JavaScript

4.0 Introduction

WHY
JavaScript is an interpreted scripting language. Every collection of JavaScript code is nothing but a sequence of lines which may be terminated with a semicolon. This is called a *script.* A JavaScript document can have any number of embedded scripts.
JavaScript is a case-sensitive *scripting* language - which means that it's not compiled (unlike Java), it is executed at run-time.
Several lines put together forms a script. A block of code is delimited using curly braces {}.

JavaScript and Java
Although JavaScript and Java seem to denote a close relationship, they are actually very different. The most basic difference is that JavaScript does not support the object-oriented software development paradigm. Java is a strongly typed language while JavaScript variables are dynamically typed (more about this later). Essentially, JavaScript objects are dynamic - the number of data members and methods of an object can change during execution.

HOW
JavaScript can be used inline, in a <style> element, or in a separate file.
You may write your code inline but this isn't very helpful as you have to write the entire code out fully, for every button. A better way to write scripts is via stand alone functions in the header tags (`<head>...</head>`). You may also write scripts in a separate file.

TRY IT
Here's an example of an inline script. This creates a 'Button' element and throws a pop up when clicked.

```
<input type ="button" id="button1" value="Press me!"
onclick="alert('You have clicked the button!’);" />
```

onclick refers to the action that needs to happen when this button is clicked.

> WARNING: Event attributes should be not be used. They mix up HTML with JavaScript, making documents harder to read and debug, and larger in size.

Here’s how you’d write the same JavaScript code in a style element.

```
<head>
  <script>
      function doSomething() {
            alert('You have clicked the button!’) ;
      }
      var button1 = document.getElementById('button1');
      button1.onclicked = doSomething;
  </script>
</head>

<input type ="button" id="button1" value="Press me!" />
```

This is similar in functionality to what we saw above, except that it’s a more modular way of writing code. Needless to say, this is the preferred method!

4.1 Element Access in JavaScript

WHY

There are several ways the object associated with an XHTML form element can be addressed in JavaScript. The Document Object Model (DOM) way is to use `Forms` and `Element` arrays of the `Document` object, which is referenced by the `Window` object.
To reiterate, the `Window` object is the window in which the XHTML content is displayed. Every `Window` has a property named `Document`. Every `Document` object has a `Forms` array, and each element represents a form of the document. The `Forms` array element has an `Element` array as a property which contains the objects that represent the XHTML form elements like buttons, menus etc.

HOW

The elements of a document are objects, with both data and operations. The data are called properties, and the operations are called methods.
Look at this example:

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

Here, `type` and `name` are the properties of an object and 'text’ and 'age’ are the values.

The DOM address is the address of the JavaScript object. You can access deeper elements in the DOM tree by using the dot property. The name or unique id of the element is used to refer to it. This can be done using methods like getElementByName() or getElementById().

Let's look at this example:

```
var dom = document.getElementById("button1");
```

This line of code is used to access the element whose unique identification name or id is *button1*, which is a part of the document.

TRY IT

<u>To count the number of ticked checkboxes on a form:</u>

This script creates an implicit array of 3 items, *colors,* via a checkbox. All the elements have the same name and id, which makes them easy to access later in the script. Individual addresses to the elements are stored using array indexes which start from 0 and go on till length-1.

Since the input type is checkbox, each element will also have an additional property called `'checked'`. The `checked` property is set to true if the button is checked and false if it is not.

```
<form id="FavoriteColor">
  <input type="checkbox" name="color"
      value="red" /> Red
  <input type="checkbox" name="color"
      value="blue" /> Blue
  <input type="checkbox" name="color"
      value="green" /> Green
</form>
```

```
var numChecked = 0'
var dom = document.getElementById("FavoriteColor");
for (i = 0; i < dom.color.length; i++)
      if (dom.car[i].checked)
            numChecked++;
```

4.2 Variables, Operators and Comments

4.2.1 Variables

WHY

Simply put, variables hold values. Before using a variable, you have to declare it.
JavaScript is a dynamically typed language, which means that a variable can be used for anything - a value or a reference to an object.
If a value is assigned to a variable, it automatically takes the type of that value.
JavaScript types are split into 2 groups:
Primitive types: Number, String, Boolean, Undefined and Null

Reference types: Objects, Arrays and Functions

Important things to note about variables:

- JavaScript is case sensitive, unlike HTML and CSS
- Keywords (like var, document etc) cannot be used as variable names
- Default value (if unspecified) of a variable is Undefined
- Strings have to be declared within quotes

Scope of a variable:

- Local variables:
  When a variable is declared within a function, it is only accessible in that function and will not be valid once the function closes.
- Global Variables
  When a variable is declared in the top level of the script file (outside a function), it is accessible anywhere within the JavaScript program.
- Undeclared variables are automatically global variables

HOW

A variable can be assigned a value by either assigning it a value (in which case the interpreter implicitly declares it to be a variable), or by listing it in a declaration statement that begins with the keyword `var`.

TRY IT

```
var counter,
    pi = 3.14,
    flag = true,          // Boolean variable
    shape = “circle";     //String variable
```

4.2.2 Operators

WHY

There are different functions that need to be performed on values like addition, subtraction etc. These functions are executed using Operators. Some operators like + may also be used on Strings for concatenation.

HOW

There are different kinds of Operators in JavaScript.

- Arithmetic
  - This performs the basic arithmetic functions.

+	Addition
-	Subtraction
*	Multiplication
/	Division

%	Remainder function
++,--	Increment/decrement a variable (unary operator)

+ may also be used as a concat operator.

```
var a = "Hello" + "World"
```

This means that *a* will contain *Hello World.*

- Equality
  - This is used to compare 2 values and types

=	Assignment operator `var a = 2` Assignes the value 2 to the variable a
==	Equality operator `var a = "2" == 2;` //true This checks the types and the values of the two variables

- Comparison
  - This is used to compare 2 values. It may also be used to compare Strings.

< , >	Less than, Greater than
<= , >=	Less than or equal to, Greater than or equal to

4.2.3 Comments

WHY

Comments are human readable lines in the code, which are not executed by the compiler. They do not need to follow any specific format. It is added to the code only to help the programmer identify what the block of code is meant to do.

HOW

Single line comments have to be written with //
Multi-line comments need to be within /* and */

TRY IT

```
var a = 2 ;          //This is a comment. It should be in a single line.
var b = "Hello" ;    /*This is a multi-line comment.
It can spread across more than one line.*/
```

4.3 Functions, Objects and Events

4.3.1 Functions

WHY

A function is a stand-alone module of code that performs a particular action.
This enhances reusability of your code, especially if you have to perform a particular action repeatedly like addition, asking for user input etc.

HOW

A function definition consists of a function *header* and a set of compound statements known as the *body*.
The function header begins with the reserved word (keyword) function. It also has a unique name which describes the function's purpose. The body consists of statements which then perform the actions determined by the function.
JavaScript functions are objects, so variables that reference them can be treated as other object references.
Functions can also have parameters. Parameters are the variables in a function header which can contain values. When a function is called, *arguments* can be passed to the function which means that values can be passed as an input to the function. This can then be used within the function. An important point to note is that function parameters are passed using the call-by-value method. This means that only the value (and not the object's reference) is passed to the function as input.

```
function functionName(var1, var2) {
      /* function specific code here
      …
      */
      Return returnVal; // optional return value
}
```

TRY IT

Write a function to accept 2 Strings from the user and then return it as one concatenated String.

```
function returnConcatString(var1 var2) {
      var result = "";            //initialising a blank string for
output
      for (var i = 0; i < arguments.length; i++) {
            result = result + " " + arguments[i];
      }
      return result;
}

var b = returnConcatString("Hello", "World");
```

b will contain *Hello World.*

4.3.2 Objects

WHY

An object is a collection of properties. If a property's value is a function, then it's called a method.
An object essentially could consist of 2 things: a property and a method.
To make it simpler, all objects of the same type have the same properties but their property values could differ.

HOW

An object is created by defining the properties and methods associated with it. You can access the properties and methods using the dot operator (.)
Properties of an object could be accessed like this:

```
objectName.propertyName ;
objectName["propertyName"] ;
```

Methods of a property could be accessed like this:

```
objectName.methodName() ;
```

An empty object is declared using a constructor like this:

```
var car = new Object () ;
```

After you create the object, properties could be initialised.

```
car.type = "Sedan";
car.color = "red";
```

TRY IT

Create an object of type car with various properties like type, color and model.

```
var car = {type: 'Sedan',
           color: 'red',
           Model: 'Fiat};
```

4.3.3 Events

WHY

An event is a user action or a notification that something has happened. JavaScript code interacts with HTML pages via events. For example, clicking a button, moving your mouse over some text etc are all examples of events.

HOW

An event handler is a script that is implicitly called when an event occurs. Every element has an associated event attribute. Several elements can have the same event attributes as well. This means that an event like `onclick` will have the same actions on a link (anchor tag) and a text box.

Here are some events and their tag attributes:

EVENT	TAG ATTRIBUTE
blur	onblur
click	onclick
dblclick	ondblclick
focus	onfocus
keydown	onkeydown
keypress	onkeypress
keyup	onkeypress
load	onload
mousedown	onmousedown
mousemove	onmousemove
mouseover	onmouseover
reset	onreset
select	onselect
submit	onsubmit
unload	onunload

TRY IT
Produce an alert message when the body of the document has been loaded.

```
<html>
<head>
      <title> load.html </title>
      <script type = "text/javascript"  src = "load.js">
      </script>
</head>
<body onload = load_greeting();">
      <p> Page has loaded! </p>
</body>
</html>

//load.js
function load_greeting() {
```

```
     alert("This is a pop up message with a greeting - Hi! Welcome
to my page!");
}
```

4.4 Strings, Numbers, Math and Date Functions

4.4.1 Strings

WHY

JavaScript Strings are not stored or treated as arrays of characters, instead they are unit scalar values.

HOW

Strings can have several functions or operations that can be performed.
String methods can be used through String primitive values, as if the values were objects.
For example, the number of characters in a String is stored in the `length` property.

A few of the common String methods are shown in the table below.

Method	Parameter	Result
length	None	Returns the number of characters in a String object
charAt	A number	Returns the character in the String object at the specified position
indexOf	One-character String	Returns the position of the parameter in the String object
substring	Two Numbers	Returns the substring of the String object from the first parameter position to the second parameter
toLowerCase	None	Converts any uppercase letters to lowercase
toUpperCase	None	Converts any lowercase letters to uppercase

In addition to these methods, certain character sequences called escape characters are also used.

\n	New line
\t	Tab (8 spaces)
\r	Carriage return

TRY IT

```
var str = "Hello World" ;

str.charAt(2);          // output is l
str.indexOf('o');       // output is 4
str.substring(2,4);     // output is llo
str.toLowerCase();      // output is hello world
```

4.4.2 Numbers

WHY

The Number object includes a collection of useful properties that have constant values.

HOW

The following table represents the number properties.

Property	Meaning
MAX_VALUE	Largest representable number
MIN_VALUE	Smallest representable number
NaN	Not a number
POSITIVE_INFINITY	Special value to represent infinity
NEGATIVE_INFINITY	Special value to represent negative infinity
PI	The value of pi (3.14)

4.4.3 Math Function

WHY

The Math object has methods for the trigonometric functions such as sinc and cosine. It also has methods for other commonly used mathematical functions.

HOW

The Math function has trigonometric functions like sin (sine) and cos (cosine) as well as certain mathematical functions like `floor` which truncates a number; `round` which rounds off the number to the nearest whole value; `max` which returns the largest of two numbers etc. All Math functions are referenced using the Math object.

TRY IT

```
Math.floor(4.2343);          //Output is 4
```

4.4.4 Date Function

WHY

The Date function is used to represent the date and time.

HOW
The Date object is referenced using the Date object.
It can be created using this statement:

```
var today = new Date() ;
```

Here are some commonly used methods of the Date object.

Method	Returns
toLocaleString	A String of the Date information
getDate	The day of the month
getMonth	The month of the year (0 to 11)
getDay	The day of the week (0 to 6)
getFullYear	The year
getTime	The number of milliseconds since 01-01-1970
getHours	The number of the hour (0 to 23)
getMinutes	The number of the minute (0 to 59)
getSeconds	The number of the second (0 to 59)

4.5 Arrays

WHY
An array is a collection of values, referenced by the same name. Each array element is accessed via its index. Indices begin from 0 to length-1.
An important point to note in JavaScript, unlike other programming languages, is that arrays can have mixed type. This means that all elements in an array need not have the same type.

HOW
Array objects, unlike most other JavaScript objects, can be created in 2 ways.
The first method is the usual method of creating an object, using the *new* operator.

```
var list = new Array (1, 2, “three", “four") ;
```

The second method is using a literal array value.

```
var list_2 = [1, 2, “three", “four"];
```

The lowest index of any array value will be zero.
Array elements are accessed using the numeric value of the index as a subscript to the array name.
The length of an array is the number of elements in an array. Therefore, the last index of an array will always be length-1.

Arrays have several methods associated with them. Some common ones are listed below.

Method	Purpose
join	Converts all the elements of an Array object to a String and concatenates them
reverse	Reverses the order of elements in an Array object
sort	Converts all elements of an Array object to a String and sorts them alphabetically
concat	Adds the actual parameters of the concat method to the end of the Array object
slice	Returns part of the Array object that has been specified as parameters
pop	Removes an element from the high end of an Array object
push	Adds an element to the high end of an Array object

TRY IT
Create an array object and display the values. The add more elements to the array and display the new values.

```
//array_example.js
var list = [2, 4, 6];

for (var i = 0; i < list.length; i++) {
      document.write(list[i] + "  ");
}
document.write("<br />");

var new_list = list.concat(8, 10);

for (var j = 0; j < new_list.length; j++) {
      document.write(new_list[j] + "  ");
}
```

Output will be:
2 4 6
2 4 6 8 10

4.6 Booleans, Conditions and Loops

4.6.1 Booleans

WHY

Boolean is a data type in JavaScript. It can have only 2 values:
True and False.

HOW
Boolean objects are declared like any other objects in JavaScript.

```
var bool_t = new Boolean() ;
bool_t = true;
```

4.6.2 Conditions

WHY
Conditional statements are important because they determine the flow of execution of statements. They always take parameters which return Boolean values. If the boolean value is true, then the block of statements immediately following the condition is executed.
They depend on Logical operators.

HOW
The expressions upon which statement flow control can be based include primitive values, relational expressions and compound expressions. The result of a control expression is either True or False.
JavaScript also has operators for AND, OR, and NOT Boolean operations. These are && (AND), || (OR) and ! (NOT).

There are two types of Conditional Statements:

1. Selection Statements
   This depends on the if-then and if-then-else construct.
   The syntax is as follows:

```
if (condition) {
/* if the condition evaluates to true,
      execute these statements
…
*/  }
else {
/* if the condition evaluates to false,
      evaluate these statements
…
*/  }
```

2. The switch Statement
   This may be used when there are several conditions that need to be checked and only one may match. The syntax is as follows:

```
switch (expression) {
      case value_1:         //statement(s)
      case value_2:         //statement(s)
      ...
      case value_n:         //statement(s)
      [default:        //statement(s)]
}
```

The expression in the switch construct is evaluated and the value is compared to the values in the cases in the construct. If one matches, the control is transferred to the statements following the case value. Execution then continues for the remainder of the construct.
In most cases, only one case statement needs to be executed, after which the control needs to be transferred the statements immediately outside the switch construct.
In that case, a *break* statement might be used.
The *break* statement transfers control out of the compound statement in which it appears.

TRY IT
Write a simple if-else construct to compare two numbers and output the greater number.

```
//if_then.js

var a = 5;
var b = 8;

if (a > b)
   document.write("a is greater than b <br/> ");
else
   document.write("a is lesser than b <br/> ");
```

4.6.3 Loops

WHY
Loops are used primarily to perform the same task over several elements in an array or list. Essentially, it loops through a block of code several times.

HOW
Loops usually have a control expression, followed by a block of code in parenthesis. There are two kinds of loops:

1. While Loops
   While loops keep executing a set of lines of code until the control expression no longer holds true. The syntax is as follows:

```
while (conditional statement)
{
      /* set of statements
      …
      */
}
```

2. For Loops
   For loops have an initial value, a conditional value and an increment step. The initial value of the variable keeps getting incremented every time the block of statements is executed. Once the conditional value is no longer true, the loop ends. The syntax is as follows:

```
      for (initial expression; control expression; increment
expression)
      {
                  /* set of statements
                  …
                  */
            }
```

Both the loops can terminate on demand using the break statement.

TRY IT

Write a JavaScript code to find the product of numbers from 1 to 50.

```
//loops_while.js

var i = 1;
var sum = 0;

while ( i <= 50) {
sum = sum * i ;
}

document.write("The product is" + sum);

//loops_for.js
var sum = 0;
for (var i = 0; i < 51; i++)
sum = sum * i;

document.write("The product is" + sum);
```

4.7 Regular Expressions (RegEx)

WHY

JavaScript has powerful pattern matching capabilities based on regular expressions. Although the `search` function of String objects can be used, RegEx provides much more comprehensive functionality.

HOW

The 'normal' characters are those that are not metacharacters i.e., they do not have special meanings depending on the situation. Special characters or metacharacters include:

```
      \ | ( ) [ ] { } ^ $ * + ? .
```

Metacharacters are matched in regular expressions by immediately preceding them with a backslash (\).

A period (.) matches any character except a newline. As stated above, to match a period in a string, it has to be preceded with a backslash.
Example:
`/snow./` will match “snowy", “snowed", “snows" etc

If a circumflex character (^) is the first character in a regular expression, it negates or inverts the specified set.
Example:
[^abc] will match all characters except 'a’, 'b’ and 'c’

A hyphen (-) specifies a range.

The following table gives more details about predefined classes

Name	Equivalent Pattern	Matches
`\d`	`[0-9]`	A digit
`\D`	`[^0-9]`	Not a digit
`\w`	`[A-Za-z_0-9]`	An alphanumeric character
`\W`	`[^A-Za-z_0-9]`	Not an alphanumeric character
`\s`	`[ \r\t\n\f]`	A whitespace character
`\S`	`[^ \r\t\n\f]`	Not a whitespace character

TRY IT
[\d\.\d\d] //Matches a digit, followed by a period and then 2 digits
[\D\d\D] //Matches a single digit
[\d+\.\d*] //Matches one or more digits followed by a period and then 0 or more digits
[\w\w\w] //Matches 3 adjacent word characters