** if each of your statements are placed on a separate line.

- **Note:** It is a good programming practice to use semicolons.

## Case Sensitivity

- It is a case-sensitive language. This means that the language keywords, variables, function names, and any other identifiers must always be typed with a consistent capitalization of letters.
- So the identifiers **Time** and **TIME** will convey different meanings in JavaScript.

**NOTE:** Care should be taken while writing variable and function names in JavaScript.

# Comments in JavaScript

- JavaScript supports both C-style and C++-style comments. Thus:
  - Any text between a `//` and the end of a line is treated as a comment and is ignored by JavaScript.
  - Any text between the characters `/*` and `*/` is treated as a comment. This may span multiple lines.
  - JavaScript also recognizes the HTML comment opening sequence `<!--`. JavaScript treats this as a single-line comment, just as it does the `//` comment.
  - The HTML comment closing sequence `-->` is not recognized by JavaScript so it should be written as `//-->`.



# Warning for Non-JavaScript Browsers

- If you have to do something important using JavaScript, then you can display a warning message to the user using **<noscript>** tags.
- You can add a **noscript** block immediately after the script block as follows:

```
<html>
<body>
<script language="javascript" type="text/javascript">
<!--
document.write ("Hello World!")
//-->
</script>
<noscript>
Sorry...JavaScript is needed to go ahead.
</noscript>
</body>
</html>
```
- Now, if the user's browser does not support JavaScript or JavaScript is not enabled, then the message from **</noscript>** will be displayed on the screen.

# PLACEMENT

- There is a flexibility given to include JavaScript code anywhere in an HTML document.
- The most preferred ways are as follows:
  - Script in `<head>...</head>` section.
  - Script in `<body>...</body>` section.
  - Script in `<body>...</body>` and `<head>...</head>` sections.
  - Script in an external file and then include in `<head>...</head>` section.

# 1. JavaScript in <head>...</head> Section

- Sample prog: to have a script run on some event, such as a user click;

```
<html>
<head>
<script type="text/javascript">
<!--
function sayHello() {
alert("Hello World")
}
//-->
</script>
</head>
<body>
Click here for the result
<input type="button" onclick="sayHello()" value="Say Hello" />
</body>
</html>
```

## 2. JavaScript in <body>...</body> Section

- Sample prog: a script to run as the page loads, the script generates content in the page. It goes in the <body> portion of the document.
- In this case, no function is defined using JavaScript. E.g.

```
<html>
<head>
</head>
<body>
<script type="text/javascript">
<!--
document.write("Hello World")
//-->
</script>
<p>This is web page body </p>
</body>
</html>
```

# 3. JavaScript in <body> and <head> Sections

- Sample code.

```
<html>
<head>
<script type="text/javascript">
<!--
function sayHello() {
alert("Hello World")
}
//-->
</script>
</head>
<body>
<script type="text/javascript">
<!--
document.write("Hello World")
//-->
</script>
<input type="button" onclick="sayHello()" value="Say Hello" />
</body>
</html>
```

# JavaScript in External File

- In cases where there is reusing of identical JavaScript code on multiple pages of a site.
- **Script** tag provides a mechanism to allow you to store JavaScript in an external file and then include it into your HTML files. For example;

```
<html>
<head>
<script type="text/javascript" src="filename.js" ></script>
</head>
<body>
.....
</body>
</html>
```

- write all your JavaScript source code in a simple text file with the extension ".js" and then include the file as shown above.
- For example, say the following content is in **filename.js** then you can use **sayHello** function in your HTML file after including the filename.js file.

```
function sayHello() {
alert("Hello World")
}
```

# JavaScript Datatypes

- The set of data types a programming language supports-the type of values that can be represented and manipulated in a programming language.
- JavaScript allows you to work with three primitive data types:
  - **Numbers**, e.g., 123, 120.50 etc.
  - **Strings** of text, e.g. "This text string" etc.
  - **Boolean**, e.g. true or false.
- also defines two trivial data types, **null** and **undefined**, each of which defines only a single value.
- In addition to these primitive data types, JavaScript supports a composite data type known as **object**.
- **Note:** Java does not make a distinction between integer values and floating-point values. All numbers in JavaScript are represented as floating-point values. JavaScript represents numbers using the 64-bit floating-point format defined by the IEEE 754 standard.

# Variables

- Variables can be thought of as named containers. You can place data into these containers and then refer to the data simply by naming the container.
- Before you use a variable in a JavaScript program, you must declare it with the **var** keyword as follows.

```
<script type="text/javascript">  
<!--  
var money;  
var name;  
//-->  
</script>
```

- You can also declare multiple variables with the same **var** keyword as follows:

```
<script type="text/javascript">  
<!--  
var money, name;  
//-->  
</script>
```



- Storing a value in a variable is called **variable initialization**. You can do variable initialization at the time of variable creation or at a later point in time when you need that variable.
- For instance, you might create a variable named **money** and assign the value 2000.50 to it later. For another variable, you can assign a value at the time of initialization as follows.

```
<script type="text/javascript">  
<!--  
var name = "Ali";  
var money;  
money = 2000.50;  
//-->  
</script>
```

- **Note:** Use the **var** keyword only for declaration or initialization, once for the life of any variable name in a document. You should not re-declare same variable twice.
- JavaScript is **untyped** language. i.e. a variable can hold a value of any data type. The value type of a variable can change during the execution of a program and JavaScript takes care of it automatically.

## JavaScript Variable Scope

- The scope of a variable is the region of your program in which it is defined. JavaScript variables have only two scopes.
  - **Global Variables:** A global variable has global scope which means it can be defined anywhere in your JavaScript code.
  - **Local Variables:** A local variable will be visible only within a function where it is defined. Function parameters are always local to that function.
- Within the body of a function, a local variable takes precedence over a global variable with the same name. If you declare a local variable or function parameter with the same name as a global variable, you effectively hide the global variable. Take a look into the following example.

```
<script type="text/javascript">
<!--
var myVar = "global"; // Declare a global variable
function checkscope( ) {
var myVar = "local"; // Declare a local variable
document.write(myVar);
}
//-->
</script>
```

# JavaScript Variable Names

- While naming your variables in JavaScript, keep the following rules in mind.
  - You should not use any of the JavaScript reserved keywords as a variable name. For example, **break** or **boolean** variable names are not valid.
  - JavaScript variable names should not start with a numeral (0-9). They must begin with a letter or an underscore character. For example, **123test** is an invalid variable name but **\_123test** is a valid one.
  - JavaScript variable names are case-sensitive. For example, **Name** and **name** are two different variables.

# JavaScript Reserved Words

- A list of all the reserved words in JavaScript are given below. They cannot be used as JavaScript variables, functions, methods, loop labels, or any object names.

- |            |              |              |                |
|------------|--------------|--------------|----------------|
| • abstract | • else       | • Instanceof | • switch       |
| • boolean  | • enum       | • int        | • synchronized |
| • break    | • export     | • interface  | • this         |
| • byte     | • extends    | • long       | • throw        |
| • case     | • false      | • native     | • throws       |
| • catch    | • final      | • new        | • transient    |
| • char     | • finally    | • null       | • true         |
| • class    | • float      | • package    | • try          |
| • const    | • for        | • private    | • typeof       |
| • continue | • function   | • protected  | • var          |
| • debugger | • goto       | • public     | • void         |
| • default  | • if         | • return     | • volatile     |
| • delete   | • implements | • short      | • while        |
| • do       | • import     | • static     | • with         |
| • double   | • in         | • Super      |                |

# OPERATORS

## What is an Operator?

- JavaScript supports the following types of operators.
  - Arithmetic Operators
  - Comparison Operators
  - Logical (or Relational) Operators
  - Assignment Operators
  - Conditional (or ternary) Operators

The following code shows how to use arithmetic operators in JavaScript.

```
<html>
<body>
<script type="text/javascript">
<!--
var a = 33;
var b = 10;
var c = "Test";
var linebreak = "<br />";
document.write("a + b = ");
result = a + b;
document.write(result);
document.write(linebreak);
document.write("a - b = ");
result = a - b;
document.write(result);
document.write(linebreak);
document.write("a / b = ");
result = a / b;
document.write(result);
```

```
document.write(linebreak);
document.write("a % b = ");
result = a % b;
document.write(result);
document.write(linebreak);
document.write("a + b + c = ");
result = a + b + c;
document.write(result);
document.write(linebreak);
a = a++;
document.write("a++ = ");
result = a++;
document.write(result);
document.write(linebreak);
b = b--;
document.write("b-- = ");
result = b--;
document.write(result);
document.write(linebreak);
```

```
//-->
</script>
<p>Set the variables to
different values and then
try...</p>
</body>
</html>
```

The following code shows how to use comparison operators in JavaScript.

```
<html>
<body>
<script type="text/javascript">
<!--
var a = 10;
var b = 20;
var linebreak = "<br />";
document.write("(a == b) => ");
result = (a == b);
document.write(result);
document.write(linebreak);
document.write("(a < b) => ");
result = (a < b);
document.write(result);
document.write(linebreak);
document.write("(a > b) => ");
result = (a > b);
document.write(result);
document.write(linebreak);
```

```
document.write("(a != b) => ");
result = (a != b);
document.write(result);
document.write(linebreak);
document.write("(a >= b) => ");
result = (a >= b);
document.write(result);
document.write(linebreak);
document.write("(a <= b) => ");
result = (a <= b);
document.write(result);
document.write(linebreak);
//-->
</script>
<p>Set the variables to different values
and different operators and then
try...</p>
</body>
</html>
```

Try the following code to learn how to implement Logical Operators in JavaScript.

```
<html>
<body>
<script type="text/javascript">
<!--
var a = true;
var b = false;
var linebreak = "<br />";
document.write("(a && b) => ");
result = (a && b);
document.write(result);
document.write(linebreak);
document.write("(a || b) => ");
result = (a || b);
document.write(result);
document.write(linebreak);
```

```
document.write("!(a && b) => ");
result = (!(a && b));
document.write(result);
document.write(linebreak);
//-->
</script>
<p>Set the variables to different
values and different operators and
then try...</p>
</body>
</html>
```



# Miscellaneous Operators

- We have two operators that are quite useful in JavaScript: the **conditional operator** (**? :**) and the **typeof operator**.

## 1. Conditional Operator (**? :**)

- The conditional operator first evaluates an expression for a true or false value and then executes one of the two given statements depending upon the result of the evaluation.

**? : (Conditional )**

If Condition is true? Then value X : Otherwise value Y

## Example

```
<html>
<body>
<script
type="text/javascript">
<!--
var a = 10;
var b = 20;
var linebreak = "<br />";
document.write ("((a > b) ?
100 : 200) => ");
result = (a > b) ? 100 : 200;
document.write(result);
document.write(linebreak);
```

```
document.write ("((a < b) ?
100 : 200) => ");
result = (a < b) ? 100 : 200;
document.write(result);
document.write(linebreak);
//-->
</script>
<p>Set the variables to
different values and
different operators and then
try...</p>
</body>
</html>
```

## typeof Operator

- a unary operator placed before its single operand, which can be of any type.
- Its value is a string indicating the data type of the operand.
- The *typeof* operator evaluates to "number", "string", or "boolean" if its operand is a number, string, or boolean value and returns true or false based on the evaluation.
- Here is a list of the return values for the **typeof** Operator.

Type	String Returned by typeof
Number	"number"
String	"string"
Boolean	"boolean"
Object	"object"
Function	"function"
Undefined	"undefined"
Null	"object"

```
Example;  
<html>  
<body>  
<script type="text/javascript">  
<!--  
var a = 10;  
var b = "String";  
var linebreak = "<br />";  
result = (typeof b == "string" ?  
"B is String" : "B is Numeric");  
document.write("Result => ");  
document.write(result);  
document.write(linebreak);  
result = (typeof a == "string" ?  
"A is String" : "A is Numeric");
```

```
document.write("Result => ");  
document.write(result);  
document.write(linebreak);  
//-->  
</script>  
<p>Set the variables to  
different values and different  
operators and then try...</p>  
</body>  
</html>
```

The following example shows how to implement Label with a break statement.

```
<html>
<body>
<script type="text/javascript">
<!--
document.write("Entering the
loop!<br /> ");
outerloop: // This is the label name
for (var i = 0; i < 5; i++)
{
document.write("Outerloop: " + i +
"<br />");
innerloop:
for (var j = 0; j < 5; j++)
{
if (j > 3 ) break ; // Quit the
```

```
innermost loop
if (i == 2) break innerloop; // Do the
same thing
if (i == 4) break outerloop; // Quit
the outer loop
document.write("Innerloop: " + j + "
<br />");
}
}
document.write("Exiting the
loop!<br /> ");
//-->
</script>
</body>
</html>
```

The following example shows how to implement Label with continue.

```
<html> Javascript
<body>
<script type="text/javascript">
<!--
document.write("Entering the
loop!<br /> ");
outerloop: // This is the label name
for (var i = 0; i < 3; i++)
{
document.write("Outerloop: " + i +
"<br />");
for (var j = 0; j < 5; j++)
{
if (j == 3){
```

```
continue outerloop;
}
document.write("Innerloop: " + j +
"<br />");
}
}
document.write("Exiting the loop!<br
/> ");
//-->
</script>
</body>
</html>
```

# Function Definition

- Before we use a function, we need to define it. The most common way to define a function in JavaScript is by using the **function** keyword, followed by a unique function name, a list of parameters (that might be empty), and a statement block surrounded by curly braces.

## Syntax

```
<script type="text/javascript">
<!--
function functionname(parameter-list)
{
statements
}
//-->
</script>
```

## Example

```
<html>
<head>
<script type="text/javascript">
function sayHello()
{
document.write ("Hello there!");
}
</script>
</head>
<body>
<p>Click the following button to call the function</p>
<form>
<input type="button" onclick="sayHello()" value="Say Hello">
</form>
<p>Use different text in write method and then try...</p>
</body>
</html>
```



# Events

## What is an Event?

- JavaScript's interaction with HTML is handled through events that occur when the user or the browser manipulates a page.
- Examples of events; When the page loads, When the user clicks a button, pressing any key, closing a window, resizing a window, etc.
- Developers can use these events to execute JavaScript coded responses, which cause buttons to close windows, messages to be displayed to users, data to be validated, and virtually any other type of response imaginable.
- Events are a part of the Document Object Model (DOM) Level 3 and every HTML element contains a set of events which can trigger JavaScript Code.

## onclick Event Type

- occurs when a user clicks the left button of his mouse. You can put your validation, warning etc., against this event type.

## Example

Try the following example;

```
<html>
<head>
<script type="text/javascript">
<!--
function sayHello() {
document.write ("Hello World")
}
//-->
</script>
</head>
<body>
<p> Click the following button and see result</p>
<input type="button" onclick="sayHello()" value="Say Hello" />
</body>
</html>
```

# onsubmitEvent Type

- **onsubmit** is an event that occurs when you try to submit a form. You can put your form validation against this event type.

## Example

- The following example shows how to use onsubmit. Here we are calling a **validate()** function before submitting a form data to the webserver. If **validate()** function returns true, the form will be submitted, otherwise it will not submit the data.

```
<html>
<head>
<script type="text/javascript">
<!--
function validation() {
all validation goes here
.....
return either true or false
}
//-->
</script>
</head>
<body>
<form method="POST" action="t.cgi" onsubmit="return validate()">
.....
<input type="submit" value="Submit " />
</form>
</body>
</html>
```

# Strings

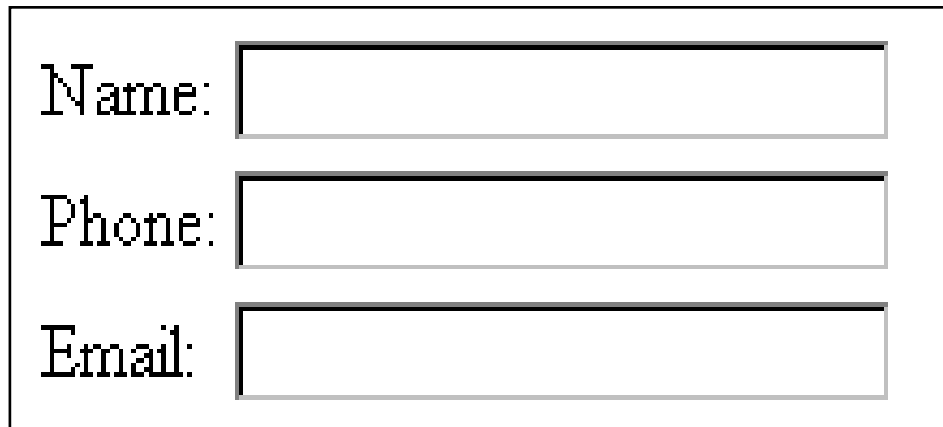
- Strings are used to represent text. enclose their content in quotes.  
    " Patch my boat with chew i n g gum "  
    ' M o n k e y s wave goodbye '
- Both single and double quotes can be used to mark strings - the quotes at the start and the end of the string should match.
- Almost anything can be put between quotes, JavaScript will make a string value out of it.

- a backslash (\) inside quoted text, indicates that the character after it has a special meaning. This is called *escaping* the character.
  - A quote that is preceded by a backslash will not end the string but be part of it.
  - When an n character occurs after a backslash, it is interpreted as a newline. Similarly,
  - a t after a backslash means a tab character.
- E.g. Take the following strings:
  - " This is the first line \ n And this is the s e c o n d "
  - " A n e w l i n e c h a r a c t e r i s w r i t t e n l i k e \" \ n \". "
- the + operator *concatenates*—glues two strings together.
  - e.g;
  - " con " + " cat " + " e " + " nate " will produce the string "concatenate":

# HTML Forms and JavaScript

- JavaScript is very good at processing user input in the web browser
- HTML `<form>` elements receive input
- Forms and form elements have unique names
  - Each unique element can be identified
  - Uses JavaScript Document Object Model (DOM)

# Naming Form Elements in HTML



A visual representation of an HTML form. It consists of a rectangular box containing three vertically stacked input fields. Each field is preceded by a label: 'Name:', 'Phone:', and 'Email:'. The input fields are empty text boxes with a thin border.

```
<form name="addressform">
```

```
Name: <input name="yourname"><br />
```

```
Phone: <input name="phone"><br />
```

```
Email: <input name="email"><br />
```

```
</form>
```



# Forms and JavaScript

`document.formname.elementname.value`

Thus:

`document.addressform.yourname.value`

`document.addressform.phone.value`

`document.addressform.email.value`

Name:

Phone:

Email:

# Using Form Data

Personalising an alert box

Enter your name:



```
<form name="alertform">
```

Enter your name:

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

```
<input type="button" value= "Go"  
      onClick="window.alert('Hello ' + →  
      document.alertform.yourname.value); ">
```

```
</form>
```

We cannot cover everything on this course but this is a reasonable introduction from which you can build upon.