Control Statements in JavaScript

Control statements are used to dictate the flow of execution in a program.
JavaScript offers several control statements, such as:

Types:

- **Conditional Statements:**
  - if statement
  - if-else statement
  - else if ladder
  - switch statement
- **Looping Statements:**
  - for loop
  - while loop
  - do...while loop
- **Jump Statements:**
  - break statement
  - continue statement

JavaScript if-statement

It is a conditional statement used to decide whether a certain statement or block of statements will be executed or not i.e. if a certain condition is true then a block of statements is executed otherwise not.

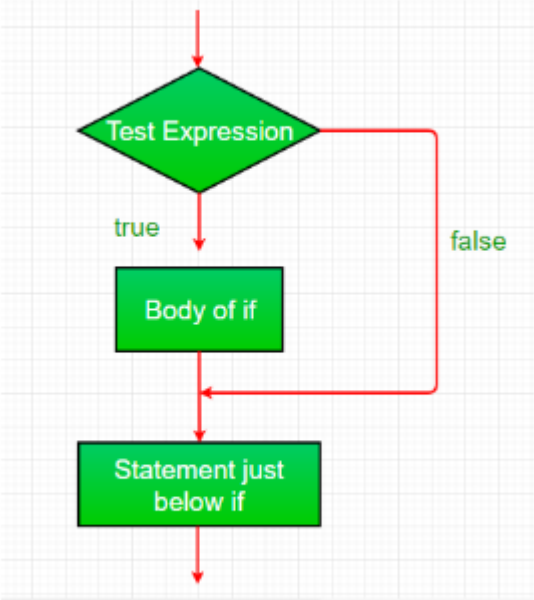
Syntax:

```
if(condition)
{
   // Statements to execute if
   // condition is true
}
```

The if statement accepts Boolean values – if the value is true then it will execute the block of statements under it. If we do not provide the curly braces '{' and '}' after **if(condition)** then by default if statement considers the immediate one statement to be inside its block. For example,

```
if(condition)
    statement1;
    statement2;
// Here if the condition is true, if block
// will consider only statement1 to be inside
// its block.
```

Flow chart:



if-condition statement

Example: Here is a simple example demonstrating the **if** statement.

```
1  // JavaScript program to illustrate If statement
2  let age = 19;
3
4  if (age > 18)
5  console.log("Congratulations, You are eligible to drive");
6
```

Output:

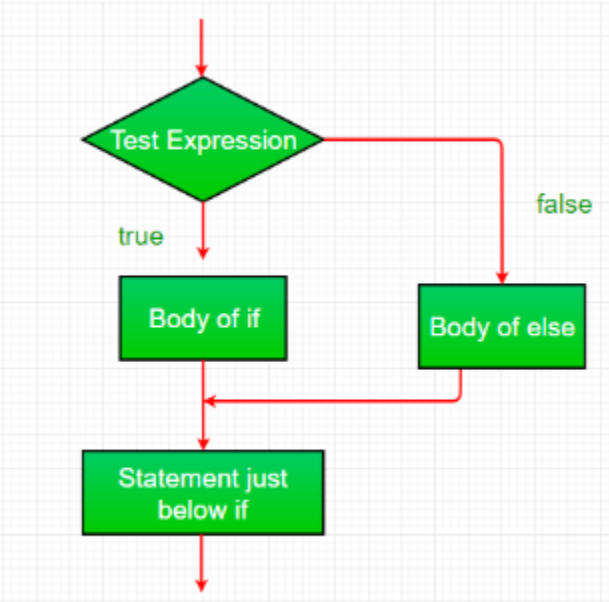
```
Congratulations, You are eligible to drive
```

JavaScript if-else statement

The if statement alone tells us that if a condition is true it will execute a block of statements and if the condition is false it won't. But what if we want to do something else if the condition is false? Here comes the else statement. We can use the else statement with the if statement to execute a block of code when the condition is false.

Syntax:

```
if (condition)
{
    // Executes this block if
    // condition is true
}
else
{
    // Executes this block if
    // condition is false
}
```



if-else statement

Flow chart

***Example*:** This example describes the if-else statement in Javascript.

```
// JavaScript program to illustrate If-else statement
let i = 10;

if (i < 15)
    console.log("i is less than 15");
else
    console.log("I is greater than 15");

```

Output:

```
i is less than 15
```

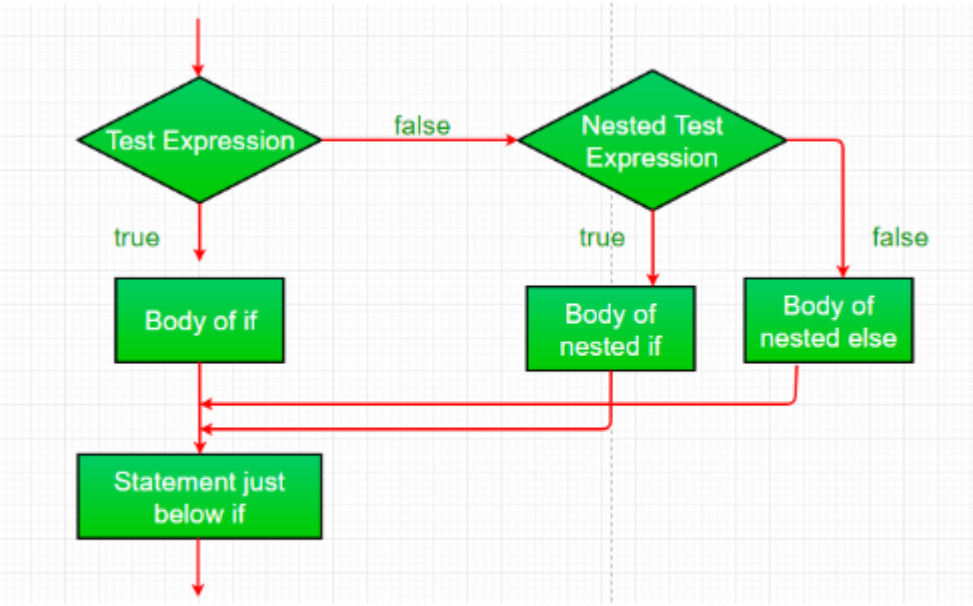
JavaScript nested-if statement

JavaScript allows us to nest if statements within if statements. i.e, we can place an if statement inside another if statement. A nested if is an if statement that is the target of another if or else.

Syntax:

```
if (condition1)
{
   // Executes when condition1 is true
   if (condition2)
   {
      // Executes when condition2 is true
   }
}
```

Flow chart:



nested-if statement

Example: This example describes the nested-if statement in Javascript.

```
// JavaScript program to illustrate nested-if statement
let i = 10;

if (i == 10) {  // First if statement
    if (i < 15) {
        console.log("i is smaller than 15");
        // Nested - if statement
        // Will only be executed if statement above
        // it is true
        if (i < 12)
            console.log("i is smaller than 12 too");
        else
            console.log("i is greater than 15");
    }
}

```

```
i is smaller than 15
i is smaller than 12 too
```

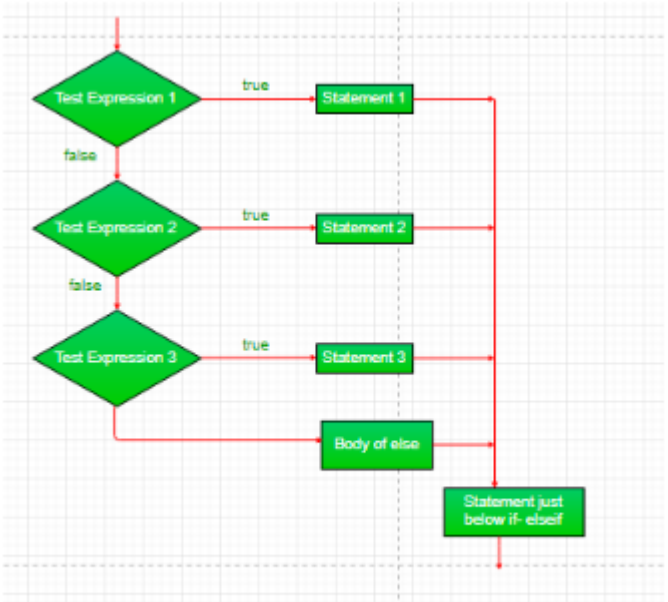
JavaScript if-else-if ladder statement

Here, a user can decide among multiple options. The if statements are executed from the top down. As soon as one of the conditions controlling the if is true, the statement associated with that if is executed, and the rest of the ladder is bypassed. If none of the conditions is true, then the final else statement will be executed.

Syntax:

```
if (condition)
    statement;
else if (condition)
    statement;
.
.
else
    statement;
```

Flow chart:



if-else-if ladder statement

***Example*:** This example describes the if-else-if ladder statement in Javascript.

```
// JavaScript program to illustrate nested-if statement
let i = 20;

if (i == 10)
    console.log("i is 10");
else if (i == 15)
    console.log("i is 15");
else if (i == 20)
    console.log("i is 20");
else
    console.log("i is not present");

```

Output:

```
i is 20
```

JavaScript switch Statement

The JavaScript switch statement evaluates an expression and executes a block of code based on matching cases. It provides an alternative to long if-else chains, improving readability and maintainability, especially when handling multiple conditional branches.

Syntax:

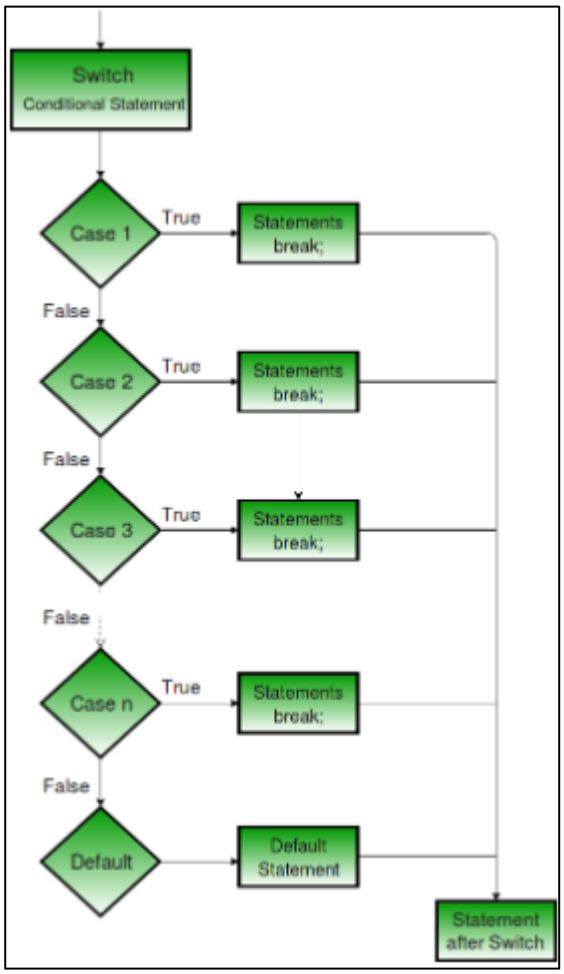
```
switch (expression) {
    case value1:
        // code block 1;
        break;
    case value2:
         // code block 2;
        break;
   ...
    default:
       // default code block;
}
```

- Expression is the value that you want to compare.
- Case value1, case value2, etc., represent the possible values of the expression.
- break statement terminates the switch statement. Without it, execution will continue into the next case.
- Default specifies the code to run if none of the cases match the expression.

How Switch Statement Works

- **Evaluation**: The expression inside the switch the statement is evaluated once.
- **Comparison**: The value of the expression is compared with each case label (using strict equality ===).
- **Execution**: If a match is found, the corresponding code block following the matching case the label is executed. If no match is found, the execution jumps to the default case (if present) or continues with the next statement after the switch block.
- **Break Statement**: After executing a code block, the break statement terminates the switch statement, preventing execution from falling through to subsequent cases. If break is omitted, execution will continue to the next case (known as "fall-through").
- **Default Case**: The default case is optional. If no match is found, the code block under default is executed.

Flowchart of Switch Statement



Switch Statement Example:

Here, we will print the day name on day 3.

```
let day = 3;
let dayName;

switch (day) {
    case 1:
        dayName = "Monday";
        break;
    case 2:
        dayName = "Tuesday";
        break;
    case 3:
        dayName = "Wednesday";
        break;
    case 4:
        dayName = "Thursday";
        break;
    case 5:
        dayName = "Friday";
        break;
    case 6:
        dayName = "Saturday";
        break;
    case 7:
        dayName = "Sunday";
        break;
    default:
        dayName = "Invalid day";
}

console.log(dayName);
```

Output:

```
Wednesday
```

Explanation:

- Day is set to 3.
- The switch statement evaluates day.
- Since day is 3, the case 3 the block is executed, assigning "Wednesday" to dayName.
- The break statement ends the switch statement, preventing execution from continuing into other cases.

JavaScript Loops

JavaScript loops are essential for efficiently handling repetitive tasks. They execute a block of code repeatedly as long as a specified condition remains true. These loops are powerful tools for automating tasks and streamlining your code

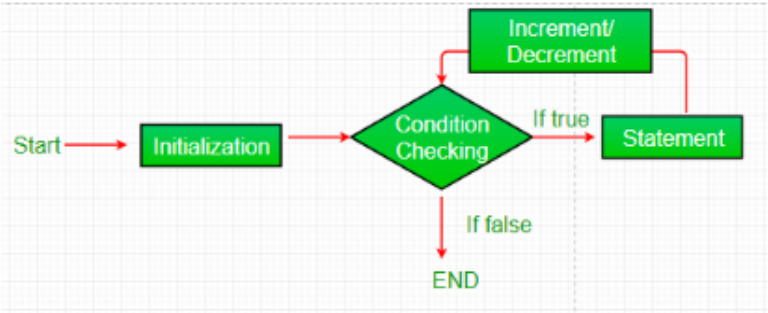
1. JavaScript for Loop

The JS for loop provides a concise way of writing the loop structure. The for loop contains initialization, condition, and increment/decrement in one line thereby providing a shorter, easy-to-debug structure of looping.

Syntax

```
for (initialization; testing condition; increment/decrement) {
    statement(s)
}
```

Flowchart



- **Initialization condition:** It initializes the variable and mark the start of a for loop. An already declared variable can be used or a variable can be declared, local to loop only.
- **Test Condition:** It is used for testing the exit condition of a for loop. It must return a boolean value. It is also an Entry Control Loop as the condition is checked prior to the execution of the loop statements.
- **Statement execution:** Once the condition is evaluated to be true, the statements in the loop body are executed.
- **Increment/ Decrement:** It is used for updating the variable for the next iteration.
- **Loop termination:** When the condition becomes false, the loop terminates marking the end of its life cycle.

Example

```
// JavaScript program to illustrate for loop
let x;

// for loop begins when x = 2
// and runs till x <= 4
for (x = 2; x <= 4; x++) {
    console.log("Value of x: " + x);
}

```

Output:

```
Value of x: 2
Value of x: 3
Value of x: 4
```

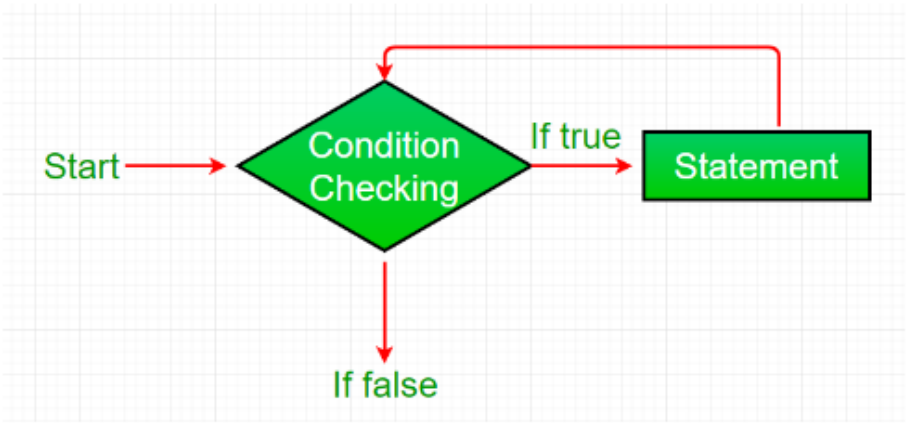
2. JavaScript while Loop

The JS while loop is a control flow statement that allows code to be executed repeatedly based on a given Boolean condition. The while loop can be thought of as a repeating if statement.

Syntax

```
while (boolean condition) {
    loop statements...
}
```

Flowchart



- While loop starts with checking the condition. If it is evaluated to be true, then the loop body statements are executed otherwise first statement following the loop is executed. For this reason, it is also called the Entry control loop
- Once the condition is evaluated to be true, the statements in the loop body are executed. Normally the statements contain an updated value for the variable being processed for the next iteration.
- When the condition becomes false, the loop terminates which marks the end of its life cycle.

Example:

```
// JavaScript code to use while loop
let val = 1;

while (val < 6) {
    console.log(val);
    val += 1;
}

```

```
1
2
3
4
5
```

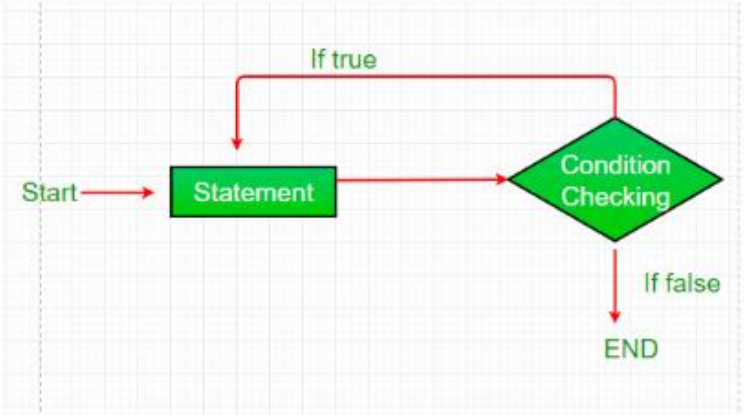
3. JavaScript do-while Loop

The JS do-while loop is similar to the while loop with the only difference is that it checks for the condition after executing the statements, and therefore is an example of an Exit Control Loop. It executes loop content at least once event the condition is false.

Syntax

```
do {
    Statements...
}
while (condition);
```

Flow chart:



- The do-while loop starts with the execution of the statement(s). There is no checking of any condition for the first time.
- After the execution of the statements and update of the variable value, the condition is checked for a true or false value. If it is evaluated to be true, the next iteration of the loop starts.
- When the condition becomes false, the loop terminates which marks the end of its life cycle.
- It is important to note that the do-while loop will execute its statements at least once before any condition is checked and therefore is an example of the exit control loop.

Example:

```
let test = 1;

do
{
    console.log(test);
    test++;
} while(test <= 5)

```

Output:

```
1
2
3
4
5
```

4. JavaScript for-in Loop

JS for-in loop is used to iterate over the properties of an object. The for-in loop iterates only over those keys of an object which have their enumerable property set to "true".

Syntax

```
for(let variable_name in object_name) {
    // Statement
}
```

***Example*:** This example shows the use of for-in loop.

```
let myObj = { x: 1, y: 2, z: 3 };
for (let key in myObj) {
    console.log(key, myObj[key]);
}

```

Output:

```
x 1
y 2
z 3
```

5. JavaScript for-of Loop

JS for-of loop is used to iterate the iterable objects for example – array, object, set and map. It directly iterate the value of the given iterable object and has more concise syntax than for loop.

Syntax:

```
for(let variable_name of  object_name) {
    // Statement
}
```

Example: This example shows the use of for-of loop.

```
let arr = [1, 2, 3, 4, 5];
for (let value of arr) {
    console.log(value);
}

```

Output:

```
1
2
3
4
5
```

6. JavaScript Break Statement

JS break statement is used to terminate the execution of the loop or switch statement when the condition is true.

Syntax:

```
break;
```

Example:

```
for (let i = 1; i < 6; i++)
{
    if (i == 4)
        break;

    console.log(i);
}
```

Output:

```
1
2
3
```

7. JavaScript Continue Statement

JS continue statement is used to break the iteration of the loop and follow with the next iteration. The break in iteration is possible only when the specified condition going to occur. The major difference between the continue and break statement is that the break statement breaks out of the loop completely while continue is used to break one statement and iterate to the next statement.

Syntax

```
continue;
```

Example:

```
for (let i = 0; i < 11; i++)
{
    if (i % 2 == 0)
        continue;

    console.log(i);
}

```

Output:

```
1
3
5
7
9
```

Functions in JavaScript

A function in JavaScript is a reusable block of code that performs a specific task. You define it once, and then you can run (or "call") it whenever you need that task done in your program.

A JavaScript function runs when it is "called" by some part of your code.

Syntax: The basic syntax to create a function in JavaScript is shown below.

```
function functionName(Parameter1, Parameter2, ...)
{
    // Function body
}
```

To create a function in JavaScript, we have to first use the keyword function, separated by the name of the function and parameters within parenthesis. The part of the function inside the curly braces {} is the body of the function.

Why Functions?

- Functions can be used multiple times, reducing redundancy.
- Break down complex problems into manageable pieces.
- Manage complexity by hiding implementation details.
- Can call themselves to solve problems recursively.

Function Invocation
The function code you have written will be executed whenever it is called.

- Triggered by an event (e.g., a button click by a user).
- When explicitly called from JavaScript code.
- Automatically executed, such as in self-invoking functions.

Function Definition
Before, using a user-defined function in JavaScript we have to create one. We can use the above syntax to create a function in JavaScript.

A function definition is sometimes also termed a function declaration or function statement. Below are the rules for creating a function in JavaScript:

- Every function should begin with the keyword function followed by,
- A user-defined function name that should be unique,
- A list of parameters enclosed within parentheses and separated by commas,
- A list of statements composing the body of the function enclosed within curly braces {}.

Example: This example shows a basic declaration of a function in javascript.

```
1  function calcAddition(number1, number2)
2  {
3      return number1 + number2;
4  }
5  console.log(calcAddition(6,9));
6
```

Output:

```
15
```

In the above example, we have created a function named **calcAddition**,

- This function accepts two numbers as parameters and returns the addition of these two numbers.
- Accessing the function with just the function name without () will return the function object instead of the function result.

What is Window Object in Javascript?

The Window object represents the window in the browser. The window object is automatically created from the browser. All browsers support the Window object. Global variables are the properties of the Window objects, and global functions are the methods of the Window object. The window object methods are used to retrieve the information from the browser window.

Window object Methods

1. alert()

The alert() method displays the alert box in the window with the OK button. This method is used when you want the information to come through the user.

Ex:

```
alert("Hello...Most welcome");
```

2. blur()

The blur() method removes the focus on the window.

Ex:

```
const myWind = window.open("", "", "width=250, height=200");
myWind.blur();
```

3. focus()

The focus() method focuses on the window.

Ex:

```
const myWindow = window.open("", "", "width=250, height=200");
myWindow.focus();
```

4. setInterval()

This method calls on a function with a specific interval.

setInterval(function, milliseconds, param1,.....)

Ex:

```
setInterval(displayHi, 1000);
function displayHi()
{
  document.getElementById("demo").innerHTML += "Hello world";
}
```

The code will return that displayHi() function within 1 second.

5. clearInterval()

The clearInterval() method that clears the time set by the setInterval() method.

clearInterval(functon_name);

Ex:

```
const mynewInterval = setInterval(mynewTimer, 1000);
function mynewTimer()
{
   const date = new Date();
   document.getElementById("demo").innerHTML = date.toLocaleTimeString();
}
function mynewStopFunction()
{
   clearInterval(mynewInterval);
}
```

The mynewStopFunction() stops the serInterval() method.

6. setTimeout()

The setTimeout() method calls a function after a number of milliseconds. This method calls the function only once.

setTimeout(function, milliseconds, param1, ...)

Ex:

```
const mynewTimeout = setTimeout(mynewGreeting, 5000);
function mynewGreeting()
{
  document.getElementById("demo").innerHTML = "Happy Birthday!"
}
```

This mynewGreeting() method returns Happy Birthday after 5 seconds.

7. clearTimeout()

The clearTimeout() method that clears the time set by the setTimeout() method.

clearTimeout(function_name);

Ex:

```
const mynewTimeout = setTimeout(mynewGreeting, 3000);
function mynewGreeting()
{
 document.getElementById("demo").innerHTML = "Happy Birthday to You !!"
}
function mynewStopFunction()
{
 clearTimeout(mynewTimeout);
}
```

We have to clear the timer within the setTimeout() method.

8. close()

The close() method closes the window
window.close()

Ex:

```
function closenewWin()
{
  mynewWindow.close();
}
```

9. open()

The open() method opens the browser window.
window.open()

Ex:

```
function mynewFunction()
{
   window.open("https://www.cybrosys.com");
}
```

10. atob()

Decodes a base 64 encoded string.
window.atob(encoded);

Ex:

```
let text = "Hello";
let encoded = window.btoa(text);
let decoded = window.atob(encoded);
```

11. btoa()

Encodes a string in base 64.
window.btoa(text);

Ex:

```
let text = "Hello HI!";
let encoded = window.btoa(text);
```

12. confirm()

The confirm method displays a dialog box with a message, Ok button, and Cancel button.
confirm("Click Me!");

Ex:

```
function mynewFunction()
{
        confirm("Press a button!");
}
```

13. getComputedStyle()

This method gets the computed CSS properties and values of an HTML element. This method returns a CSSStyleDeclaration object.
window.getComputedStyle(element, pseudoElement)

Ex:

```
const element = document.getElementById("test");
const cssObj = window.getComputedStyle(element, null);
```

14. getSelection()

Returns the Selection object representing the range of text selected by the user

15. matchMedia()

The matchMedia() method returns the MediaQueryList and it results in the query.
window.matchMedia(mediaQuery)
The media queries can be any of the media features of the CSS

Ex:

```
let text;
If(window.matchMedia("(max-width: 700px)").matches)
{
   text = "The screen";
}
else
{
   text = "The screen wide.";
}
```

16. moveBy()

This method moves a window to a number of pixels relative to its current position.
window.moveBy(x, y)

Ex:

```
let mynewWindow;
function openWin()
{
   mynewWindow = window.open("", "", "width=400, height=400");
}
function moveWin()
{
   mynewWindow.moveBy(250, 250);
}
```

17. moveTo()

This method moves a window to the specified position.
window.moveTo(x, y)

Ex:

```
let mynewWindow;
function openWin()
{
   mynewWindow = window.open("", "", "width=400, height=200");
}
function moveWin()
{
   mynewWindow.moveTo(500, 100);
}
```

We can access the document object using the objects. The properties of window objects are used to retrieve the information about the window that is currently opened, whereas its methods perform specific tasks like opening, minimizing the window, etc.