AWD-03-JavaScript

In HTML, JavaScript code is inserted between `<script>` and `</script>`

```
<!DOCTYPE html>
<html>
    <body>
        <h2>JavaScript in Body</h2>
        <p id="demo"></p>
    <script>
        document.getElementById("demo").innerHTML = "My First JavaScript";
    </script>
    </body>
</html>
```

Output Methods

Using `innerHTML`:

- To access an HTML element, JavaScript can use the `document.getElementById(id)` method.
- The `id` attribute defines the HTML element. The `innerHTML` property defines the HTML content

```
<!DOCTYPE html>
<html>
    <body>
        <h1>My First Web Page</h1>
        <p>My First Paragraph</p>
        <p id="demo"></p>
    <script>
        document.getElementById("demo").innerHTML = 5 + 6;
    </script>
    </body>
</html>
```

```
My First Web Page
My First Paragraph
11
```

Using `document.write()`:

- For testing purposes, it is convenient to use `document.write()`

```
<!DOCTYPE html>
<html>
    <body>
        <h1>My First Web Page</h1>
        <p>My first paragraph</p>
    <script>
        document.write(5 + 6);
    </script>
    </body>
</html>
```

```
My First Web Page
My first paragraph
11
```

Using `window.alert()`:

- You can use an alert box to display data
- In JavaScript, the window object is the global scope object.
- This means that variables, properties, and methods by default belong to the window object.
- This also means that specifying the `window` keyword is optional

```
<!DOCTYPE html>
<html>
    <body>
        <h1>My First Web Page</h1>
        <p>My first paragraph.</p>
    <script>
        window.alert(5 + 6);
        //OR
        alert(5 + 6);
    </script>
    </body>
</html>
```

```
My First Web Page
My first paragraph
11
```

Using `console.log()`:

- For debugging purposes, you can call the `console.log()` method in the browser to display data.

```
<!DOCTYPE html>
<html>
    <body>
    <script>
        console.log(5 + 6);
    </script>
    </body>
</html>
```

JavaScript Print

- JavaScript does not have any print object or print methods.
- You cannot access output devices from JavaScript.
- The only exception is that you can call the `window.print()` method in the browser to print the content of the current window.

```
<!DOCTYPE html>
<html>
    <body>
        <button onclick="window.print()">Print this page</button>
    </body>
</html>
```

Statements

- A computer program is a list of "instructions" to be "executed" by a computer.
- In a programming language, these programming instructions are called statements.
- A JavaScript program is a list of programming statements.
- JavaScript statements are composed of: Values, Operators, Expressions, Keywords, and Comments.
- This statement tells the browser to write "Hello World." inside an HTML element with id="demo":

```
<!DOCTYPE html>
<html>
    <body>
        <h2>JavaScript Statements</h2>
        <p>In HTML, JavaScript statements are executed by the browser.</p>
        <p id="demo"></p>
    <script>
        document.getElementById("demo").innerHTML = "Hello World.";
    </script>
    </body>
</html>
```

```
JavaScript Statements
In HTML, JavaScript statements are executed by the browser.
Hello World.
```

Variables

- Variables are Containers for Storing Data
- **Block Scope:**
    - JavaScript had Global Scope and Function Scope.
    - ES6 introduced the two new JavaScript keywords: `let` and `const`.
    - These two keywords provided Block Scope in JavaScript

    - Variables declared inside a { } block cannot be accessed from outside the block

```
{
    let x = 2;
}
// x can NOT be used here
```

- **Global Scope**:
    - Variables declared with the `var` always have Global Scope.
    - Variables declared with the `var` keyword can NOT have block scope:
    - Variables declared with `var` inside a { } block can be accessed from outside the block

```
{
    var x = 2;
}
// x CAN be used here
```

Using `var`

- The `var` keyword was used in all JavaScript code from 1995 to 2015.
- The `let` and `const` keywords were added to JavaScript in 2015.
- Variables defined with `var` can be redeclared
- The `var` keyword should only be used in code written for older browsers.

```
<body>
    <h1>JavaScript Variables</h1>
    <p>In this example, x, y, and z are variables.</p>
    <p id="demo"></p>
    <script>
        var x = 5;
        var y = 6;
        var z = x + y;
        document.getElementById("demo").innerHTML =
        "The value of z is: " + z;
    </script>
</body>
```

```
JavaScript Variables
In this example, x, y, and z are variables.
The value of z is: 11
```

Using `let`

- The `let` keyword was introduced in ES6 (2015)
- Variables declared with `let` have Block Scope
- Variables declared with `let` must be Declared before use

- Variables declared with `let` cannot be Redeclared in the same scope

```
<!DOCTYPE html>
<html>
<body>
    <h2>Redeclaring a Variable Using let</h2>
    <p id="demo"></p>
    <script>
        let  x = 10;
        // Here x is 10
        {
          let x = 2;
          // Here x is 2
        }
        // Here x is 10
        document.getElementById("demo").innerHTML = x;
    </script>
</body>
</html>
```

```
Redeclaring a Variable Using let
10
```

Using `const`

- The `const` keyword was introduced in ES6 (2015)
- Variables defined with `const` cannot be Redeclared
- Variables defined with `const` cannot be Reassigned
- Variables defined with `const` have Block Scope
- Variable defined with the `const` keyword cannot be reassigned
- Always declare a variable with `const` when you know that the value should not be changed.
- Use `const` when you declare:
    - A new Array
    - A new Object
    - A new Function
    - A new RegExp

```
<!DOCTYPE html>
<html>
    <body>
        <h2>JavaScript const</h2>
        <p id="demo"></p>
    <script>
        const PI = 3.141592653589793;
        PI = 3.14;
        document.getElementById("demo").innerHTML = PI;
    </script>
    </body>
</html>
```

```
JavaScript const
3.14
```

Operators

- JavaScript operators are used to perform different types of mathematical and logical computations.
- There are different types of JavaScript operators:
  - Arithmetic Operators
  - Assignment Operators
  - Comparison Operators
  - Logical Operators

Arithmetic Operator

- Arithmetic Operators are used to perform arithmetic operations on numbers

```
<p id="demo"></p>
<script>
    let a = 3;
    let x = (100 + 50) * a;
    document.getElementById("demo").innerHTML = x;
</script>
```

```
450
```

Operator	Description
+	Addition
-	Substraction
*	Multiplication
/	Division
**	Exponentiation
%	Modulus (Remainder)
++	Increament
--	Decreament

Assignment Operators

- Assignment operators assign values to JavaScript variables.

Operator	Example	Same As
=	x = y	x = y
+=	x += y	x = x + y

Operator	Example	Same As
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y
**=	x **= y	x = x ** y

Comparison Operator

Operator	Description
==	equal to
===	equal value and equal type
!=	not equal
!==	not equal value or not equal type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
?	ternary operator

Logical Operators

Operator	Description
&&	logical and
|	logical or
!	logical not

Type Operators

- You can use the JavaScript `typeof` operator to find the type of a JavaScript variable.
- The `typeof` operator returns the type of a variable or an expression

Operator	Description
typeof	Returns the type of a variable
instanceof	Returns true if an object is an instance of an object type

Bitwise Operators (optional)

Operator	Description	Example	Same as	Result	Decimal
&	AND	5 & 1	0101 & 0001	0001	1
|	OR	5 | 1	0101 | 0001	0101	5
~	NOT	~ 5	~0101	1010	10
^	XOR	5 ^ 1	0101 ^ 0001	0100	4
<<	left shift	5 << 1	0101 << 1	1010	10

Operator	Description	Example	Same as	Result	Decimal
>>	right shift	5 >> 1	0101 >> 1	0010	2
>>>	unsigned right shift	5 >>> 1	0101 >>> 1	0010	2

Data Types

- String
- Number
- Bigint
- Boolean
- Undefined
- Null
- Symbol
- Object
    - An Object
    - An array
    - A date

String

- A string (or a text string) is a series of characters like "John Doe".
- Strings are written with quotes. You can use single or double quotes

```
<p id="demo"></p>
<script>
    let carName1 = "Volvo XC60";
    let carName2 = 'Volvo XC90';
    document.getElementById("demo").innerHTML =
    carName1 + "<br>" + carName2;
</script>
```

```
Volvo XC60
Volvo XC90
```

Numbers

- All JavaScript numbers are stored as decimal numbers (floating point).
- Numbers can be written with, or without decimals

```
<p id="demo"></p>
<script>
    let x1 = 34.00;
    let x2 = 34;
    let x3 = 3.14;
    document.getElementById("demo").innerHTML =
    x1 + "<br>" + x2 + "<br>" + x3;
</script>
```

```
34
34
3.14
```

Booleans

- Booleans can only have two values: true or false.

```
<p id="demo"></p>
<script>
    let x = 5;
    let y = 5;
    let z = 6;
    document.getElementById("demo").innerHTML =
    (x == y) + "<br>" + (x == z);
</script>
```

```
true
false
```

Arrays

- JavaScript arrays are written with square brackets.
- Array items are separated by commas.

```
<p id="demo"></p>
<script>
    const cars = ["Benz","Volvo","BMW"];
    document.getElementById("demo").innerHTML = cars[0];
</script>
```

```
Benz
```

Objects

- JavaScript objects are written with curly braces `{}`.
- Object properties are written as `name:value` pairs, separated by commas.

```
<p id="demo"></p>
<script>
    const person = {
        firstName : "John",
        lastName : "Doe",
        age : 50,
        eyeColor : "blue" };
    }
    document.getElementById("demo").innerHTML =
    person.firstName + " is " + person.age + " years old.";
</script>
```

```
John is 50 years old.
```

Functions

- A JavaScript function is a block of code designed to perform a particular task.
- A JavaScript function is executed when its called.

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

- When JavaScript reaches a `return` statement, the function will stop executing.
- If the function was invoked from a statement, JavaScript will "return" to execute the code after the invoking statement.
- Functions often compute a return value. The return value is "returned" back to the "caller"

```
<p id="demo"></p>
<script>
    function myFunction(p1, p2) {
        return p1 * p2;
    }
    let result = myFunction(4, 3);
    document.getElementById("demo").innerHTML = result;
</script>
```

```
12
```

Objects

- You can define (and create) a JavaScript object with an object literal
- The `name:values` pairs in JavaScript objects are called properties
- Objects can also have methods. Methods are actions that can be performed on objects. Methods are stored in properties as function definitions.
- A method is a function stored as a property

```
const person = {
    firstName: "John",
    lastName : "Doe",
    id : 5566,
    fullName : function(){
                return this.firstName + " " + this.lastName;
```

```
                  }
};
```

- In a function definition, `this` refers to the "owner" of the function.
- In JavaScript, the `this` keyword refers to an object
  - In an object method, `this` refers to the object.
  - Alone, `this` refers to the global object.
  - In a function, `this` refers to the global object.
  - In a function, in strict mode, `this` is undefined.
  - In an event, `this` refers to the element that received the event.

Array Methods

Array Length:

- The `length` property returns the length (size) of an array

```
<p id="demo"></p>
<script>
    const fruits = ["Banana", "Orange", "Apple", "Mango"];
    let size = fruits.length;
    document.getElementById("demo").innerHTML = size;
</script>
```

```
4
```

Array `toString()`

```
<p id="demo"></p>
<script>
    const fruits = ["Banana", "Orange", "Apple", "Mango"];
    document.getElementById("demo").innerHTML = fruits.toString();
</script>
```

```
Banana,Orange,Apple,Mango
```

Array `join()`

- The `join()` method also joins all array elements into a string.
- It behaves just like `toString(),` but in addition you can specify the separator

```
<p id="demo"></p>
<script>
    const fruits = ["Banana", "Orange", "Apple", "Mango"];
    document.getElementById("demo").innerHTML = fruits.join(" * ");
</script>
```

```
Banana * Orange * Apple * Mango
```

Array `pop()`

- The `pop()` method removes the last element from an array

```
<p id="demo1"></p>
<p id="demo2"></p>
<script>
    const fruits = ["Banana", "Orange", "Apple", "Mango"];
    document.getElementById("demo1").innerHTML = fruits;
    fruits.pop();
    document.getElementById("demo2").innerHTML = fruits;
</script>
```

```
Banana,Orange,Apple, Mango
Banana,Orange,Apple
```

Array `push()`

- The `push()` method adds a new element to an array (at the end)

```
<p id="demo1"></p>
<p id="demo2"></p>
<script>
    const fruits = ["Banana", "Orange", "Apple", "Mango"];
    document.getElementById("demo1").innerHTML = fruits;
    fruits.push("Kiwi");
    document.getElementById("demo2").innerHTML = fruits;
</script>
```

```
Banana,Orange,Apple,Mango
Banana,Orange,Apple,Mango,Kiwi
```

Array `shift()`

- The `shift()` method removes the first array element and "shifts" all other elements to a lower index.

```
<p id="demo1"></p>
<p id="demo2"></p>
<script>
    const fruits = ["Banana", "Orange", "Apple", "Mango"];
    document.getElementById("demo1").innerHTML = fruits;
    fruits.shift();
    document.getElementById("demo2").innerHTML = fruits;
</script>
```

```
Banana,Orange,Apple,Mango
Orange,Apple,Mango
```

Array `unshift()`

- The `unshift()` method adds a new element to an array (at the beginning), and "unshifts" older elements

```
<p id="demo1"></p>
<p id="demo2"></p>
<script>
    const fruits = ["Banana", "Orange", "Apple", "Mango"];
    document.getElementById("demo1").innerHTML = fruits;
    fruits.unshift("Lemon");
    document.getElementById("demo2").innerHTML = fruits;
</script>
```

```
Banana,Orange,Apple,Mango
Lemon,Banana,Orange,Apple,Mango
```

Array `delete()`

- Using `delete()` leaves undefined holes in the array.

```
<p id="demo1"></p>
<p id="demo2"></p>
<script>
    const fruits = ["Banana", "Orange", "Apple", "Mango"];
    document.getElementById("demo1").innerHTML = "The first fruit is: " + fruits[0];
    delete fruits[0];
    document.getElementById("demo2").innerHTML = "The first fruit is: " + fruits[0];
</script>
```

```
The first fruit is: Banana
The first fruit is: undefined
```

Conditional Statements

- Conditional statements are used to perform different actions based on different conditions

`if` Statement:

- Use the `if` statement to specify a block of JavaScript code to be executed if a condition is true.

```
<p id="demo">Good Evening!</p>
<script>
```

```
    if (new Date().getHours() < 18) {
        document.getElementById("demo").innerHTML = "Good day!";
    }
</script>
```

```
Good Day!
```

`else` Statement:

- Use the `else` statement to specify a block of code to be executed if the condition is false.

```
<p id="demo"></p>
<script>
    const hour = new Date().getHours();
    let greeting;
    if (hour < 18) {
        greeting = "Good day";
    } else {
        greeting = "Good evening";
    }
    document.getElementById("demo").innerHTML = greeting;
</script>
```

```
Good day
```

`else if` Statement:

- Use the `else if` statement to specify a new condition if the first condition is false.

```
<p id="demo"></p>
<script>
    const time = new Date().getHours();
    let greeting;
    if (time < 10) {
        greeting = "Good morning";
    } else if (time < 20) {
        greeting = "Good day";
    } else {
        greeting = "Good evening";
    }
    document.getElementById("demo").innerHTML = greeting;
</script>
```

```
Good day
```

Switch Statement

- The `switch` statement is used to perform different actions based on different conditions.
- The switch expression is evaluated once.

- The value of the expression is compared with the values of each case.
- If there is a match, the associated block of code is executed.
- If there is no match, the default code block is executed.

```
//Syntax
switch(expression) {
    case x:
        // code block
        break;
    case y:
        // code block
        break;
    default:
        // code block
}
```

```
<p id="demo"></p>
<script>
    let day;
    switch (new Date().getDay()) {
        case 0:
            day = "Sunday";
            break;
        case 1:
            day = "Monday";
            break;
        case 2:
            day = "Tuesday";
            break;
        case 3:
            day = "Wednesday";
            break;
        case 4:
            day = "Thursday";
            break;
        case 5:
            day = "Friday";
            break;
        case 6:
            day = "Saturday";
    }
    document.getElementById("demo").innerHTML = "Today is " + day;
</script>
```

```
Today is Friday
```

Loops

for Loop

- The `for` statement creates a loop with 3 optional expressions

- `Expression 1` is executed (one time) before the execution of the code block.
- `Expression 2` defines the condition for executing the code block.
- `Expression 3` is executed (every time) after the code block has been executed.

```
//Syntax
for (expression 1; expression 2; expression 3){
    // code block to be executed
}
```

```
<p id="demo"></p>
<script>
    let text = "";
    for (let i = 0; i < 5; i++){
        text += "The number is " + i + "<br>";
    }
    document.getElementById("demo").innerHTML = text;
</script>
```

```
The number is 0
The number is 1
The number is 2
The number is 3
The number is 4
```

for in Loop

- The JavaScript `for in` statement loops through the properties of an Object

```
//Syntax
for (key in object) {
    // code block to be executed
}
```

```
<p id="demo"></p>
<script>
    const person = {fname:"John", lname:"Doe", age:25};
    let txt = "";
    for (let x in person) {
        txt += person[x] + " ";
    }
    document.getElementById("demo").innerHTML = txt;
</script>
```

```
John Doe 25
```

for of Loop

- The JavaScript `for of` statement loops through the values of an iterable object.

```
//Syntax
for (variable of iterable) {
    // code block to be executed
}
```

```
<p id="demo"></p>
<script>
    const cars = ["BMW", "Volvo", "Mini"];
    let text = "";
    for (let x of cars) {
        text += x + "<br>";}
    document.getElementById("demo").innerHTML = text;
</script>
```

```
BMW
Volvo
Mini
```

while Loop

- The `while` loop loops through a block of code as long as a specified condition is true

```
//Syntax
while (condition) {
    // code block to be executed
}
```

```
<p id="demo"></p>
<script>
    let text = "";
    let i = 0;
    while (i < 10) {
        text += "<br>The number is " + i;
        i++;
    }
    document.getElementById("demo").innerHTML = text;
</script>
```

```
The number is 0
The number is 1
The number is 2
The number is 3
The number is 4
The number is 5
The number is 6
The number is 7
The number is 8
The number is 9
```

do while Loop

- The `do while` loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

```
//Syntax
do {
    // code block to be executed
} while (condition);
```

```
<p id="demo"></p>
<script>
    let text = ""
    let i = 0;
    do {
      text += "<br>The number is " + i;
      i++;
    }
    while (i < 10);
    document.getElementById("demo").innerHTML = text;
</script>
```

```
The number is 0
The number is 1
The number is 2
The number is 3
The number is 4
The number is 5
The number is 6
The number is 7
The number is 8
The number is 9
```