Government Arts and Science College (Women), Sathankulam – 628704

Class : II B.Sc. (Computer Science)

Semester : IV

Subject : Web Technology

Topic : Unit III

Faculty : Mrs. A. Angeline Nancy Sophia M.Sc., M.Phil.,
Department of Computer Science

JAVA SCRIPT

- Java Script was designed to provide a quicker and simpler language for enhancing web pages and web servers.
- JavaScript is a dynamic computer programming language.
- It is 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.
- Simple statements in JavaScript are generally followed by a semicolon character
- JavaScript, however, allows you to omit this semicolon if each of your statements are placed on a separate line
- JavaScript is a case-sensitive language
- There is a flexibility given to include JavaScript code anywhere in an HTML document.
- Script in <body>...</body> and <head>...</head> sections.

- JavaScript in <head>...</head> section

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

- JavaScript in <body>...</body> section

```
<html>
<head>
</head>
```

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

Variables

- Variables are declared with the **var** keyword

  <script type = "text/javascript">

  <!-- var money; var name; //--> </script>

- Storing a value in a variable is called **variable initialization**

  <script type = "text/javascript">

  <!-- var name = "Ali"; var money; money = 2000.50; //--> </script>

- 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.

LANGUAGE ELEMENTS

- Identifiers
- Expressions
- JavaScript keywords
- Operators
- Statements
- Functions

Identifiers

- It is an unique term to identify a variable, method or an object.
- It can be literals and variables
- Literals are identifiers having fixed values
- Variables have different values during execution.
- There are several types of data. They are integers, floating point numbers, Strings and Boolean

Expressions

- An expression is a statement that is evaluated to a value.
- The result may be any type.
- Expressions contains variables, literals, operators and other expressions.
- Example:

```
Y=67;
Str="Good Morning";
Val= x + y * b;
```

Keywords

- The language has a set of keywords reserved for a specific purpose.
- They cannot be used as variables or constants.
- Example:

abstract boolean break byte case catch

Operators

- Operators are commands which perform operations on variables and or literals and produce a result.
- There are two types of operator: unary, binary.
- Unary operate on only one operand.
- Binary operator operate on two operands.
- Types of operator:
    - Arithmetic operators
    - Comparison operators
    - Logical operators

Arithmetic operators

+ (Addition)

Adds two operands

Ex: A + B will give 30

- (Subtraction)

Subtracts the second operand from the first

Ex: A - B will give -10

*** (Multiplication)**

Multiply both operands

Ex: A * B will give 200

/ (Division)

Divide the numerator by the denominator

Ex: B / A will give 2

% (Modulus)

Outputs the remainder of an integer division

Ex: B % A will give 0

++ (Increment)

Increases an integer value by one

Ex: A++ will give 11

-- (Decrement)

Decreases an integer value by one

Ex: A-- will give 9

Example

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

Output

```
a + b = 43
a - b = 23
a / b = 3.3
a % b = 3
a + b + c = 43
```

++a = 35

--b = 8

Comparison operators

= = (Equal)

Checks if the value of two operands are equal or not, if yes, then the condition becomes true.

Ex: (A == B) is not true.

!= (Not Equal)

Checks if the value of two operands are equal or not, if the values are not equal, then the condition becomes true.

Ex: (A != B) is true.

> (Greater than)

Checks if the value of the left operand is greater than the value of the right operand, if yes, then the condition becomes true.

Ex: (A > B) is not true.

< (Less than)

Checks if the value of the left operand is less than the value of the right operand, if yes, then the condition becomes true.

Ex: (A < B) is true.

>= (Greater than or Equal to)

Checks if the value of the left operand is greater than or equal to the value of the right operand, if yes, then the condition becomes true.

Ex: (A >= B) is not true.

<= (Less than or Equal to)

Checks if the value of the left operand is less than or equal to the value of the right operand, if yes, then the condition becomes true.

Ex: (A <= B) is true.

Example

```
<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("(a < b) => ");
result = (a < b); document.write(result);
```

```
 document.write("(a > b) => ");
result = (a > b); document.write(result);
 document.write("(a != b) => ");
result = (a != b); document.write(result);
document.write("(a >= b) => ");
result = (a >= b); document.write(result);
document.write("(a <= b) => ");
result = (a <= b); document.write(result);  //--> </script> </body> </html>
```

Output

```
(a == b) => false
(a < b) => true
(a > b) => false
(a != b) => true
(a >= b) => false
(a <= b) => true
```

Logical operators

&& (Logical AND)

If both the operands are non-zero, then the condition becomes true.

Ex: (A && B) is true.

|| (Logical OR)

If any of the two operands are non-zero, then the condition becomes true.

Ex: (A || B) is true.

! (Logical NOT)

Reverses the logical state of its operand. If a condition is true, then the Logical NOT operator will make it false.

Ex: ! (A && B) is false.

Example

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

```
 document.write("!(a && b) => ");
result = (!(a && b)); document.write(result);
//--> </script> </body> </html>
```

Output

```
(a && b) => false
(a || b) => true
!(a && b) => true
```

String operators

- For concatenating two strings '+' is used.
  str = "Good" + "day";

Statements

if statement

Syntax

```
if (expression) {
        Statement(s) to be executed if expression is true }
```

If the resulting value is true, the given statement(s) are executed. If the expression is false, then no statement is executed.

```
<html>
 <body>
 <script type = "text/javascript">
 <!-- var age = 20;
if( age > 18 ) { document.write("<b>Qualifies for driving</b>");
} //--> </script>
 <p>Set the variable to different value and then try...</p>
 </body>
</html>
```

Output

Qualifies for driving

Set the variable to different value and then try...

if...else statement

The '**if...else**' statement allows JavaScript to execute statements in a more controlled way.

Syntax

```
if (expression) { Statement(s) to be executed if expression is true
}
else { Statement(s) to be executed if expression is false
}
```

If the resulting value is true, the given statement(s) in the 'if' block, are executed. If the expression is false, then the given statement(s) in the else block are executed.

```
<html>
 <body>
<script type = "text/javascript">
 <!-- var age = 15;
if( age > 18 ) { document.write("<b>Qualifies for driving</b>");
} else { document.write("<b>Does not qualify for driving</b>");
} //--> </script>
 <p>Set the variable to different value and then try...</p> </body> </html>
```

Output

Does not qualify for driving

Set the variable to different value and then try...

if...else if... statement

Syntax

```
if (expression 1) {
   Statement(s) to be executed if expression 1 is true
} else if (expression 2) {
   Statement(s) to be executed if expression 2 is true
} else if (expression 3) {
   Statement(s) to be executed if expression 3 is true
} else {
   Statement(s) to be executed if no expression is true
}
```

Statement(s) are executed based on the true condition, if none of the conditions is true, then the **else** block is executed.

```
<html>
 <body>
 <script type = "text/javascript">
 <!-- var book = "maths";
if( book == "history" ) {
     document.write("<b>History Book</b>");
} else if( book == "maths" ) {
    document.write("<b>Maths Book</b>");
} else if( book == "economics" ) { document.write("<b>Economics Book</b>");
} else {
    document.write("<b>Unknown Book</b>"); } //--> </script>
<p>Set the variable to different value and then try...</p> </body>
<html>
```

Output

Maths Book

Set the variable to different value and then try...

Switch Statement

To perform a multiway branch we use a **switch** statement

Syntax

```
switch (expression) {
                case condition 1: statement(s)
                break;
                case condition 2: statement(s)
                break;
                ...
                case condition n: statement(s)
                break;
                default: statement(s)
}
```

```
<html>
 <body>
<script type = "text/javascript">
 <!-- var grade = 'A';
   document.write("Entering switch block<br />");
   switch (grade) {
```

```
       case 'A': document.write("Good job<br />");
       break;
       case 'B': document.write("Pretty good<br />");
       break;
       case 'C': document.write("Passed<br />");
       break;
       case 'D': document.write("Not so good<br />");
       break;
       case 'F': document.write("Failed<br />");
       break;
      default: document.write("Unknown grade<br />")
}
     document.write("Exiting switch block"); //--> </script>
 <p>Set the variable to different value and then try...</p> </body> </html>
```

Output

Entering switch block
Good job
Exiting switch block
Set the variable to different value and then try...

The while Loop

while loop is to execute a statement or code block repeatedly as long as an **expression** is true. Once the expression becomes **false,** the loop terminates.

Syntax

```
while (expression) {
      Statement(s) to be executed if expression is true
}
```

```
<html>
<body>
 <script type = "text/javascript">
 <!-- var count = 0;
document.write("Starting Loop ");
while (count < 10) {
    document.write("Current Count : " + count + "<br />"); count++;
} document.write("Loop stopped!"); //--> </script>
 <p>Set the variable to different value and then try...</p>
```

</body> </html>

Output

Starting Loop
Current Count : 0
Current Count : 1
Current Count : 2
Current Count : 3
Current Count : 4
Current Count : 5
Current Count : 6
Current Count : 7
Current Count : 8
Current Count : 9
Loop stopped! Set the variable to different value and then try...

The do...while Loop

The **do...while** loop is similar to the **while** loop except that the condition check happens at the end of the loop.

Syntax

```
do {
Statement(s) to be executed;
} while (expression);
```

```
<html>
 <body>
<script type = "text/javascript">
 <!-- var count = 0;
    document.write("Starting Loop" + "<br />");
   do {
    document.write("Current Count : " + count + "<br />"); count++;
} while (count < 5);
document.write ("Loop stopped!"); //--> </script>
<p>Set the variable to different value and then try...</p> </body> </html>
```

Output

Starting Loop
Current Count : 0
Current Count : 1

Current Count : 2
Current Count : 3
Current Count : 4
Loop Stopped!
Set the variable to different value and then try...

For Loop

The '**for**' loop is the most compact form of looping. It includes the following three important parts −

- The **loop initialization** where we initialize our counter to a starting value. The initialization statement is executed before the loop begins.
- The **test statement** which will test if a given condition is true or not. If the condition is true, then the code given inside the loop will be executed, otherwise the control will come out of the loop.
- The **iteration statement** where you can increase or decrease your counter.

Syntax

```
for (initialization; test condition; iteration statement) {
        Statement(s) to be executed if test condition is true
}
```

```
<html>
<body>
 <script type = "text/javascript">
<!-- var count;
document.write("Starting Loop" + "<br />");
for(count = 0; count < 10; count++) { document.write("Current Count : " + count );
document.write("<br />");
} document.write("Loop stopped!"); //--> </script>
 <p>Set the variable to different value and then try...</p> </body> </html>
```

Output

Starting Loop
Current Count : 0
Current Count : 1
Current Count : 2
Current Count : 3
Current Count : 4
Current Count : 5

Current Count : 6
Current Count : 7
Current Count : 8
Current Count : 9
Loop stopped!
Set the variable to different value and then try...

The **for...in** loop is used to loop through an object's properties.
Syntax

```
for (variablename in object) {
    statement or block to execute
}
```

In each iteration, one property from **object** is assigned to **variablename** and this loop continues till all the properties of the object are exhausted.

```
<html>
<body>
 <script type = "text/javascript">
 <!-- var aProperty;
   document.write("Navigator Object Properties<br /> ");
for (aProperty in navigator) {
    document.write(aProperty);
    document.write("<br />");
} document.write ("Exiting from the loop!"); //--> </script>
 <p>Set the variable to different object and then try...</p> </body> </html>
```

Output

Navigator Object Properties
serviceWorker
webkitPersistentStorage
webkitTemporaryStorage
geolocation
doNotTrack

FUNCTIONS

- A function is a group of reusable code which can be called anywhere in your program.
- This eliminates the need of writing the same code again and again.

- It helps programmers in writing modular codes.
- Functions allow a programmer to divide a big program into a number of small and manageable functions.

Function Definition

Before we use a function, we need to define it. We 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>
<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>
```

Function Parameter and the return Statement

A JavaScript function can have an optional **return** statement. This is required if you want to return a value from a function. This statement should be the last statement in a function.

```
<html>
 <head>
 <script type = "text/javascript">
function concatenate(first, last) {
```

```
var full; full = first + last; return full;
}
function secondFunction() {
var result;
result = concatenate('Zara', 'Ali');
document.write (result );
} </script>
</head> <body>
<p>Click the following button to call the function</p>
<form> <input type = "button" onclick = "secondFunction()" value = "Call
Function"> </form>
<p>Use different parameters inside the function and then try.</p> </body>
 </html>
```