
JavaScript

Session 1

What/Why/How is JS?



- JavaScript is an interpreted, **object-based** scripting language modeled after C++.
- **Interpreted languages:** an **interpreter** is needed to translate a program's statement into machine code. JavaScript Interpreter is built into Web browsers.
- **Object-based** language: most of client-side JavaScript's objects come from Web pages such as images, forms, and form elements.
- **Scripting languages:** are easy to learn, easy to use, excellent for small routines and applications, and developed to serve a particular purpose such as PERL, PHP.
- JavaScript is loosely typed language, which means that variables do not need to have a type specified.



Client-Side JavaScript sample

+

```
<html>
<head>
  <title>JavaScript sample</title>
</head>
<body>
<h1>Client-side JavaScript sample</h1>

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

+

Lexical Structure/Character Set



- The **Lexical Structure** of a programming language is the set of **elementary rules** that specifies how you write a program in that language.
- JavaScript programs are written using the **Unicode character set**.
- The 7-bit ASCII encoding is useful only for English.
- The 8-bit ISO Latin-1 encoding is useful only for English and major Western European languages.
- The 16-bit Unicode encoding can represent virtually every written language in common use on the planet.
- The **ECMAScript v3 standard** allows Unicode characters anywhere in a JavaScript program, version 1 and 2 of the standard allow Unicode characters only in comments and quoted string literals; all elements are restricted to the ASCII character set.



JavaScript is Case Sensitive



- JavaScript is a **case-sensitive** language:

```
Document.Write("Hello!");    // NOT OK
document.write("Hello!");    // OK
```

- JavaScript ignores “Whitespaces” (spaces, tabs, and newlines) that appear between token in programs.
- JavaScript automatically inserts semicolons before a line break.

```
var mytext = "abc";          // OK
var mytext = "abcd
                             efgh ";    //NOT OK – a line break inserted into a string
```

- JavaScript statements are end with semicolon (;). (optional)
- Omitting** semicolon is **not a good** programming **practice**.
- Single line comment:** any text **between a // and the end of a line is** treated as a comment and is ignored by JavaScript.
- Multiple lines comment: any text between the **characters /* and */ is also** treated as a comment.



JavaScript Syntax



Set of rules for how JavaScript programs are built

JavaScript uses most of the usual instructions and syntax that many programming languages use

Variables, Expressions, Arrays, Objects, Loops, Conditionals, Comparisons, Switches, Functions



Literals/Identifiers



- A literal is a data value that appears directly in a program.
- 12 //the number twelve (number literal)
- 1.2 //the number one point two (number literal)
- "hello world" //a string of text (string literal)
- 'Hi' //another string (string literal)
- true //a Boolean value (Boolean literal)
- null //absence of an object (special value)
- An identifier is simply a name.
- Identifiers are used to name variables and functions, and to provide labels for certain loops in JavaScript code.
- Identifier rules
- The first character must be a letter, an underscore (_).
- Subsequent characters can be a letter, a digit, an underscore.



The <script> tag



- When **Netscape** introduced JavaScript in Netscape 2, it included support for a new tag: the <script> tag.

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

- The <script> tag has several attributes, two of which you should always set: language and type.
- The language attribute specifies which scripting language you are using.

language="JavaScript/JavaScript1.1/..."

- The type attribute specifies the MIME type of the text contained within the <script> tag or the file referenced by the <script> tag. **Type="text/javascript"**
- The **src attribute**: you only need to set this attribute when you attach an external JavaScript file.
- An external JS file is just a simple text document with a **.js extension**.

- Only contain JS statements
- <script> tags are unnecessary
- You may use either a relative or an absolute path to indicate the location of your external JS file



Integrating JS into your web documents

- In the <head> of an HTML document
- In the <body> of an HTML document
- Inline with HTML as an event handler
- In an external JavaScript file
- Inline using the `javascript` pseudo-protocol



Placing JS statements in the HTML <head> section

- Is the perfect place for any statements that need to be read and **executed before the contents** of your Web document (in the <body> tag) load.
- Good place to **declare user-defined functions** (then call it from the <body> area)
- Good place to **declare global variables**



+ Placing JS statements in the `<body>` section

- This is the **best**, and only, place to write statements that actually produce content for the inclusion in an HTML document.
- Calls to functions that declared in the `<head>`



Custom Greeting

+

```
<html>
<head><title>Custom Greeting</title>
<script>
  /* Global variable */
  var visitor = prompt("What is your name?", "");
</script>
</head>

<body>
<h1>Custom Greeting</h1>

<script>
document.write("<h1>Welcome, ", visitor, "</h1>");
</script>

</body>
</html>
```

+

+ Writing JS statements inline as event handlers

- An event handler is one or more JS statements called in response to a particular event.

```
<html>
<head>
  <title>JS inline as event handler</title>
</head>

<body onload="alert('Welcome!');">

  <h1>JS inline with event handler</h1>

</body>
</html>
```



Object Oriented Concepts



- **Object:** is **an item**, thing. It has attributes/properties that describe it. It also **has methods** which are actions that you can perform with the object or on the object
- JavaScript uses **dot notation** to refer to an object and its associated properties and methods.
- For example, pen is an object
- To reference ink color property, we use the name of the object followed by a dot (period) and the name of the property.

pen.inkColor

- To change the value of an object's property.

pen.inkColor = "blue";

- To reference the object's method, we reference the name of the object first, followed by a dot and the method name, we end with a set of parentheses. The parentheses are meant to hold any arguments or parameters that provide data to or modify the performance of a method.

pen.write("Hello");



JavaScript Output



- **console.log('some value')** – Prints to console in browser or terminal
- **window.alert()** – Displays in an alert box in the browser
- **document.write()** – Display within <script> tags in the html
- **innerHTML** – Access an html element using **document.getElementById()** and output to it



Using the write method



```
<html>
<head><title>Using the write method</title>
</head>
<body>
<h1>Using the write method</h1>

<script language="JavaScript" type="text/javascript">
document.write("Hello World!!");
</script>

</body>
</html>
```



+ Using the write method to write HTML data

```
<html>
<head>
  <title>Using the write method</title>
</head>
<body>
  <h1>Using the write method to write HTML data</h1>

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

    document.write("<h1 style='text-align: center;'>Hello World!!</h1>");

  </script>

</body>
</html>
```



The document.write method



- The document's write method can accept multiple string parameters separated with commas
`document.write(string1, string2, ...)`
- The following statement would **cause an error**. You have two choices to fix it: keep the text all on one line or break the string into two strings
`document.write("long text long text long
text long text long text");`



Summary of Object Oriented Concepts

	Description	Real-world example	JavaScript example
Object	An item or thing	pen	document
Properties	An attribute that describes an object	pen.inkColor	document.backgroundColor
Method	An action that can be performed with or on an object	pen.write()	document.write()



Where does JS Objects come from



- Built into the language, like Math, String, Date and Array (Core JS)
- Come from Web documents and are made available to Client-Side JS **via the Document Object Model (DOM)**
- Come from the browser, such as navigator, location, and history objects also made available to Client-Side JavaScript by the DOM
- Programmers create our own custom objects



Variables



Variables are used to store values

JavaScript uses the “**var**” keyword to declare variables and an equal sign

to assign values

var x;

x = 100;

Same as

var x = 100;

- Variables ARE case sensitive
- Variables can contain letters, numbers, underscores (_) and dollar signs (\$)
- Variables MUST begin with a letter, underscore (_) or dollar sign (\$)

Expressions



- An expression is a combination of values, variables and operators which computes a value

$2 * 5$

$x * 5$

"Hello" + " " + "World"



Arrays



- JavaScript arrays allow us to store multiple values in a single variable

```
var names = ['Bob', 'Jim', 'Jose', 'Paula'];
```

```
console.log(names[0]);
```



Loops



Execute a block of code as long as a condition is true and repeat

For Loop:

```
for(i = 0; i < 10; i++){  
    console.log(i)  
}
```

While Loop:

```
while(i < 10){  
    console.log(i);  
    i++;  
}
```



Objects



Almost everything in JS can be considered an “object”.
Objects have properties and methods (functions)

```
// Assign Properties  
var person = {name:"Mike", age:33, hairColor:"Brown"}  
// Access Property  
console.log(person.name);  
// Accessing Method  
person.getName()
```



Conditionals / If Statements



Runs a block of code if something is true

```
var x = 10;
```

```
if(x > 5){
```

```
    console.log('Yes');
```

```
}
```

```
if(x > 5){
```

```
    } else {
```

```
    }
```

```
        console.log('Yes');
```

```
        console.log('No');
```



Function



Block of code designed to run a task. Can be created and then invoked later on

```
function sayHello(){  
    console.log('Hello World');  
}
```

```
function sayHello(greeting){  
    console.log(greeting);  
}
```



How to run JavaScript?

JavaScript in URL (JavaScript Pseudo-Protocol)

- copy the following JS code and paste it in the URL, then hit the enter key.

```
javascript: var now=new Date(); document.write("<h1> the time is: </h1>" + now);
```



JavaScript inside the <body> section

- Directly code the JS inside the body element. (Normally use the document.write() function to display the output)
- Inside the <body> element:

```
<script language="JavaScript" type="text/javascript">  
var a = 10;  
document.write("Resut is: " + a);  
</script>
```



Call a JavaScript function

- From the body element, directly call a function defined inside the head element.
(Normally use the alert() function to display output)
- Inside the head element: (defines the function)

```
<script language="JavaScript" type="text/javascript">  
function f1() {  
  var a = 10;  
  alert("Result is " + a);  
}  
</script>
```

- Inside the body element: (call the function)

```
<a href="javascript: f1();" > Click to run JS</a>
```

Exercise: Greeting card

+

```
<head>
<script language="JavaScript" type="text/javascript">
    var YourName = prompt("May I know your name?", "");
</script>
</head>
<body>
<section>
<h1>Custom Greeting</h1>
<script language="JavaScript" type="text/javascript">
document.write("<p style='text-align: center;'>Hello, ", YourName);
</script>
</section>
<footer>
```

+