JavaScript Part-1

Unit-4

Web Designing KIT401

Topic Covered

- Introduction to Client Side Scripting , Introduction to Java Script , Javascript Types , Variables in JS, Operators in JS , Conditions Statements , Java Script Loops, JS Popup Boxes

JavaScript Introduction

- **JavaScript** often abbreviated as **JS**, is a programming language that is one of the core technologies of the World Wide Web, alongside HTML and CSS.
- 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.

Features of JavaScript

There are following features of JavaScript:

- All popular web browsers support JavaScript as they provide built-in execution environments.
- JavaScript follows the syntax and structure of the C programming language. Thus, it is a structured programming language.
- JavaScript is a weakly typed language, where certain types are implicitly cast (depending on the operation).
- JavaScript is an object-oriented programming language that uses prototypes rather than using classes for inheritance.
- It is a light-weighted and interpreted language.
- It is a case-sensitive language.
- JavaScript is supportable in several operating systems including, Windows, macOS, etc.
- It provides good control to the users over the web browsers.

Advantages of JavaScript

The merits of using JavaScript are:

- 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

JavaScript Types

1 - Server-side Scripting

2 - Client-side Scripting

1. Server-side Scripting

Server-side scripting is a programming technique for creating code that may run software on the server side. In other words, server-side scripting is any scripting method that may operate on a web server. At the server end, actions such as website customization, dynamic changes in website content, response creation to user requests, database access, and many more are carried out.
Server-side scripting creates a communication channel between a server and a client. Previously, ***CGI (Common Gateway Interface)*** server-side scripting programming languages, including ***PHP, ColdFusion, Python, ASP.net, Java, C++, Ruby, C#,***

2. Client-side scripting

Client-side scripting generates code that may be executed on the client end without needing server-side processing. These scripts are typically embedded into HTML text. Client-side scripting may be utilized to check the user's form for problems before submitting it and to change the content based on the user input. The web needs three components to function: client, database, and server.
The client-side scripting may significantly reduce server demand. It is intended to be utilized as a scripting language with a web browser as the host program.
The ***HTML*** and ***CSS*** are delivered as plain text when a user uses a browser to request a webpage from the server, and the browser understands and renders the web content at the client end.

Difference between Server-side Scripting and Client-side Scripting

Features	Server-side Scripting	Client-side Scripting
Primary Function	The main function of this scripting is to manipulate and grant access to the requested database.	The main purpose of this scripting is to give the requested output to the end-user.
Uses	It is employed at the backend, where the source code is invisible or concealed on the client side.	It is utilized at the front end, which users may view through the browser.
Processing	It needs server interaction.	It doesn't need any server interaction.
Security	It is more secure while working on a web app.	It is less secure than server-side scripting due to the code accessibility offered to the client.
Running	It executes on the web server.	It executes on the remote computer system.
Dependability	It doesn't depend on the client.	It depends on the user's browser version.
File Access	It offers complete access to the file that is stored in the web database server.	It doesn't offer any access to the files on the web servers.
Code Allowance	It enables the backend developer to hide the source code from the user.	The user is given access to the written code after confirming their requirements.
Occurrence	It only responds after the user begins the browsing request.	It happens when the browser processes all of the codes and then acts according to the client's needs.
Affect	It may reduce the server load.	It may effectively customize web pages and offer dynamic websites.
Languages Involved	The server-side scripting programming languages, such as PHP, ColdFusion, Python, ASP.net, Java, C++, Ruby, C#, etc.	Its programming languages are HTML, CSS, and JavaScript.

JavaScript SYNTAX

- JavaScript can be implemented using JavaScript statements that are placed within the <script>…….</script>

Syntax

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

First Program

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

Include JavaScript code

There is a flexibility given to include JavaScript code anywhere in an HTML document. However the most preferred ways to include JavaScript in an HTML file 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

Script in <head>...</head> section.

```
<html>
<head>
<script type="text/javascript">
alert("Hello World")
</script>
</head>

</html>
```

Script 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>
```

Script in an external file and then include in <head>...</head> section

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

filename.js

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

To use JavaScript from an external file source, you need to write all your JavaScript source code in a simple text file with the extension ".js" and then include that file as shown above.

For example, you can keep the following content in filename.js file and then you can use sayHello function in your HTML file after including the filename.js file.

JavaScript Comments

JavaScript comments can be used to explain JavaScript code, and to make it more readable.
JavaScript comments can also be used to prevent execution, when testing alternative code.

Single Line Comments

Single line comments start with //.
Any text between // and the end of the line will be ignored by JavaScript (will not be executed).
This example uses a single-line comment before each code line:

Multi-line Comments

Multi-line comments start with /* and end with */.

Any text between /* and */ will be ignored by JavaScript.

Single Line Comments

```
<script>
let x = 5;      // Declare x, give it the value of 5
let y = x + 2;  // Declare y, give it the value of x + 2
</script>
```

Multi-line Comments

```
/*
The code below will change
the heading with id = "myH"
and the paragraph with id = "myP"
in my web page:
*/
```

JavaScript Variables

A **JavaScript variable** is simply a name of storage location. There are two types of variables in JavaScript : local variable and global variable.
There are some rules while declaring a JavaScript variable (also known as identifiers).

- Name must start with a letter (a to z or A to Z), underscore(_), or dollar($) sign.
- After first letter we can use digits (0 to 9), for example value1.
- JavaScript variables are case sensitive, for example x and X are different variables.

Variables are Containers for Storing Data
JavaScript Variables can be declared in 4 ways:
•Automatically
•Using var
•Using let
•Using const

```
<script>
var x = 10;
var y = 20;
var z=x+y;
document.write(z);
</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.

- Local Variables: A local variable will be visible only within a function where it is defined. Function parameters are always local to that function.

- Global Variables: A global variable has global scope which means it can be defined anywhere in your JavaScript code.

Local Variables

```
<script>
function abc(){
var x=10;//local variable
}
</script>
```

Global Variables

```
<script>
var value=50;//global variable
function a(){
alert(value);
}
function b(){
alert(value);
}
</script>
```

JavaScript Operators

JavaScript operators are symbols that are used to perform operations on operands. For example:

var sum=10+20;
Here, + is the arithmetic operator and = is the assignment operator.

There are following types of operators in JavaScript.

- Arithmetic Operators
- Comparison (Relational) Operators
- Bitwise Operators
- Logical Operators
- Assignment Operators
- Special Operators

JavaScript Arithmetic Operators

Arithmetic operators are used to perform arithmetic operations on the operands. The following operators are known as JavaScript arithmetic operators.

Operator	Description	Example
+	Addition	10+20 = 30
-	Subtraction	20-10 = 10
*	Multiplication	10*20 = 200
/	Division	20/10 = 2
%	Modulus (Remainder)	20%10 = 0
++	Increment	var a=10; a++; Now a = 11
--	Decrement	var a=10; a--; Now a = 9

JavaScript Comparison Operators

The JavaScript comparison operator compares the two operands. The comparison operators are as follows:

Operator	Description	Example
==	Is equal to	10==20 = false
===	Identical (equal and of same type)	10==20 = false
!=	Not equal to	10!=20 = true
!==	Not Identical	20!==20 = false
>	Greater than	20>10 = true
>=	Greater than or equal to	20>=10 = true
<	Less than	20<10 = false
<=	Less than or equal to	20<=10 = false

JavaScript Bitwise Operators

The bitwise operators perform bitwise operations on operands. The bitwise operators are as follows:

Operator	Description	Example
&	Bitwise AND	(10==20 & 20==33) = false
|	Bitwise OR	(10==20 | 20==33) = false
^	Bitwise XOR	(10==20 ^ 20==33) = false
~	Bitwise NOT	(~10) = -10
<<	Bitwise Left Shift	(10<<2) = 40
>>	Bitwise Right Shift	(10>>2) = 2
>>>	Bitwise Right Shift with Zero	(10>>>2) = 2

JavaScript Logical Operators

The following operators are known as JavaScript logical operators.

Operator	Description	Example
&&	Logical AND	(10==20 && 20==33) = false
||	Logical OR	(10==20 || 20==33) = false
!	Logical Not	!(10==20) = true

JavaScript Assignment Operators

The following operators are known as JavaScript assignment operators.

Operator	Description	Example
=	Assign	10+10 = 20
+=	Add and assign	var a=10; a+=20; Now a = 30
-=	Subtract and assign	var a=20; a-=10; Now a = 10
*=	Multiply and assign	var a=10; a*=20; Now a = 200
/=	Divide and assign	var a=10; a/=2; Now a = 5
%=	Modulus and assign	var a=10; a%=2; Now a = 0

JavaScript Special Operators

The following operators are known as JavaScript special operators.

Operator	Description
(?:)	Conditional Operator returns value based on the condition. It is like if-else.
,	Comma Operator allows multiple expressions to be evaluated as single statement.
delete	Delete Operator deletes a property from the object.
in	In Operator checks if object has the given property
instanceof	checks if the object is an instance of given type
new	creates an instance (object)
typeof	checks the type of object.
void	it discards the expression's return value.
yield	checks what is returned in a generator by the generator's iterator.

Conditional Statements

In JavaScript we have the following conditional statements:

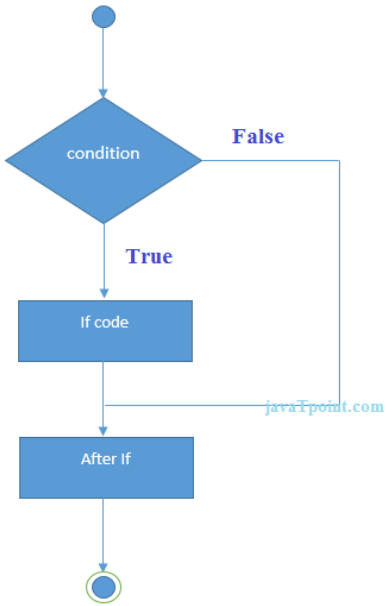
- Use if to specify a block of code to be executed, if a specified condition is true
- Use else to specify a block of code to be executed, if the same condition is false
- Use else if to specify a new condition to test, if the first condition is false
- Use switch to specify many alternative blocks of code to be executed

The if Statement

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

Syntax

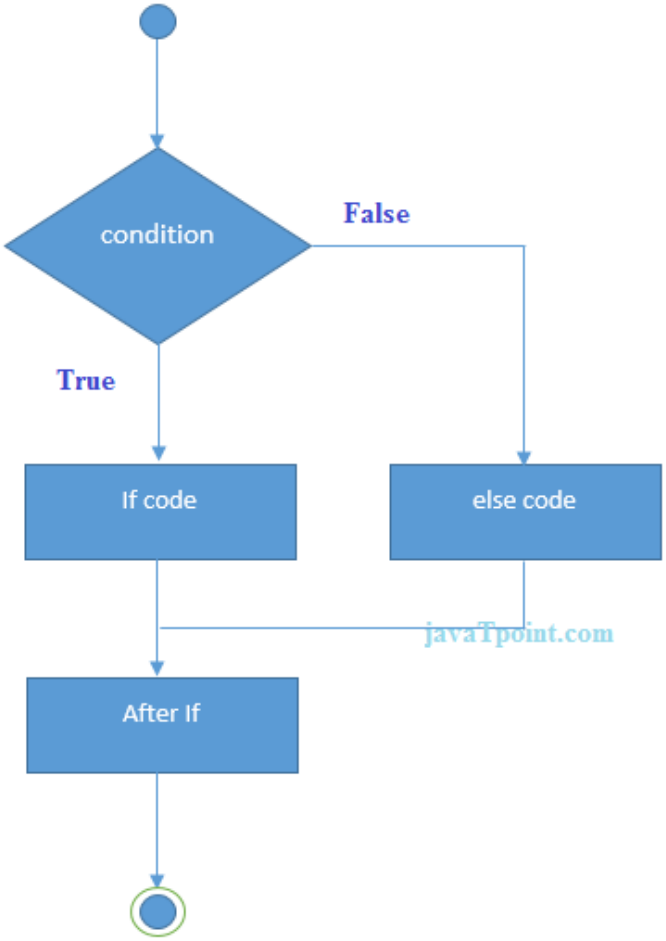
```
if (condition) {
  //  block of code to be executed if the condition is true
}
```



The else Statement

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

```
if (condition) {
  // block of code to be executed if the condition is true
} else {
  // block of code to be executed if the condition is false
}
```



The else if Statement

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

Syntax

```
if (condition1) {
  // block of code to be executed if condition1 is true
} else if (condition2) {
  // block of code to be executed if the condition1 is false and condition2 is true
} else {
  // block of code to be executed if the condition1 is false and condition2 is false
}
```

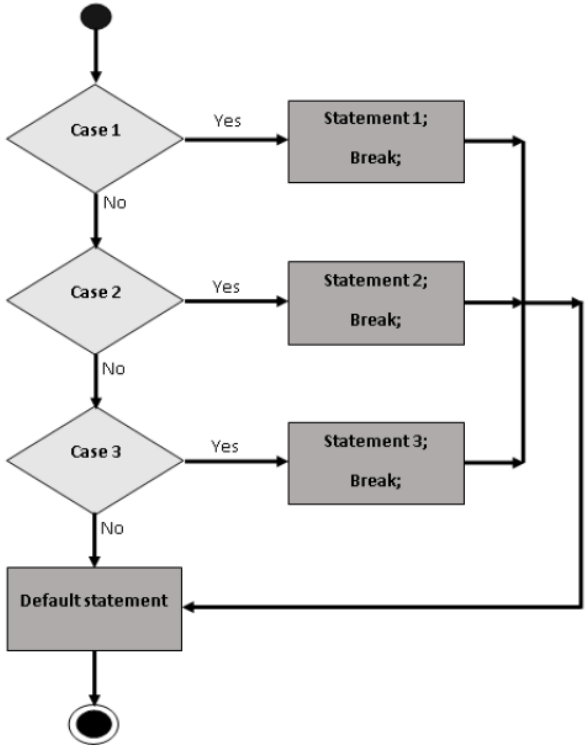
```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript if .. else</h2>

<p>A time-based greeting:</p>

<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>

</body>
</html>
```

Switch

The **JavaScript switch statement** is used *to execute one code from multiple expressions*. It is just like else if statement that we have learned in previous page. But it is convenient than *if..else..if* because it can be used with numbers, characters etc.

The signature of JavaScript switch statement is given below.

```
switch(expression){
case value1:
 code to be executed;
 break;
case value2:
 code to be executed;
 break;
......

default:
 code to be executed if above values are not matched;
}
```



Switch Example

```
<!DOCTYPE html>
<html>
<body>
<script>
var grade='B';
var result;
switch(grade){
case 'A':
result="A Grade";
break;
case 'B':
result="B Grade";
break;
case 'C':
result="C Grade";
break;
default:
result="No Grade";
}
document.write(result);
</script>
</body>
</html>
```

JavaScript Loops

The **JavaScript loops** are used *to iterate the piece of code* using for, while, do while or for-in loops. It makes the code compact. It is mostly used in array.
There are four types of loops in JavaScript.

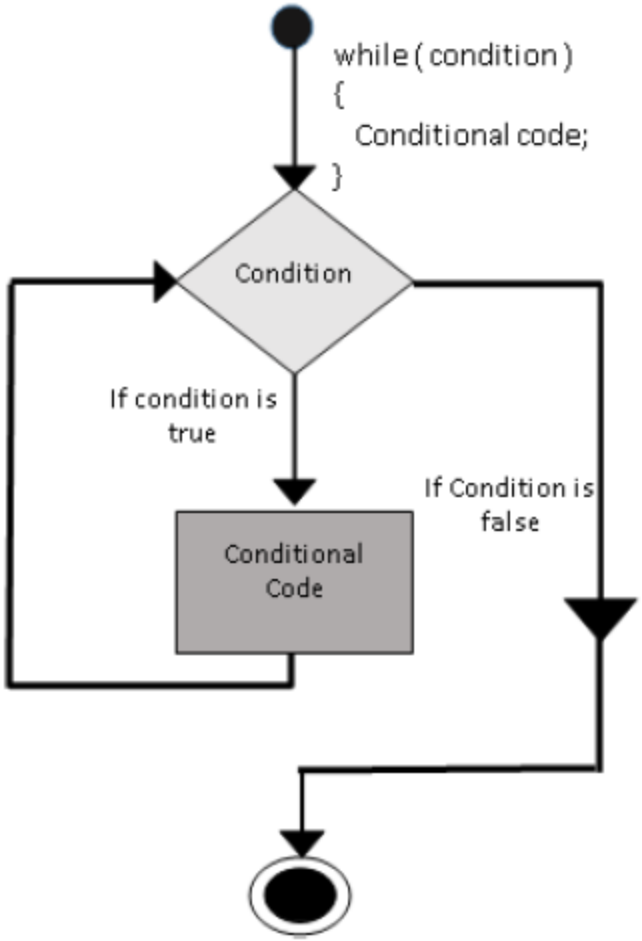
- while loop
- do-while loop
- for loop
- for-in loop
- for-of loop

JavaScript while loop

The JavaScript while loop iterates the elements for the infinite number of times. It should be used if number of iteration is not known. The syntax of while loop is given below.

```
while (condition)
{
   code to be executed
}
```

```
<script>
var i=11;
while (i<=15)
{
document.write(i + "<br/>");
i++;
}
</script>
```

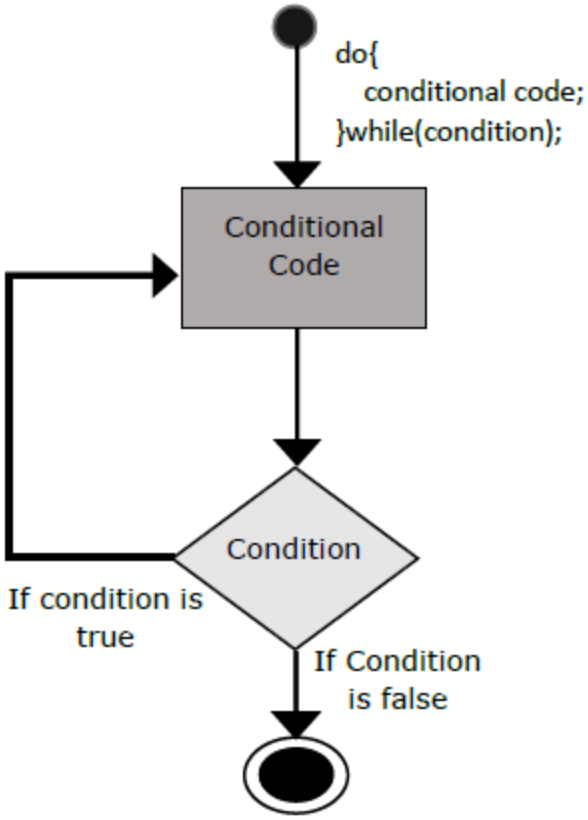


JavaScript do while loop

The **JavaScript do while loop** *iterates the elements for the infinite number of times* like while loop. But, code is *executed at least* once whether condition is true or false. The syntax of do while loop is given below.

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

```
<script>
var i=21;
do{
document.write(i + "<br/>");
i++;
}while (i<=25);
</script>
```

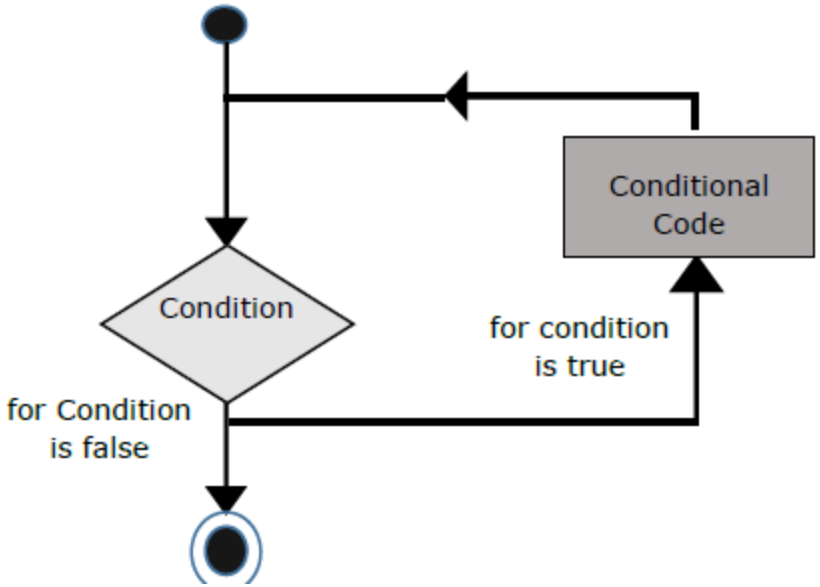


JavaScript For loop

The **JavaScript for loop** *iterates the elements for the fixed number of times*. It should be used if number of iteration is known. The syntax of for loop is given below.

```
for (initialization; condition; increment)
{
   code to be executed
}
```

```
<!DOCTYPE html>
<html>
<body>
<script>
for (i=1; i<=5; i++)
{
document.write(i + "<br/>")
}
</script>
</body>
</html>
```



for-in loop

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

Syntax

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

JavaScript For In Loop

The for in statement loops through the properties of an object**:**

OUTPUT

John
Doe
25

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript For In Loop</h2>
<p>The for in statement loops through the properties of an object:</p>

<p id="demo"></p>

<script>
const person = {fname:"John", lname:"Doe", age:25};

let txt = "";
for (let x in person) {
document.write("<br>")
  document.write(person[x])
  }


</script>

</body>
</html>
```

The for of Loop

The JavaScript for of statement loops through the values of an iterable object.
It lets you loop over iterable data structures such as Arrays, Strings, Maps, NodeLists, and more:

Syntax

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

JavaScript Arrays
For In Loops
The for in statement can loops over array values:

OUTPUT

45
4
9
16
25

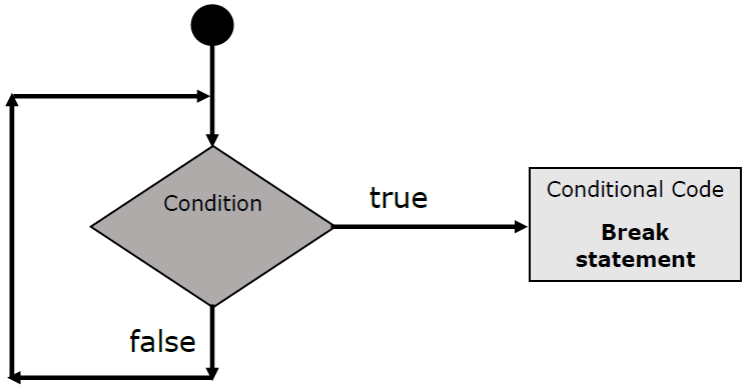
```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Arrays</h1>
<h2>For In Loops</h2>
<p>The for in statement can loops over array values:</p>

<p id="demo"></p>

<script>
const numbers = [45, 4, 9, 16, 25];

let txt = "";
for (let x of numbers) {
  document.write("<br>")
  document.write(x);
}



</script>

</body>
</html>
```

JavaScript Break and Continue

Break Statement

The break statement "jumps out" of a loop.

The break statement, which was briefly introduced with the switch statement, is used to exit a loop early, breaking out of the enclosing curly braces.
Flow Chart
The flow chart of a break statement would look as follows:




```
<!DOCTYPE html>
<html>
<body>
<p>break statement:</p>
<script>
const numbers = [45, 4, 9, 16, 25];
for (let x of numbers) {
  if (x === 16) { break; }
document.write("<br>")
  document.write(x);
}
</script>
</body>
</html>
```

OUTPUT

break statement:

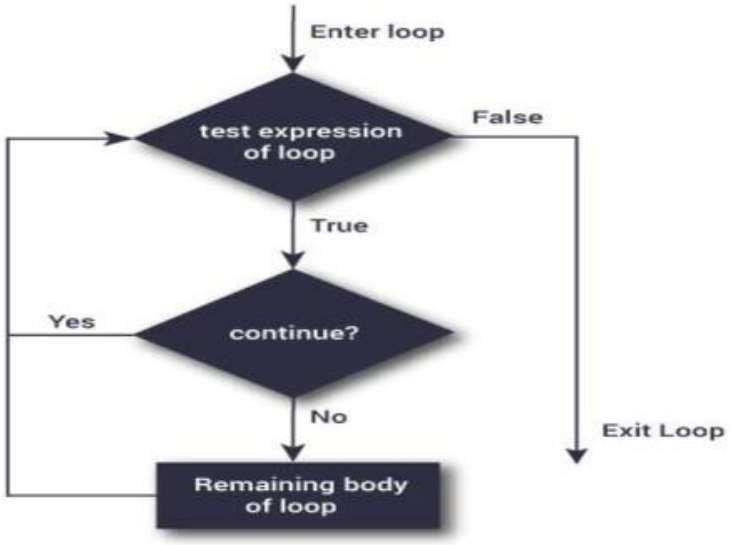
45
4
9

Continue Statement

The continue statement "jumps over" one iteration in the loop.

The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

This example skips the value of 16




```
<!DOCTYPE html>
<html>
<body>
<p>continue statement:</p>
<script>
const numbers = [45, 4, 9, 16, 25];
for (let x of numbers) {
  if (x === 16) { continue; }
  document.write("<br>")
  document.write(x);
}
</script>
</body>
</html>
```

OUTPUT

continue statement:

45
4
9
25

JavaScript Popup Boxes

In Javascript, popup boxes are used to display the message or notification to the user. There are three types of pop-up boxes in JavaScript namely Alert Box, Confirm Box and Prompt Box.

Alert Box: It is used when a warning message is needed to be produced. When the alert box is displayed to the user, the user needs to press ok and proceed.

Syntax
window.alert("sometext");

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Alert</h2>

<button onclick="myFunction()">Try
it</button>

<script>
function myFunction() {
  alert("I am an alert box!");
}
</script>

</body>
</html>
```

Confirm Box

A confirm box is often used if you want the user to verify or accept something.
When a confirm box pops up, the user will have to click either "OK" or "Cancel" to proceed.
If the user clicks "OK", the box returns **true**. If the user clicks "Cancel", the box returns **false**.

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Confirm Box</h2>


<button onclick="myFunction()">Try it</button>

<p id="demo"></p>

<script>
function myFunction() {
  var txt;
  if (confirm("Press a button!")) {
    txt = "You pressed OK!";
  } else {
    txt = "You pressed Cancel!";
  }
  document.getElementById("demo").innerHTML = txt;
}
</script>

</body>
</html>
```

Prompt Box

A prompt box is often used if you want the user to input a value before entering a page.

When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed after entering an input value.

If the user clicks "OK" the box returns the input value. If the user clicks "Cancel" the box returns null.

Syntax
window.prompt("sometext","defaultText");

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Prompt</h2>

<button onclick="myFunction()">Try it</button>

<p id="demo"></p>

<script>
function myFunction() {
  let text;
  let person = prompt("Please enter your name:", "Harry
Potter");
  if (person == null || person == "") {
    text = "User cancelled the prompt.";
  } else {
    text = "Hello " + person + "! How are you today?";
  }
  document.getElementById("demo").innerHTML = text;
}
</script>

</body>
</html>
```