JavaScript (JS)

National Telecommunication Institute - NTI

Introduction To JavaScript

- *JavaScript is an interpreted scripting language, and it is a free and Open Source.*
- *The most popular on the internet, and works in all major browsers.*
- *Object-based scripting language.*
- *Allows adding interactivity to a web page, such as Form Validation.*
- *JavaScript consists of three main parts:*
  - *ECMAScript that provides the core functionality.*
  - *The DOM that provides interacting with web pages elements.*
  - *The BOM that provides API for interacting with web browsers.*

Client-side vs. Server-side JavaScript

- *When JavaScript is used on a web page, it is executed in the web browsers of user's computers. In this case, JavaScript works as a client-side language.*

- *JavaScript can run on both web browsers and servers. A popular server-side environment for JavaScript is Node.js. Unlike the client-side JavaScript, the server-side JavaScript executes on the server that allows you to access databases, file systems, etc.*

Are Java and JavaScript the Same?

- *Java and JS are two completely different languages in Concept, Design and Coding.*
- *Java is developed by Sun Microsystems*
- *JavaScript is Developed by Netscape.*
- *Java is a powerful and complex programming language - in the same category as C and C++.*
- *JS is a lightweight programming language.*

JavaScript Usage

- *JavaScript is used to:*
  - ✓ *Improve the design.*
  - ✓ *Add interactivity to HTML pages.*
  - ✓ *Client-side validation.*
  - ✓ *Displaying clocks, pop-up windows, and dialog boxes.*
  - ✓ *Create cookies.*
- *JavaScript can be inserted into the <head> section or into the <body> section of the HTML document using the <script> tag.*
- *Also the JavaScript can be attached to the HTML web page as a separated file.*

Adding JS to <head> Section.

Syntax

```
<html>
<head>
<title>.....</title>
    <script language="javascript" type="text/javascript" >
      JavaScript Code Goes Here
    </script>
</head>
<body>.....</body>
</html>
```

Adding JS to <body> Section.

Syntax

```
<html>
<head><title>.....</title></head>
<body>
    <script language="javascript" type="text/javascript" >
      JavaScript Code Goes Here
    </script>
</body>
</html>
```

Linking to JS External file

Syntax

```
<html>
<head>
<title>.....</title>
    <script type="text/JavaScript"  src="file1.js"></script>
</head>
    <body>
    </body>
</html>
```

- *Case Sensitivity*

  Everything in JavaScript including variables, function names, class names, and operators are case-sensitive. It means this means that "x", "X" are not the same.

- *Semicolons*

  All statements should end in a semicolon (;). The semicolon separates one statement from another.

- *White Spaces*

  JavaScript, like HTML, ignores extra white spaces, tabs, and newlines that appear in statements.

JavaScript Comments

JavaScript Comments are used to add information about the code to:

- *Make code easy to understand.*
- *Avoid the code from being executed as the JavaScript comments are ignored by the JavaScript engine.*

JavaScript supports both single-line and multiple line comments.

- *Single line comment*

Example

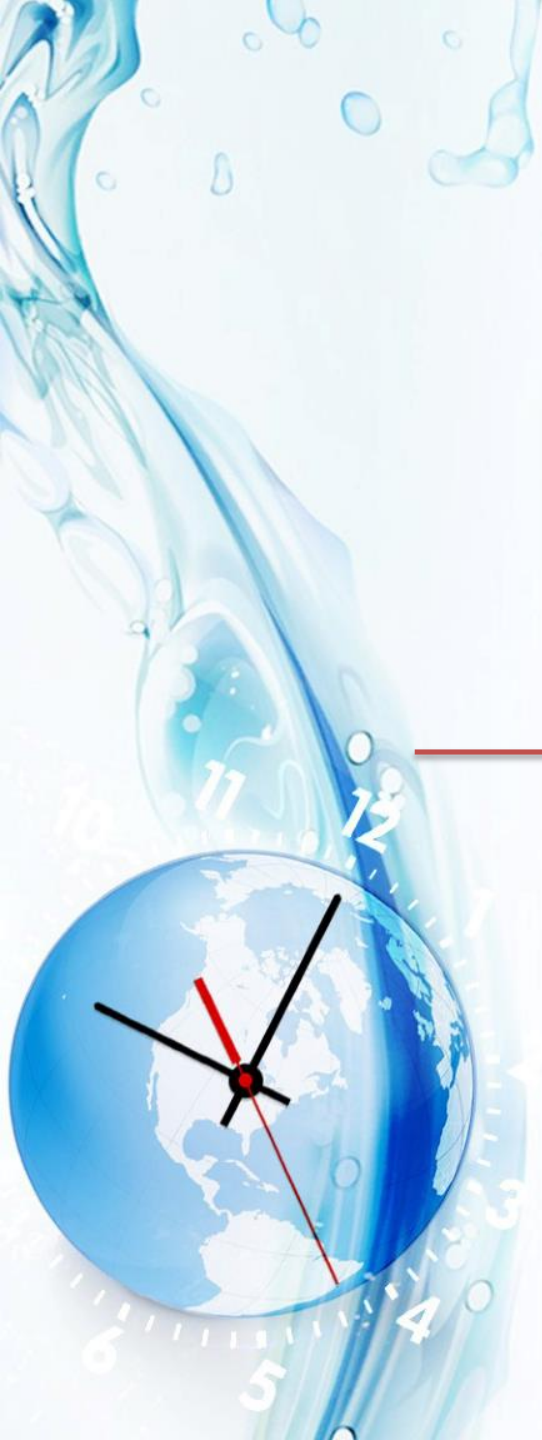
```
// this is a single-line comment
```

- *Multiple line comment*

A block comment starts with a forward slash and asterisk (/) and ends with the opposite (*/).*

Example

```
/*
   This is a block comment that can
   span multiple lines
*/
```



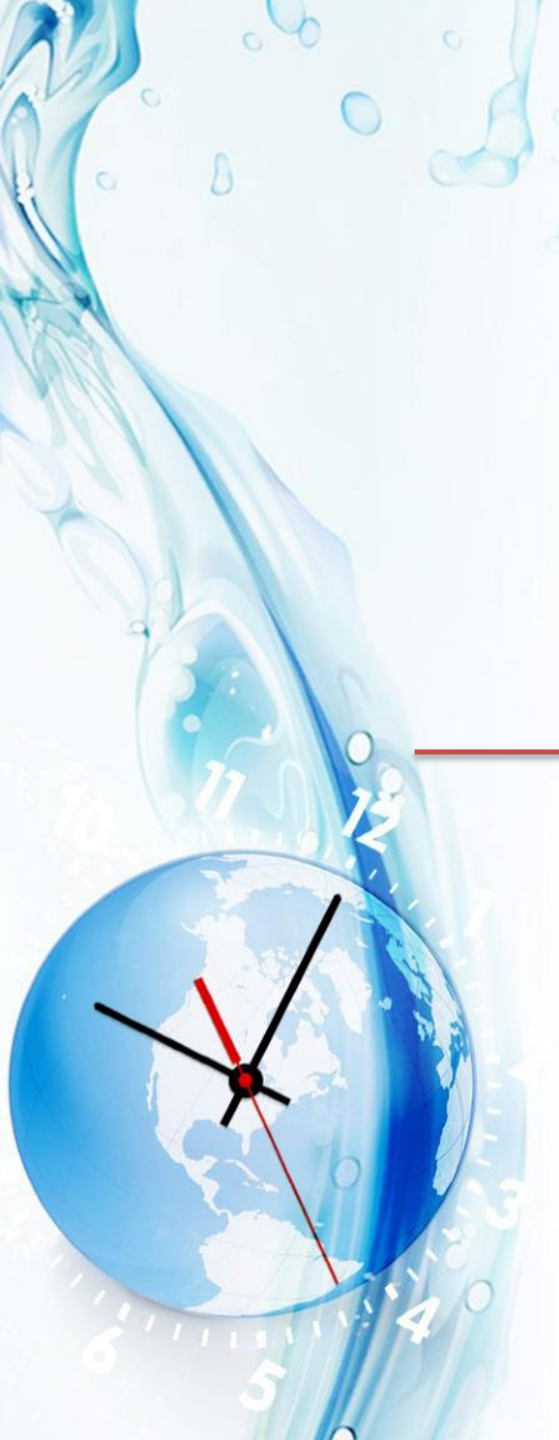
JavaScript's Events

Events are the triggers that call (start) functions. It could be an action such as clicking on a button, placing a mouse over an image or Submitting a form.

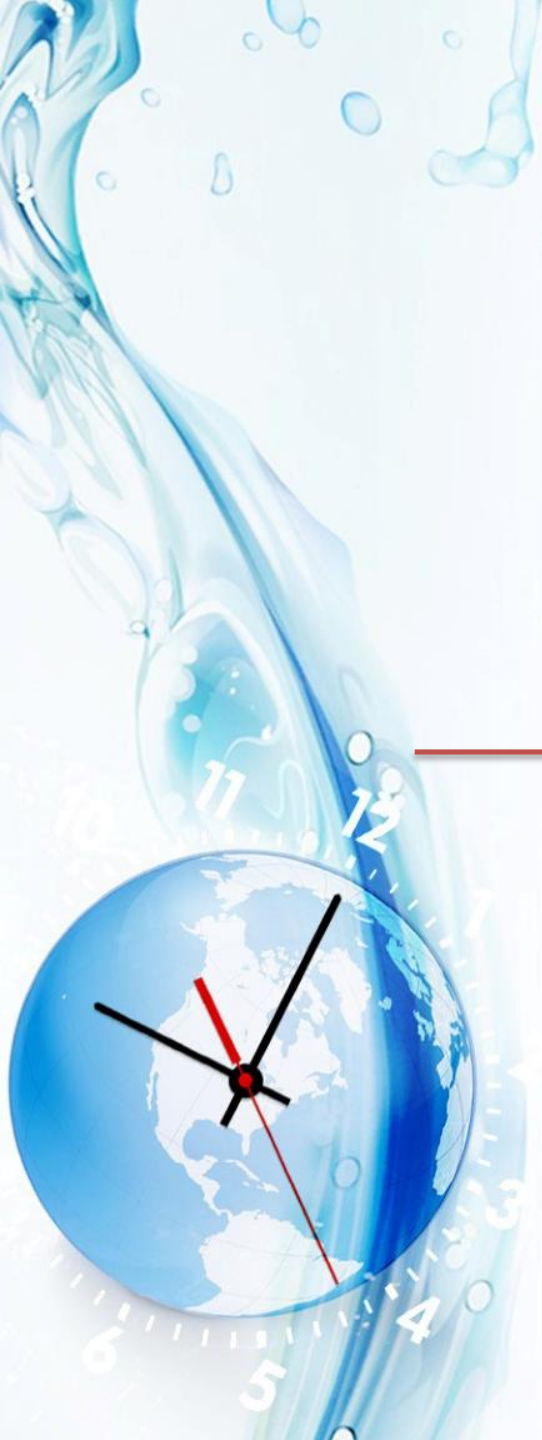
JavaScript's Events

- The available event handlers in JavaScript are:

- onClick( )
- onSubmit( )
- onMouseOver( )
- onMouseOut( )
- onFocus( )
- onChange( )
- onBlur( )
- onLoad( )
- onUnload( )



Js Alerts, Prompts, and Confirms.



JS Alerts

Alert is “modal window” (mini-window) with a message and “ok” button. The word “modal” means that the visitor can’t go through the rest of the page, until he press “OK”.

JS Alerts

Syntax

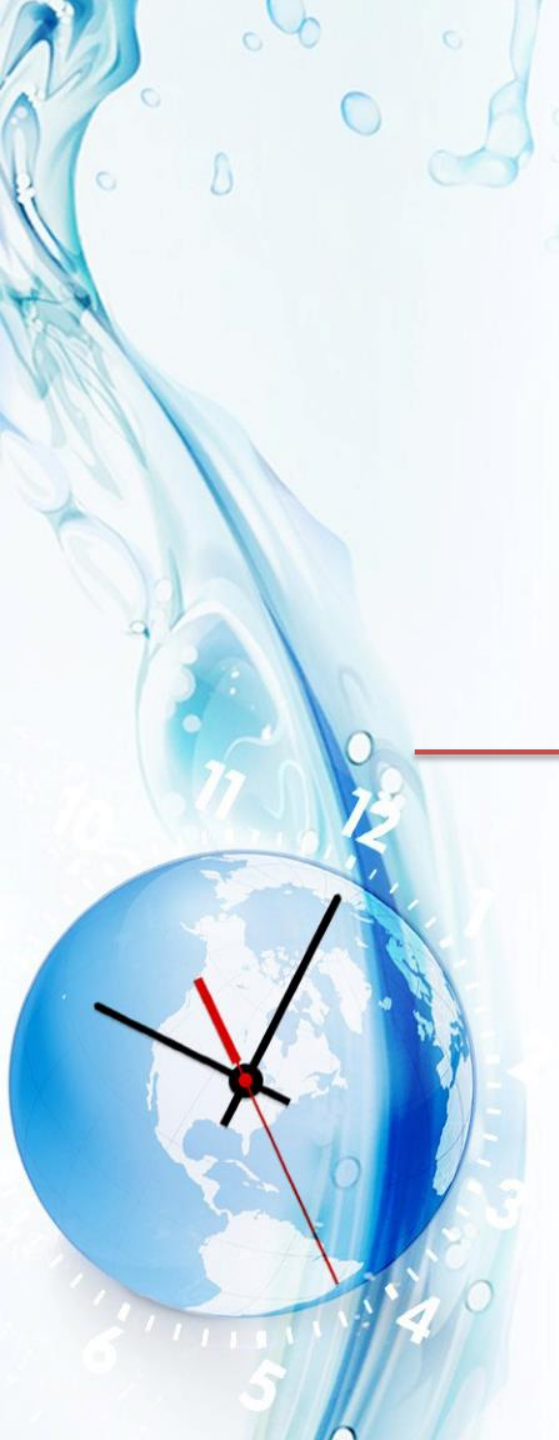
```
alert("message");
```

Example

```
<body>
    <script>
        alert("Hello World!");
    </script>
</body>
```

Example: With Button

```
<body>
    <input type="button" value="click me"
     onclick="alert('Welcome!');">
</body>
```



JS Prompt

Prompt is a modal window with a text message, an input field, and "OK/Cancel" buttons.

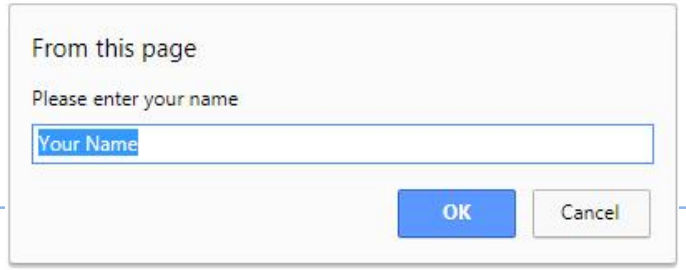
JS prompt

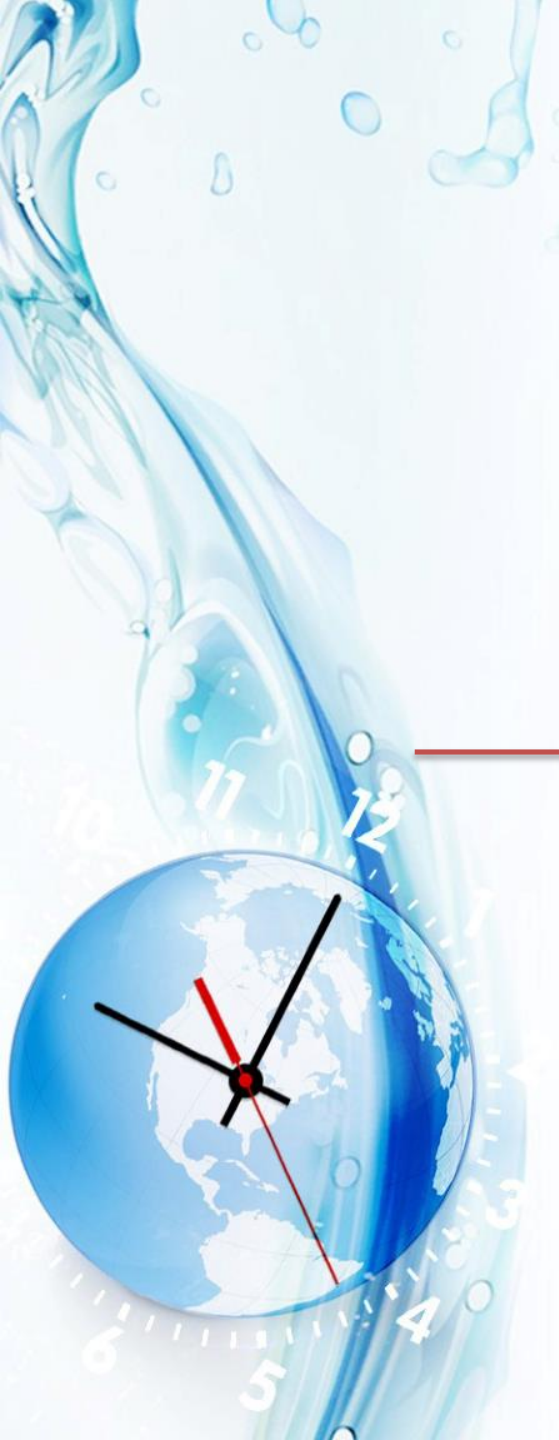
Syntax

```
prompt("message", "default value");
```

Example

```
<body>
<script>
    prompt('Please enter your name','Your Name');
</script>
</body>
```





JS Confirm

Prompt is a modal window with a question and two buttons: “OK” and “Cancel”.

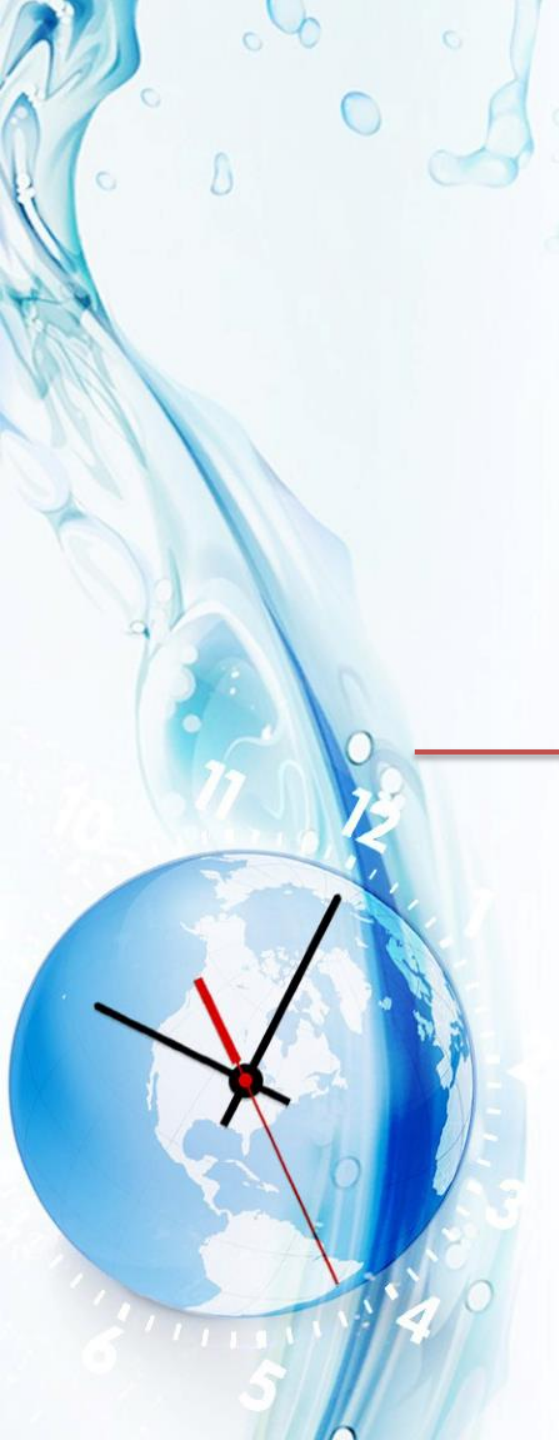
JS Confirm

Syntax

```
confirm("question");
```

Example

```
<body>

<script >

    confirm('Are you really over 21 years of age?');

</script>

</body>
```



Generating Js Outputs

Generating Js Outputs

In JavaScript there are several different ways of generating output including:

- *Displaying Output in Alert Dialog Boxes.*
- *Writing Output to the Browser Window.*
- *Writing Output to Browser Console.*
- *Inserting Output Inside an HTML Element.*

Displaying Output in Alert Dialog Boxes

Alert dialog boxes can be uses to display the message or output data to the user. An alert dialog box is created using the alert() method.

Syntax

```
alert("output");
```

Example

```
alert("Hello World!");
```

Writing Output to the Browser Window

- You can use the document.write() method to write the content to the current document.
- The write() method is mostly used for testing: If it is used after an HTML document is fully loaded, it will delete all existing HTML.

Syntax

```
document.write("output");
```

Writing Into HTML

Syntax

```
document.write("output");
```

Example

```
<body>
<script>
    document.write("<h1>This is a heading</h1>");
    document.write('<p>This is a paragraph</p>');
</script>
</body>
```

Writing Output to Browser Console

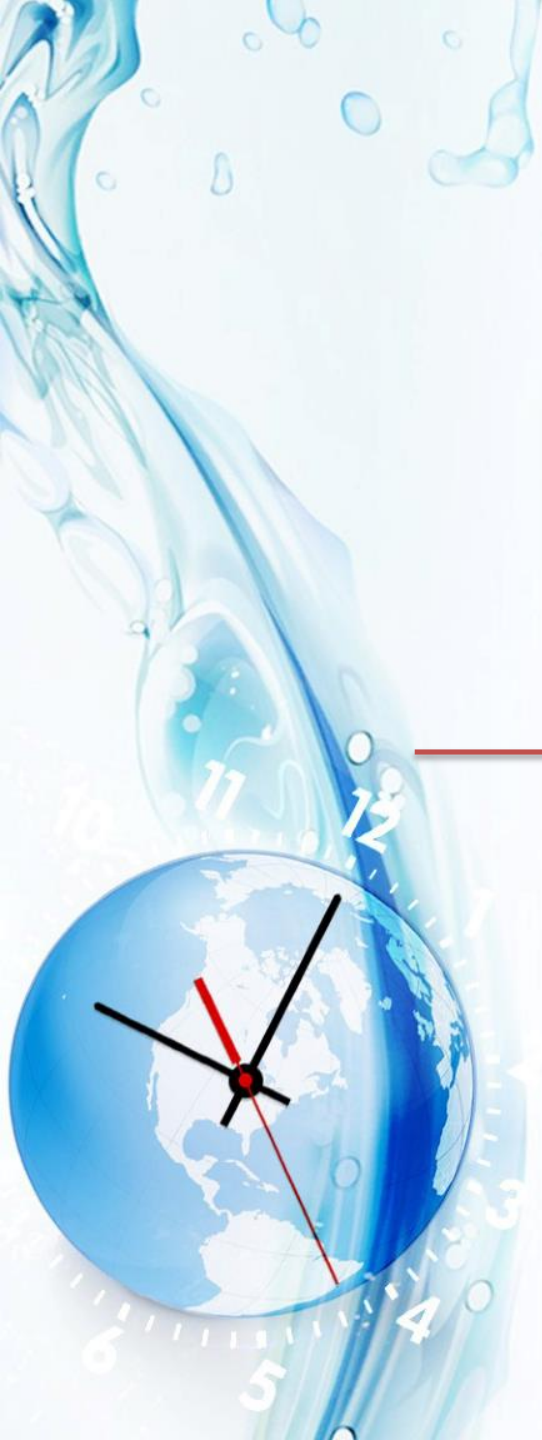
- Message and Data can be easily written to the browser console using the console.log() method. This is a simple, but very powerful method for generating detailed output.

Syntax

```
console.log("output");
```

Example

```
console.log("Hello World!");
```



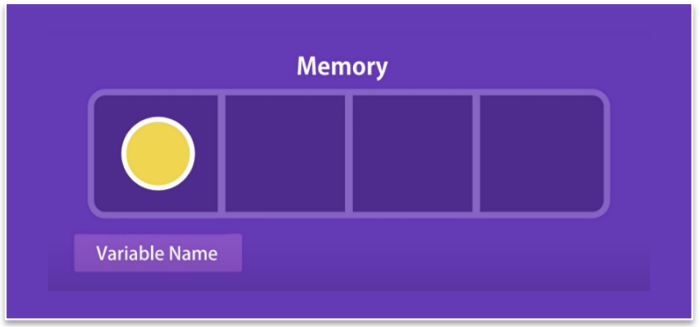
JavaScript's Variables

Variables are just named placeholders for values.

JavaScript variables are loosely typed; i.e. variables can hold values with any type of data.

JavaScript's Variables

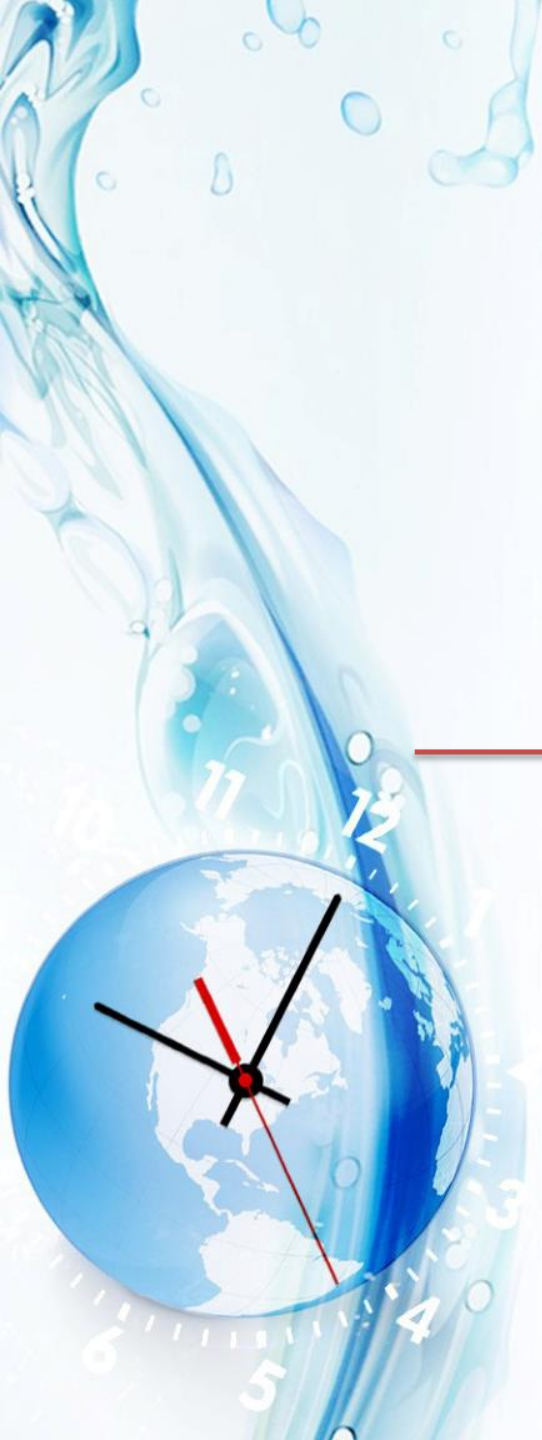
- *Like other programming languages, JS has variables. Variable is simply a name of storage container (location) used to store data value and then refer to the data simply by naming this container.*
- *The Rules for constructing variables names:*
  - ✓ *Cannot be a reserved word.*
  - ✓ *Can only contain letters, digits, underscores, and dollar signs and cannot contain space or hyphen (-).*
  - ✓ *Must begin with a letter or with $ or with _*
  - ✓ *Variable names are case-sensitive.*



JavaScript's Variables

JavaScript Reserved Words

abstract	else	instanceof	switch	volatile	void
boolean	enum	int	synchronized	while	double
break	export	interface	this	with	Public
Byte	extends	long	throw	short	return
case	false	native	throws	static	goto
catch	final	new	transient	super	if
char	finally	null	true	implements	debugger
class	float	package	try	import	default
const	for	private	typeof	in	do
continue	function	protected	var	delete	



Using “var” keyword

“var” keyword is used for variable declaration or initialization, once for the life of any variable name in a document.

Declare JavaScript variables using var keyword

To declare a variable, the "var" keyword is used followed by the variable name.

Syntax

```
var variableName;
```

Example

```
<script>
        var x;
</script>
```

- Declares an empty variable named x.
- The value of x will be "undefined".

Assigning Values to Variables

Syntax

```
variableName=value;
```

Example

```
X = "Yahoo";
```

- *The value “yahoo” is assigned to variable x.*

Declaring and Assigning in one step

Syntax

```
var variableName=value;
```

Example

```
var x="yahoo";
```

- Declares a variable named x and assigns the value “yahoo” to the variable x in one step.

Declaring and Assigning many variables in one step

- Two or more variables can be declared using one statement, each variable declaration is separated by a comma (,).

Example

```
var lastname="Ahmed", age=30, job="Driver";
```

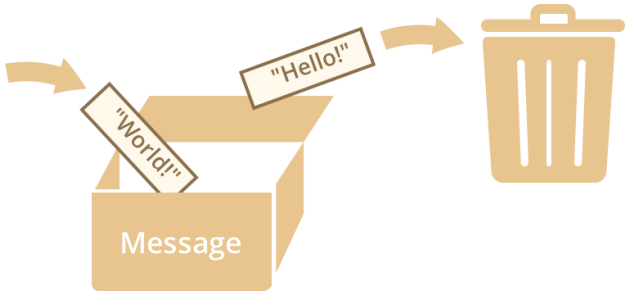
Same as

```
var lastname="Ahmed",
age=30,
job="Driver";
```

Variable Re-declaring

Example

```
var message="Hello";
var message="World";
```

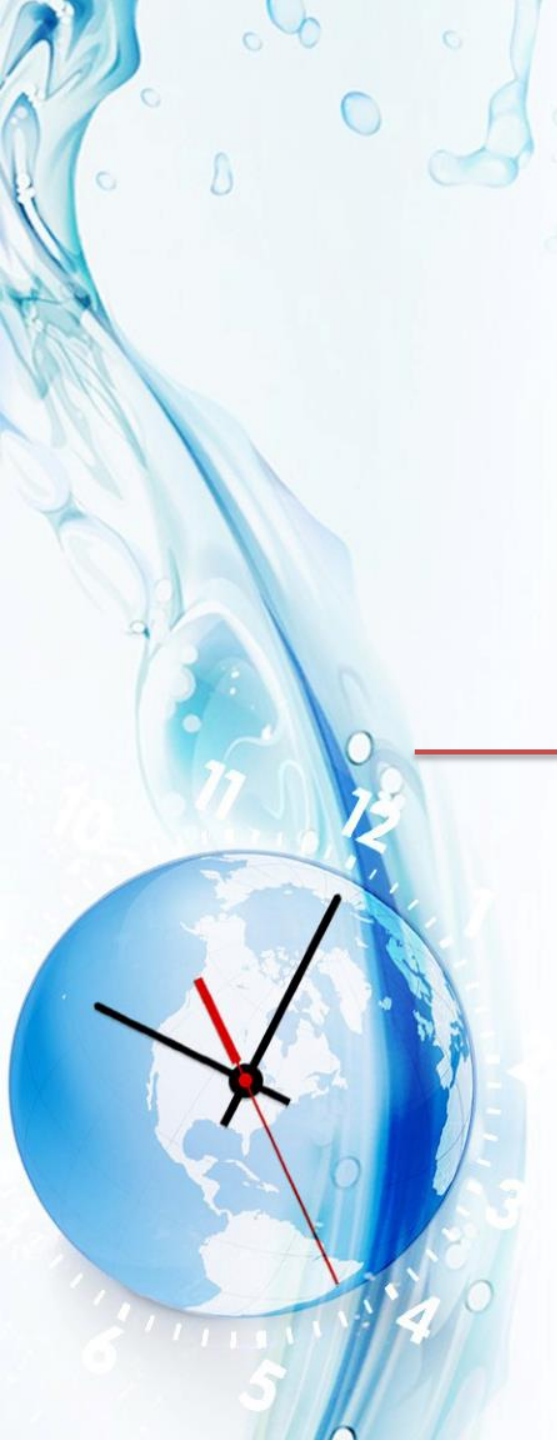


- Note that:

Re-declaring a JavaScript variable, doesn't make the variable lose its value.

Example

```
var lastname="Ahmed";
var lastname;
```



Using “let” keyword

Declare JavaScript variables using let keyword

Starting From ES6, "let" or "const" keywords can be used to declare one or more variables.

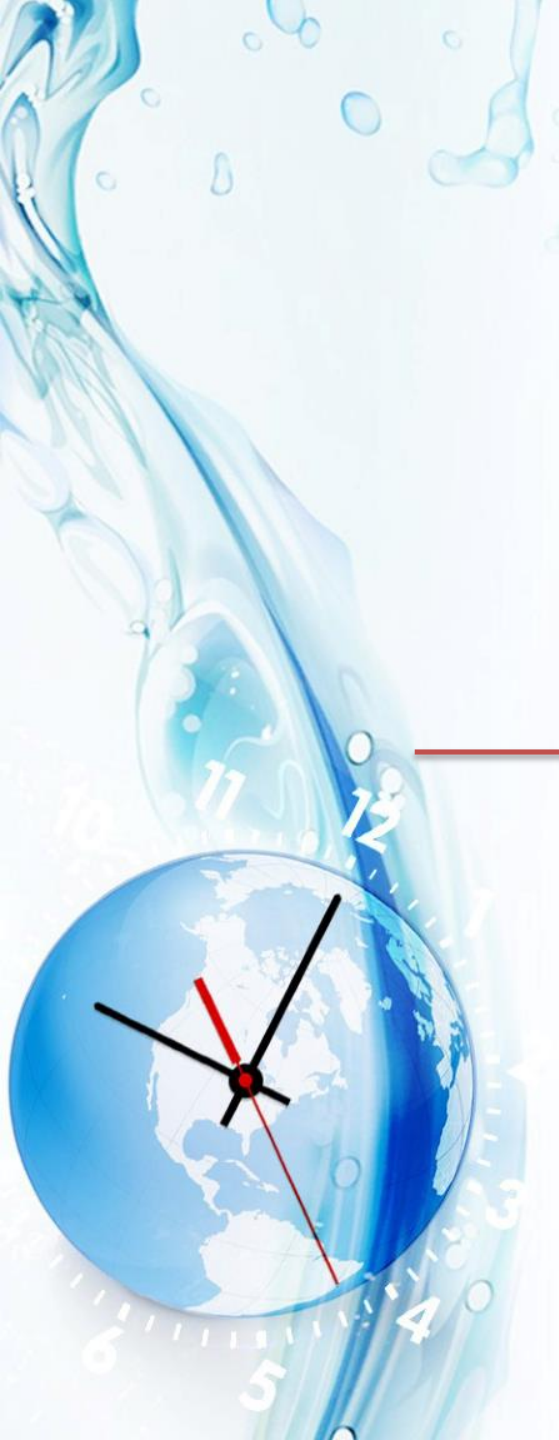
Syntax

```
let variableName=value;
```

Example

```
let x="yahoo";
```

- *Declares a variable named x and assigns a value equals to "yahoo" to the "x" variable.*



Using “const” keyword

Declare JavaScript variables using const keyword

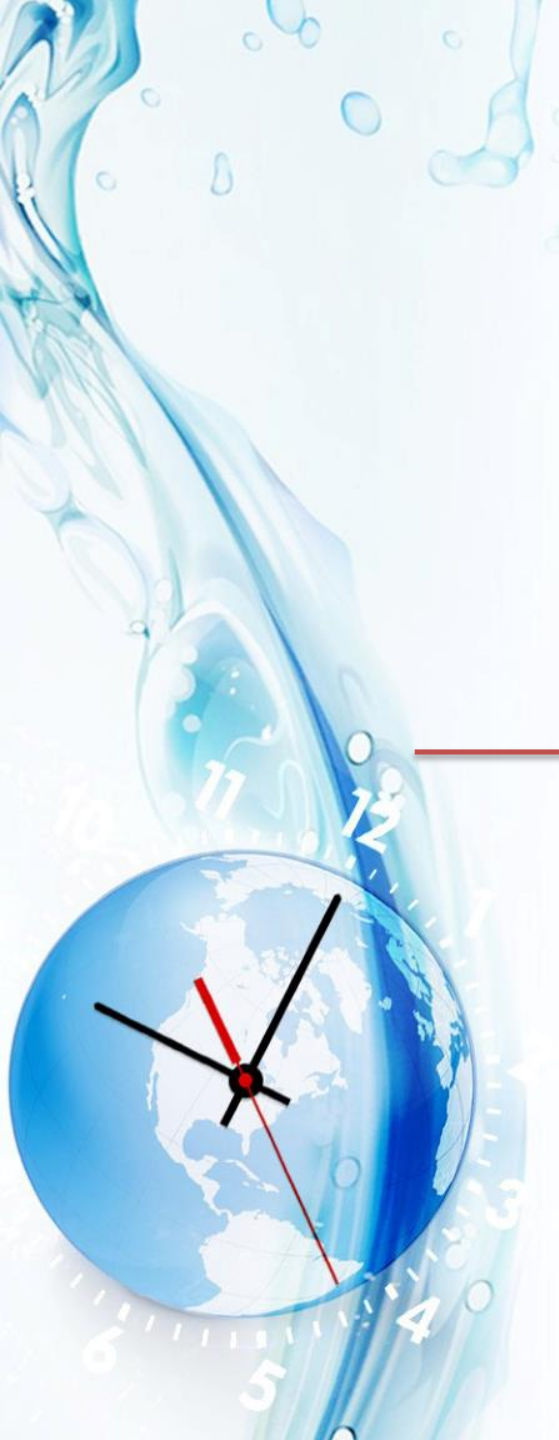
The const keyword works like the let keyword, excepts that the variable declared must be initialized immediately with a value, and that value can't be changed ever.

Syntax

```
const  variableName = value;
```

Example

```
const  x="yahoo";
const  x="hotmail";          // retruns error
```

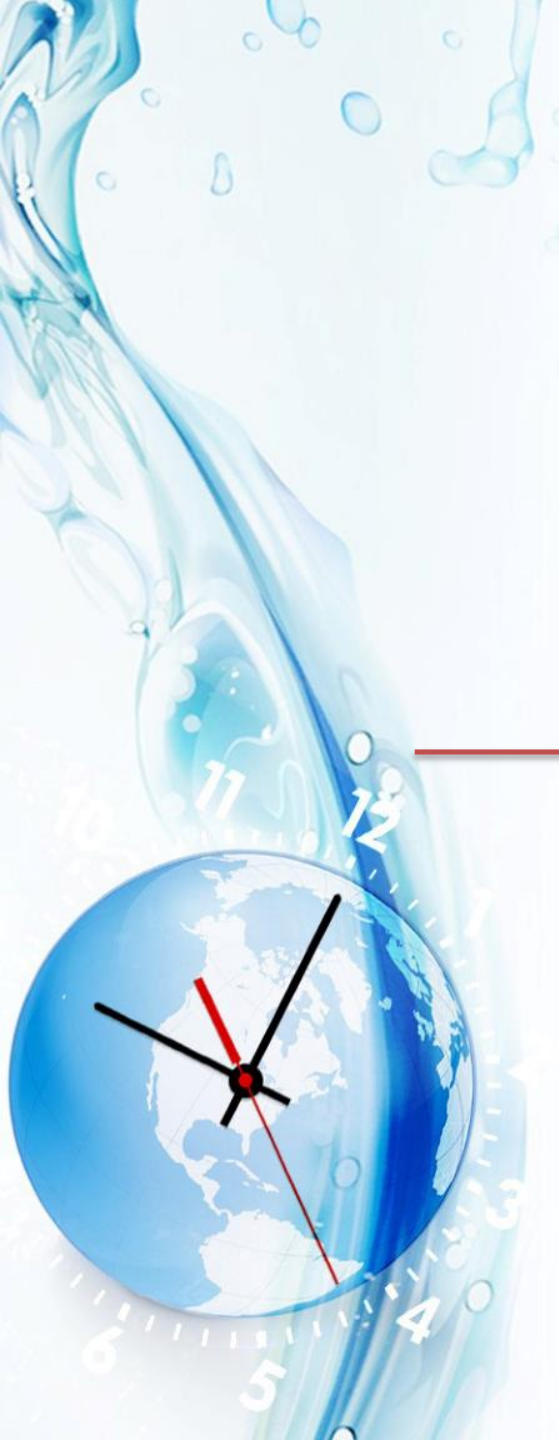


JS Data Types

Data types basically specify what kind of data can be stored and manipulated within a program.

JS Data Types

- JavaScript provides different data types to hold different types of values. There are three types of data types in JavaScript
  - Primitive (primary) data types.
    - String, Number, Boolean.
  - Reference (composite) data types.
    - Array, Object, Function.
  - Special data types.
    - Undefined and Null.
- Js is a dynamic language, i.e. you don't need to specify type of the variable because it is dynamically identified by the JavaScript engine.



Primitive (primary) Data Types

Primitive data types are the data types that can hold only one value at a time.

JS String Data Type

- The string data type is used to represent textual data (i.e. sequences of characters). Strings are created using single or double quotes surrounding one or more characters.

Example 1

```
var str = 'JavaScript';          // Single quotes
```

Example 2

```
var str = "JavaScript";          // Double quotes
```

The “typeof” Operator

The “typeof” operator is used to get the data type of a JavaScript variable.

Syntax

```
typeof variableName
```

Example

```
var x='JavaScript';
console.log(typeof x);          // string
```

JS Number Data Type

- The number data type is used to represent positive or negative numbers with or without decimal place, or numbers written using exponential notation.
- To use numbers don't use any type of quotes.

Integers

```
var num = 100;
```

Floating-point numbers (decimal)

```
var num = 15.2;
var num = 18.00;          // interpreted as integer 18
```

- To represent octal (base 8) numbers, use first digit as zero (0) followed by octal digit numbers (0 to 7), or use first two digits as (0o) followed by octal digit numbers.

Integers - octal

```
var oct = 060;
    console.log(oct);          // 48
var oct = 0o60;
    console.log(oct);          // 48
```

JS Number Data Type

- To represent hexadecimal (base 16) numbers, use (0x) (in lowercase) as the first two digits followed by any number of hexadecimal digits.

Hexadecimal Numbers

```
var num = 0xff;
    console.log(num);                    // 255
```

- JavaScript allows you to use the e-notation to represent very large or small numbers as in the following example.

e-notation Numbers

```
var num = 2.15e6;
    console.log(num);                    // 2170000
```

JS Number Data Type - Special Values

- *The JS Number Data Type also includes some special values:*
  - *Infinity, -Infinity:*

    represents the mathematical Infinity ∞, which is greater than any number. Infinity is the result of dividing a non-zero number by 0.
  - *NaN:*

    represents Not-a-Number value. It is a result of an invalid or an undefined mathematical operation, like taking the square root of -1 or dividing 0 by 0, or using strings inside a mathematical equations.

JS Number Data Type - Special Values

Infinity, -Infinity

```
console.log(16 / 0);          // Infinity
console.log(-16 / 0);         // -Infinity
console.log(16 / -0);         // -Infinity
```

NaN

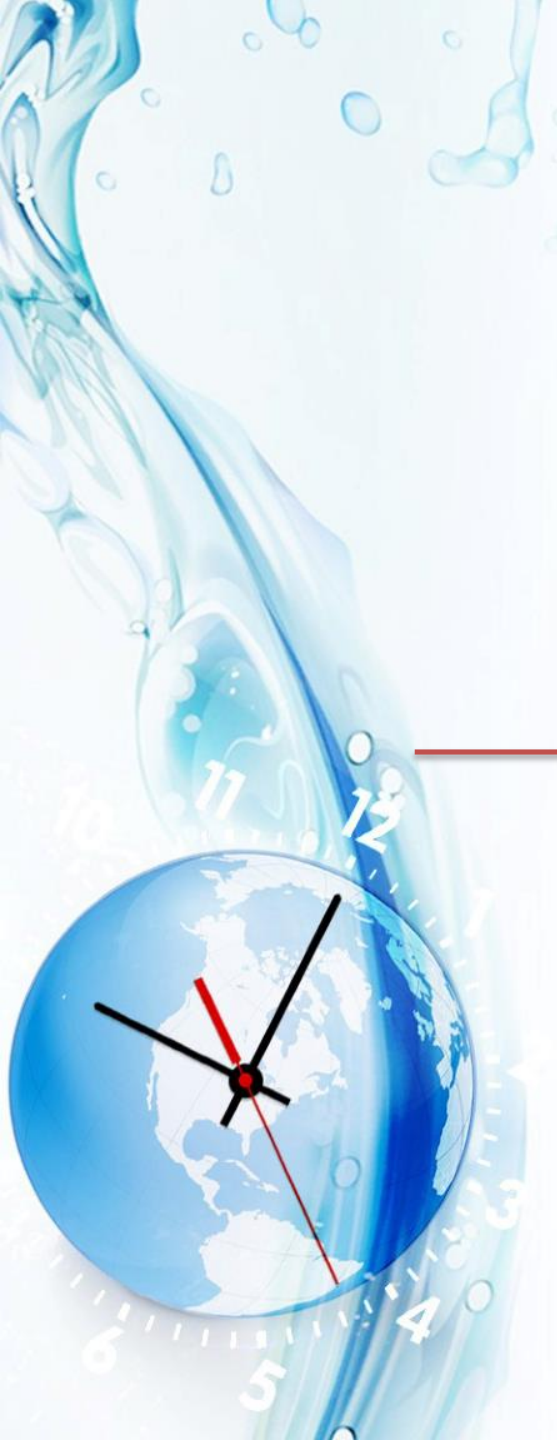
```
console.log("Text" / 2);      // NaN
console.log("Text" / 2 + 10); // NaN
```

JS Boolean Data Type

- *The Boolean data type can hold only two values: true or false. It is typically used to store values like yes (true) or no (false), on (true) or off (false).*

Boolean

```
var num1 = true;
var num2 = flase;
    console.log(num1);              // true
    console.log(typeof num1);       // Boolean
```



Special Data Types

The “undefined” Data Type

The undefined data type can only have one value named “undefined”. If a variable has been declared, and no value is assigned to it, then its value will be “undefined”.

X value is Undefined

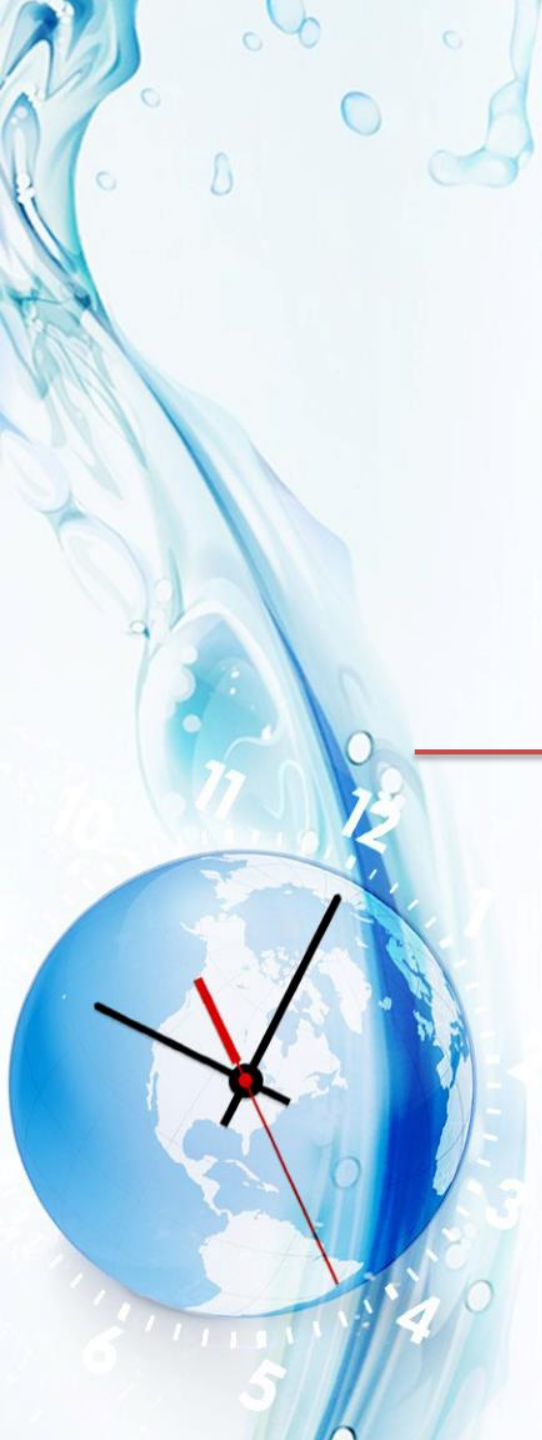
```
var x;
console.log(x);       // undefined
```

JS “null” Data Type

- *“null” is another special data type that can have only one value the “null” value. A null value means that there is no value. It is not equivalent to an empty string ("") or 0, it is simply nothing.*
- *A variable can be explicitly emptied of its current contents by assigning it the null value.*

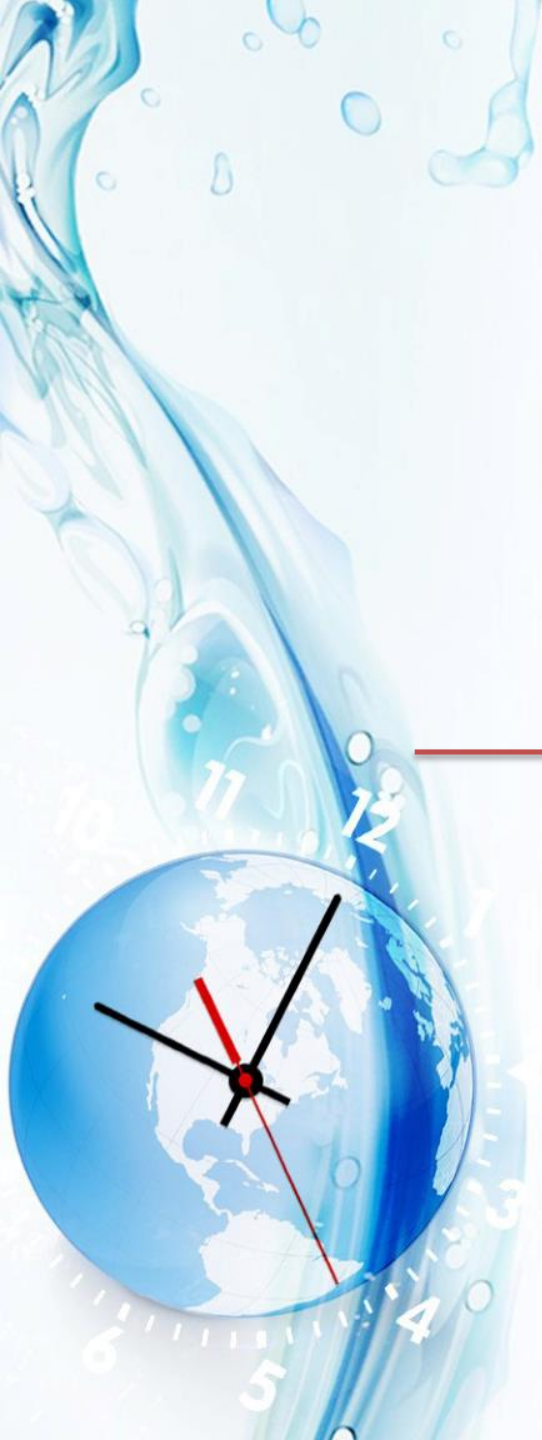
null

```
var y=50;
        console.log(y);          // 50 ➔ number
y=null;                          // explicitly emptied
        console.log(y);          // null
```



Reference (composite) Data Types

Reference data types are the data types that can hold collections of values and more complex entities.

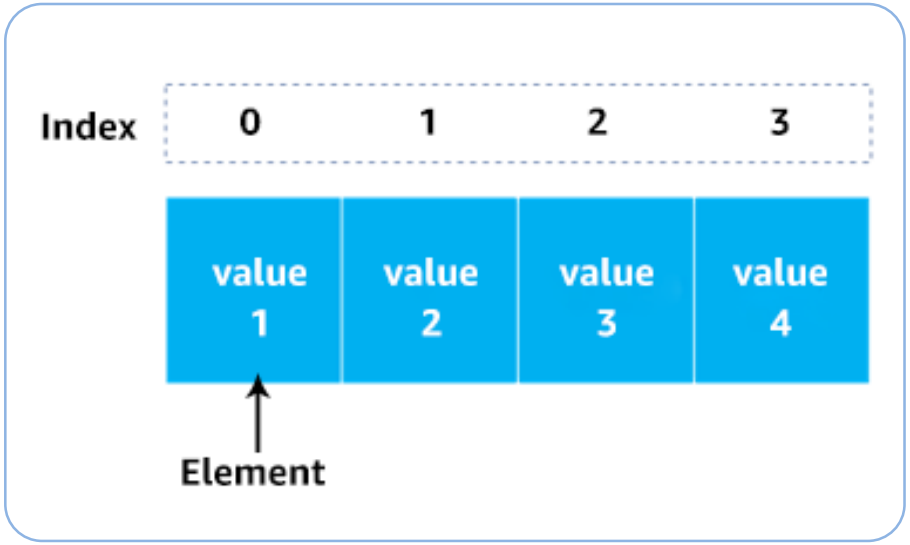


JS Array Data Type

Arrays are complex variables that allows to store more than one value or a group of values under a single variable name.

JS Array Data Type

- *In JavaScript, an array is an ordered list of values. Each value is called an element specified by an index.*
- *<u>There are three ways to create array in JavaScript:</u>*
  - ✓ *Array Literal.*
  - ✓ *Array Direct Instance.*
  - ✓ *Array Constructor.*
- *Let us create an array named "colors" and Insert the (Red-Green-Blue) values inside this array.*



Create Array by Array Literal

Syntax

```
var arrayName = [element0, element1, ... , elementN];
```

Example

```
var colors=['Red','Green','Blue'];
```

Create Array by Array Direct Instance

Syntax

```
var arrayName = new Array()
arrayName[index] = "value";
```

Example

```
var colors = new Array()
    colors[0] = 'Red';
    colors[1] = 'Green';
    colors[2] = 'Blue';
```

Create Array Array Constructor

Syntax

```
var arrayname = new Array(element0, element1, ... , elementN);
```

Example

```
var colors = new Array('Red' , 'Green' , 'Blue');
```

Accessing array elements

Syntax

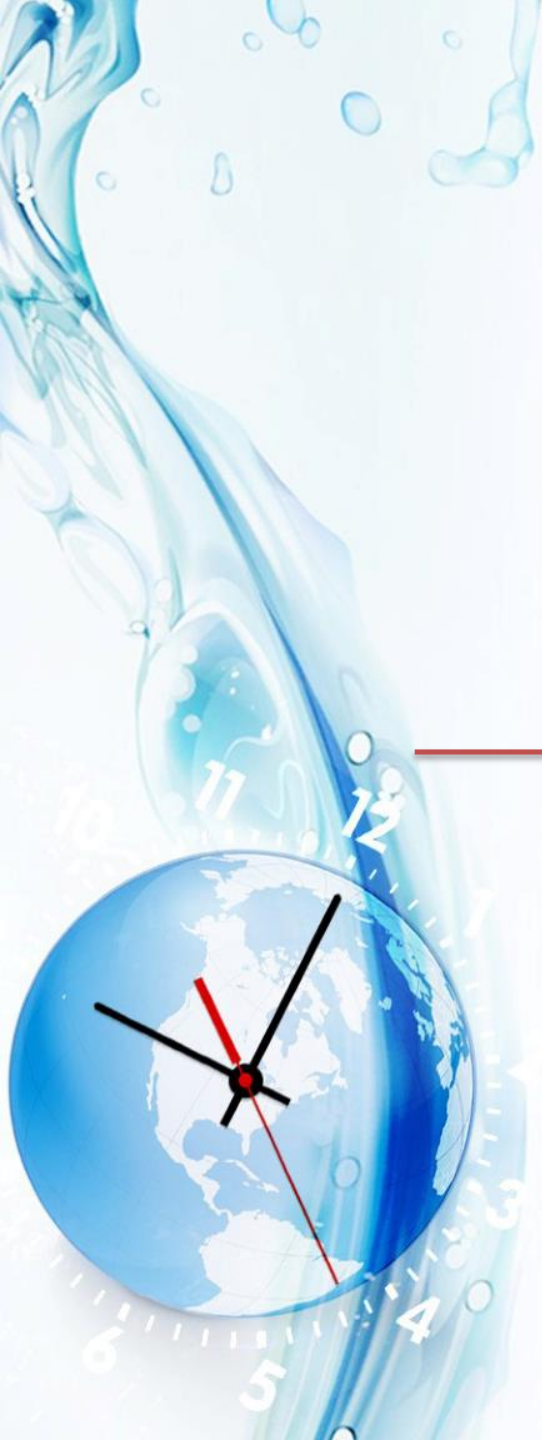
```
arrayName[index];
```

Example

```
colors[0];
```

Example

```
var colors = ['Red','Green','Blue'];
        // Creates new array and assigns values to it
console.log(colors);
        // prints all the array values separated by comma
console.log(colors[0]);
        // prints the first value in the array
console.log(colors[1]);
        // prints the second value in the array
```



JS Object Data Type

JavaScript is an object-based language and in JS almost everything is an object or acts like an object. So, to work with JavaScript effectively and efficiently we need to understand how objects work as well as how to create your own objects and use them.

JavaScript Object Data Type

- When dealing with multiple related variables, these variables can be inserted inside an object.
- An object contains properties, defined as a key-value pair. A property key (name) is always a string, but the value can be any data type, like Strings, Numbers, Booleans.

Syntax

```
var objectName = {Property : 'value', Property : 'value'};
```

JavaScript Object Data Type

Examples

```
var person = {};
        // Create Empty Object
```

```
var person = { firstName : 'Ahmed' , lastName : 'Ali' , age : 30 };
        // Create Object with properties and values
```

```
var person = {
    firstName : 'Ahmed',
    lastName : 'Ali',
    age : 30
    };
// For better reading
```

Changing Object's Values

Dot Notation Syntax

```
objectName.property = val;
```

Example

```
person.age = 30;
```

or: Bracket Notation Syntax

```
objectName['property'] = val;
```

Example

```
person['age'] = 30;
```

Printing Object's Values

Example

```
<script>
    console.log(person.age);
            // prints the age value for the person object using Dot
            Notation
    console.log(person['age']);
            // prints the age value for the person object using Bracket
            Notation
</script>
```

Deleting Object's Properties

- *The "delete" operator can be used to completely remove an object property, (remove the property and the value).*

Example

```
var person = {
    firstName : 'Ahmed' , lastName : 'Ali' , age : 30
    };
delete person.lastName;
console.log(person.lastName);                    // undefined
```

Manipulating by Value vs. Reference

JS objects are reference types that mean when you make copies of them, you're really just copying the references to that object. Whereas primitive values like strings and numbers are assigned or copied as a whole value.

Manipulating by Value vs. Reference

Example

```
var message = "Hello World!";
var greeting = message;          // copy message value to greeting
message = "Hi, there!";          // change value of greeting
console.log(message);            // Hi, there!
console.log(greeting);           // Hello World!
```

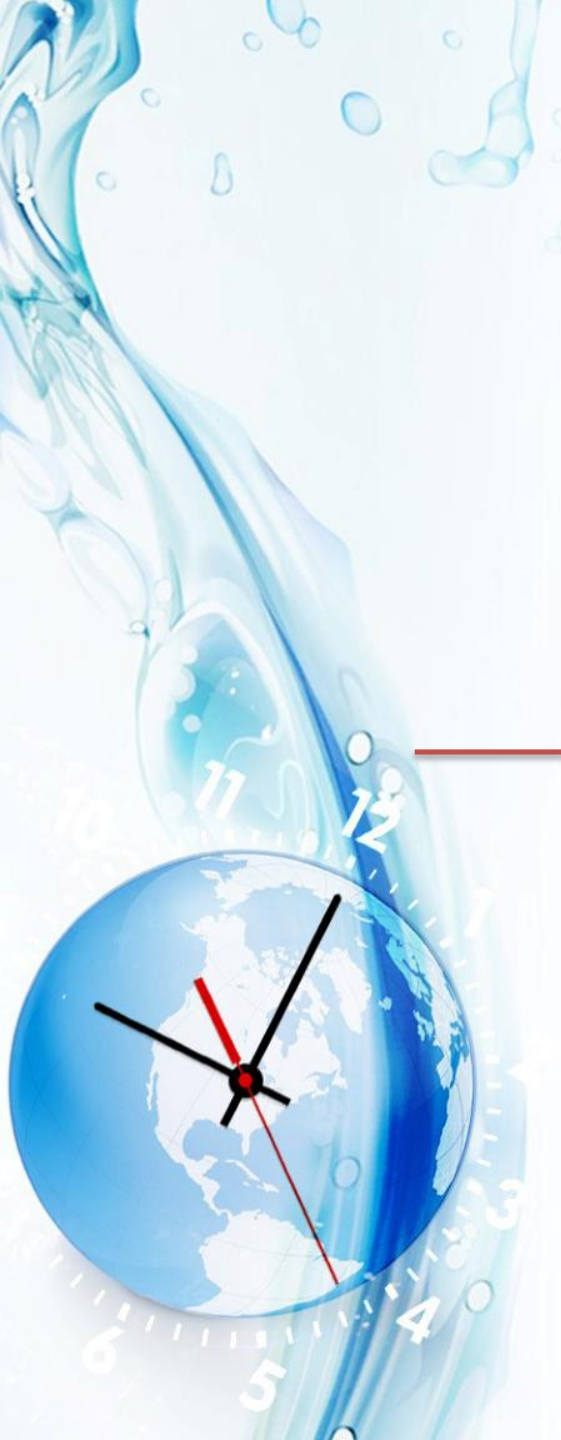
- Create a variable name “message”.
- Create a copy from “message” to another variable called “greeting”.
- Change “message” value and print both of the two variables.
- ➔ the value of “message” is changed to the new value but the greeting still holds the old one. This means that the value is actually copied to the variable and not only a reference to it.

Manipulating by Value vs. Reference

- When repeating the same example with an object notice that any changes made to the variable “user” also change the “person” variable; it happens because both variables reference the same object. So, simply copying the object does not actually clone it but copies the reference to that object.

Example

```
var person = { firstName : 'Ahmed' , lastName : 'Ali' , age : 30 };
var user = person;
user.firstName = "Fady";
console.log(person.firstName);          // Fady
console.log(user.firstName);            // Fady
```



JS Functions

- *A function is a block of code that will be executed when the function is invoked:*
  - ✓ *By an event.*
  - ✓ *By a JavaScript code.*
  - ✓ *Automatically (self invoked).*
- *There are mainly two advantages of JavaScript functions.*
  - ✓ *Code reusability: The function can be called several times.*
  - ✓ *Less coding: We don't need to write many lines of code each time to perform the same task.*

JS Functions

Syntax

```
function functionName()
    {
     // Code to be executed
    }
```

Example

```
<body> <script>
        function myFunction()
                {
                alert("Hello World!"); }
</script>
<input type = "button" value = "click here" onclick="myFunction();">
```

Functions with Parameters

- When calling a function, some values can be passed to it, these values are called arguments or parameters.
- Parameters sent separated by commas (,)

Syntax

```
function myFunction(parameter1 , parameter2 , parameter3)
    {
            // Code to be executed
    }
```

Functions with Parameters

Example

```
<body>
<script>
    function myFunction(x , y)
        {
            var x, y, z;
            z=x+y;
            alert(z);
        }
</script>

<input type="button" value="Try it" onclick="myFunction(5 , 8);">
</body>
```

Default Values for Function Parameters

- With ES6, default values can be specified to the function parameters. This means that if no arguments are provided to function when it is called these default parameters values will be used.

Example

```
function myFunction(user = 'Guest')
        {
                alert('Hello ' + user);
        }
myFunction();                   // Hello Guest
myFunction('Ahmed');            // Hello Ahmed
```

Returning Values from a Function

- A function can return a value back to the script that called the function as a result using the return statement.
- The value may be of any type, including arrays and objects.

Example 1

```
function myFunction()
    {
        var x=5, y=15;
        var z = x + y;
        return z;
    }
var m = myFunction()
console.log(m);                    // 20
```

Returning Values from a Function

Example 2

```
<html>
<body>
    <script>
        function myFunction(a , b)
            {
                return a * b;
            }
        var y = myFunction(4 , 3)
        console.log(y);     // 12
    </script>
</body>
</html>
```

Self Invoked Functions

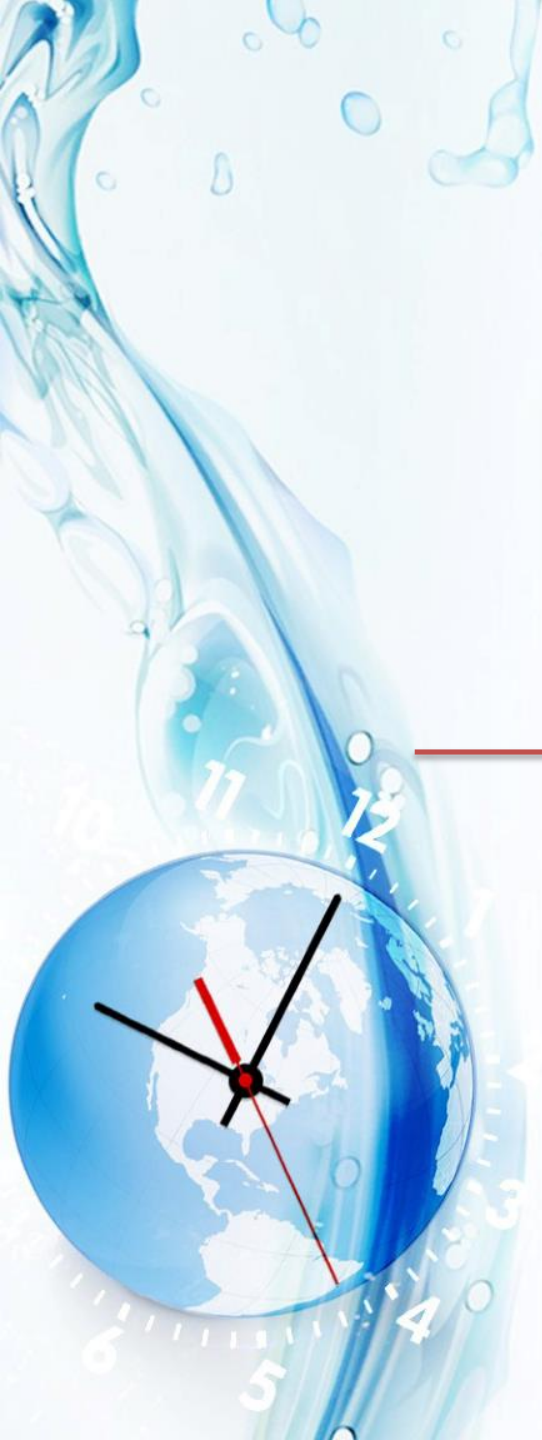
- A self-invoked function (anonymous function) is the function that runs automatically when you create it and has no name.

Syntax

```
(function () {
  // some code
}());
```

Same as

```
(function () {
  // some code
})();
```



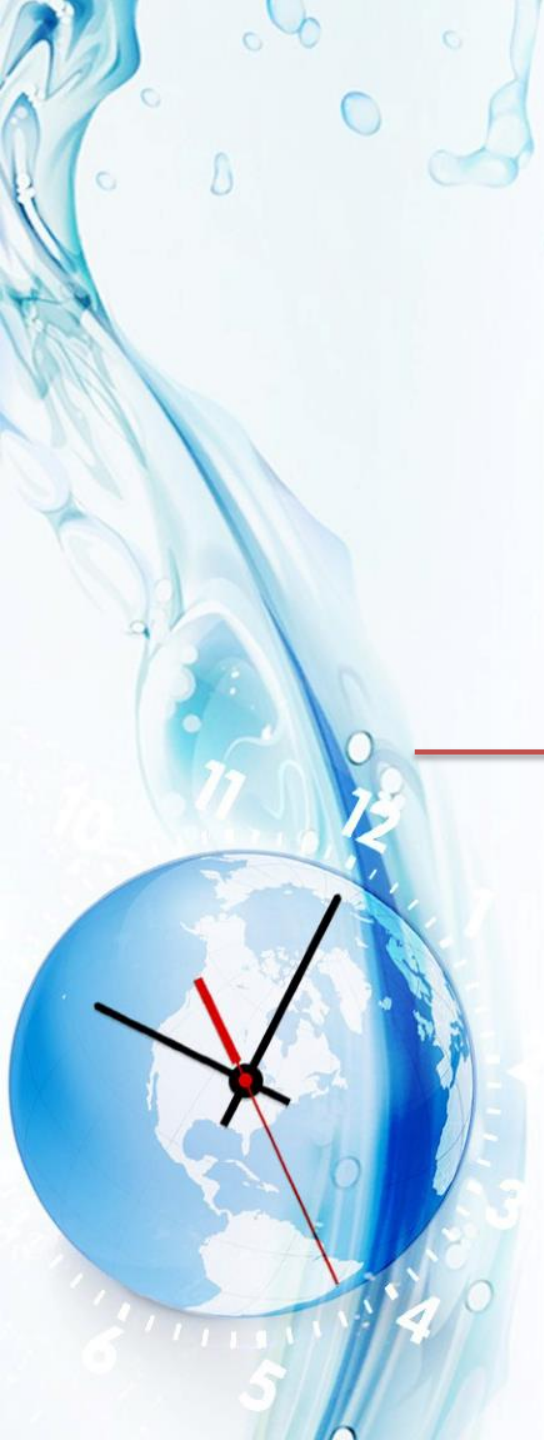
JS Function Expressions

The syntax that we've used before to create functions is called function declaration. There is another syntax for creating a function that is called a function expression.

JS Function as Data Type

Example

```
// declare variable with function data type
        var greeting = function() {
                    return "Hello World!";
                    }
var x=greeting();                   // invoke the function
console.log(typeof greeting);       //  function
console.log(x);                     //  Hello World!
```



JS Scope

In JavaScript, all variables exist within a scope that determines the lifetime of the variables and which part of the code can access them.

JS mainly has "Global" and "Local" scopes and ES6 introduced a new scope called "Block" scope.

JS Global Variables

The Global variable is the variable that can be accessed from any block of the code:

- *Declared outside the function.*
- *All scripts and functions can access it.*
- *Deleted when the page is closed.*
- *Can be declared inside a function by assigning value direct without using "var".*

JS Local Variables

The local variable is a variable declared inside a JavaScript function:

- *Created inside the function using the keyword "var".*
- *Accessed by that function only.*
- *Deleted by the end of the function.*

Example

```
function myFunction()
    {
        var x=5;
        console.log(x);          // local variable (value is 5)
    }
console.log(x);                  // undefined
```

Variable Shadowing

when a Local variable declared with the same name as a Global variable and prevent the inner block from accessing the global variable. the Global scope variable is said to be shadowed by the "Local" scope variable.

Example

```
var message = "Hello";                  // global variable
    function sayHi() {
        var message = 'Hi';             // local variable
        console.log(message);           // Hi
    }
sayHi();
console.log(message);                   // Hello
```

Variable Shadowing

- *In the previous example, The two variables share the same name "message". The first one is a global variable and the second one is local variable.*
- *Inside the sayHi() function, the global "message" variable is shadowed by the local variable; i.e. It is not accessible anymore inside the sayHi() function, but It is accessible outside of the function.*

Accessing global variable inside the function

In this example, global variable named message is declared. In the sayHi() function, the global message variable is reference by omitting the var keyword and its value is changed “Hi”.

Example

```
var message = "Hello";                    // global variable
    function sayHi() {
        message = 'Hi';                    // var is not used
        console.log(message);
    }
sayHi();                                  // Hi
console.log(message);                     // Hi
```

JS Scopes

Example

```
var x=3;
    function myFunction()
        {
        var y=1;
        s=300;
        }
m=20;
```

Global Variables

x s m

Local Variables

y

Strict Mode

To prevent the function from declaring a new global variable by omitting the “var” keyword, the "use strict"; mode is added at the beginning of the JavaScript file. (function will not able to create new global variables but still able to access them).

Example

```
"use strict";
    function sayHi() {
        message = 'Hi';
        console.log(message);
    }
sayHi();                    // error
```

Although the function is not able to create a new global variable, but it still have the permission to access all the global variables on the main script.

Example

```
"use strict";
var message = "Hello";          // global variable
    function sayHi() {
    message = 'Hi';
                // replaces “Hello” by “Hi” in the global variable
       console.log(message);    // Hi
    }
sayHi();                        // Hi ➔ global
console.log(message);           // Hi ➔ global
```

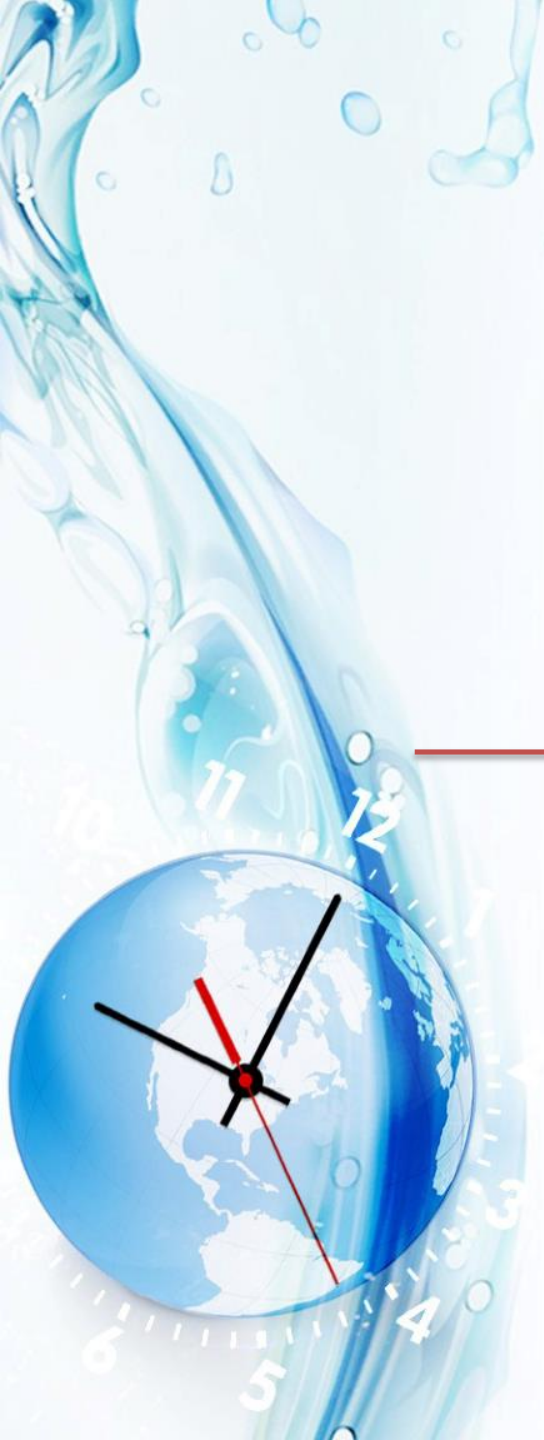
JS Block Variables

The block scope variable is a variable declared with the "let" keyword or "const" keyword.

- *Declared inside a block { }*
- *Can not be accessed from outside the block.*

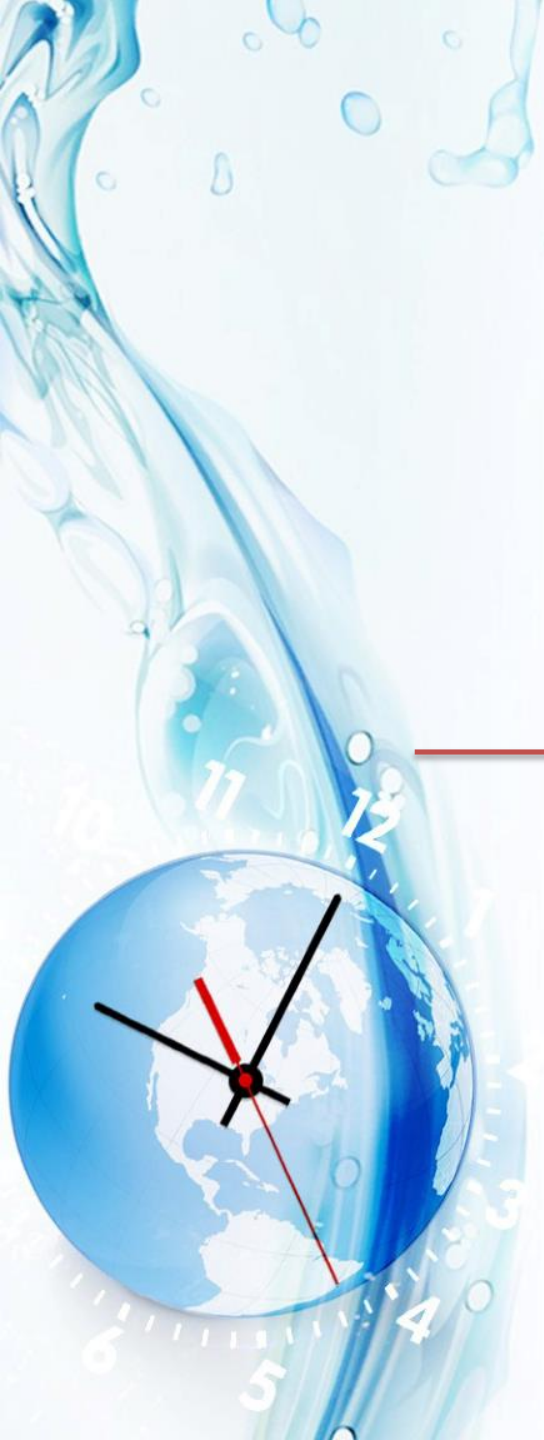
Example

```
{
    let x=5;
    console.log(x);          // block variable (value is 5)
}
    console.log(x);          // undefined
```



JS Operators

Operators are symbols or keywords that tell the JavaScript engine to perform some sort of actions. For example, the addition (+) symbol is an operator that tells JavaScript engine to add two variables or values.



JS Arithmetic Operators

The arithmetic operators are used to perform common arithmetical operations, such as addition, subtraction, multiplication.

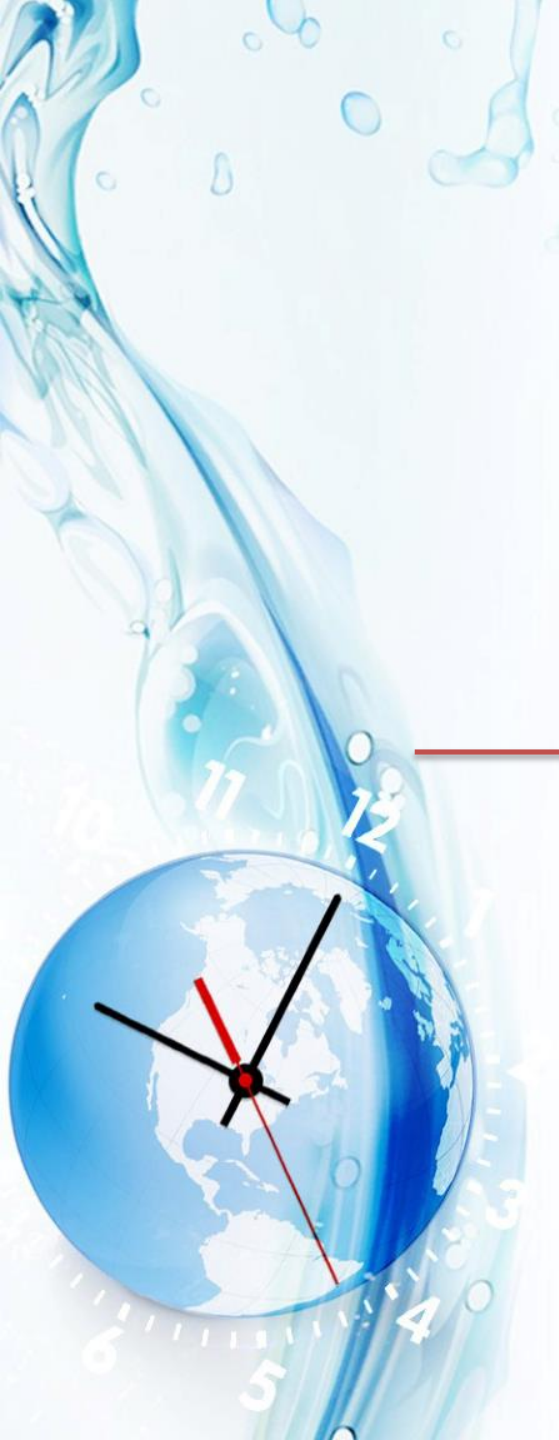
JS Arithmetic Operators

Operator	Name	Example	Result
+	Addition	$x + $y	Sum of $x and $y
-	Subtraction	$x - $y	Difference of $x and $y.
*	Multiplication	$x * $y	Product of $x and $y.
/	Division	$x / $y	Division of $x by $y
%	Modulus	$x % $y	Remainder of $x divided by $y

JS Arithmetic Operators

Example

```
var x = 10;

var y = 4;

console.log(x + y);      // 14

console.log(x - y);      // 6

console.log(x * y);      // 40

console.log(x / y);      // 2.5

console.log(x % y);      // 2
```



JS Assignment Operators

The assignment operators are used to assign values to variables.

JS Assignment Operators

Operator	Description	Example	Same as
=	Assign	x = y	x = y
+=	Add and assign	x += y	x = x + y
-=	Subtract and assign	x -= y	x = x - y
*=	Multiply and assign	x *= y	x = x * y
/=	Divide and assign	x /= y	x = x / y
%=	Divide and assign modulus	x %= y	x = x % y

JS Assignment Operators

Example

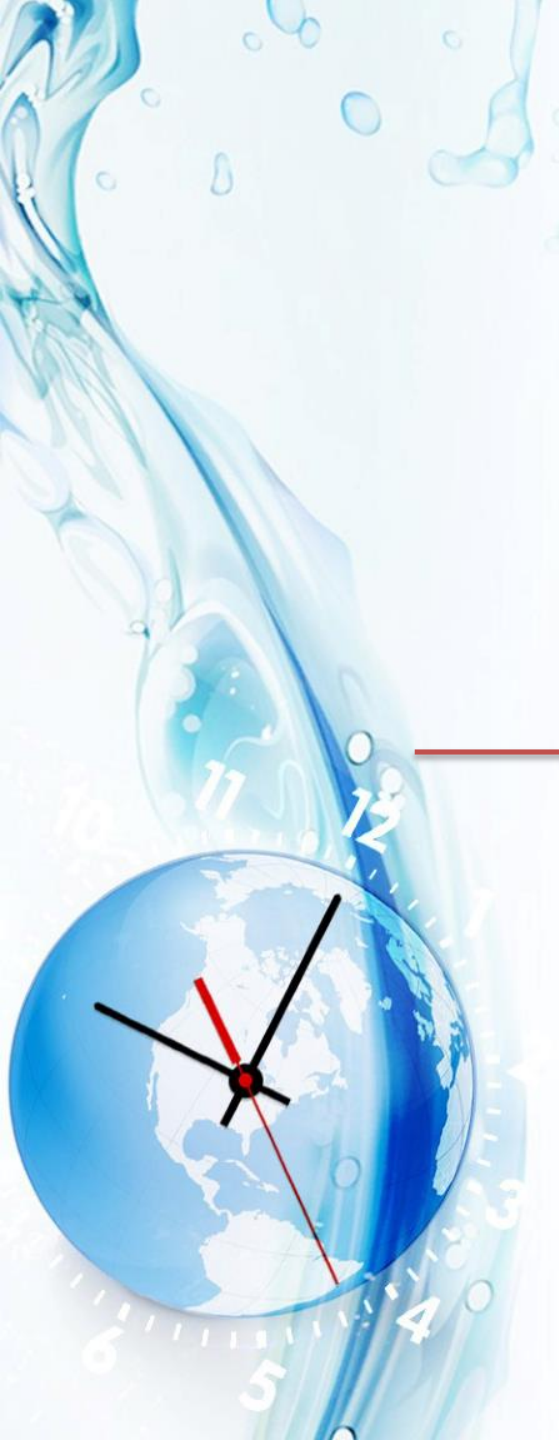
```
var x;

x = 10;                          console.log(x);     // 10

x = 20;      x += 30;            console.log(x);     // 50

x = 50;      x -= 20;            console.log(x);     // 30

x = 5;       x *= 25;            console.log(x);     // 125

x = 50;      x /= 10;            console.log(x);     // 5

x = 100;     x %= 15;            console.log(x);     // 10
```

Adding Strings and Numbers

- Adding two numbers ➔ Number
- Adding number and a string ➔ String

Example

```
x=5+5;          // 10 ➔ number
y="5"+5;        // 55 ➔ string
z="Hello"+5;    // Hello5 ➔ string
```



JS String Operators

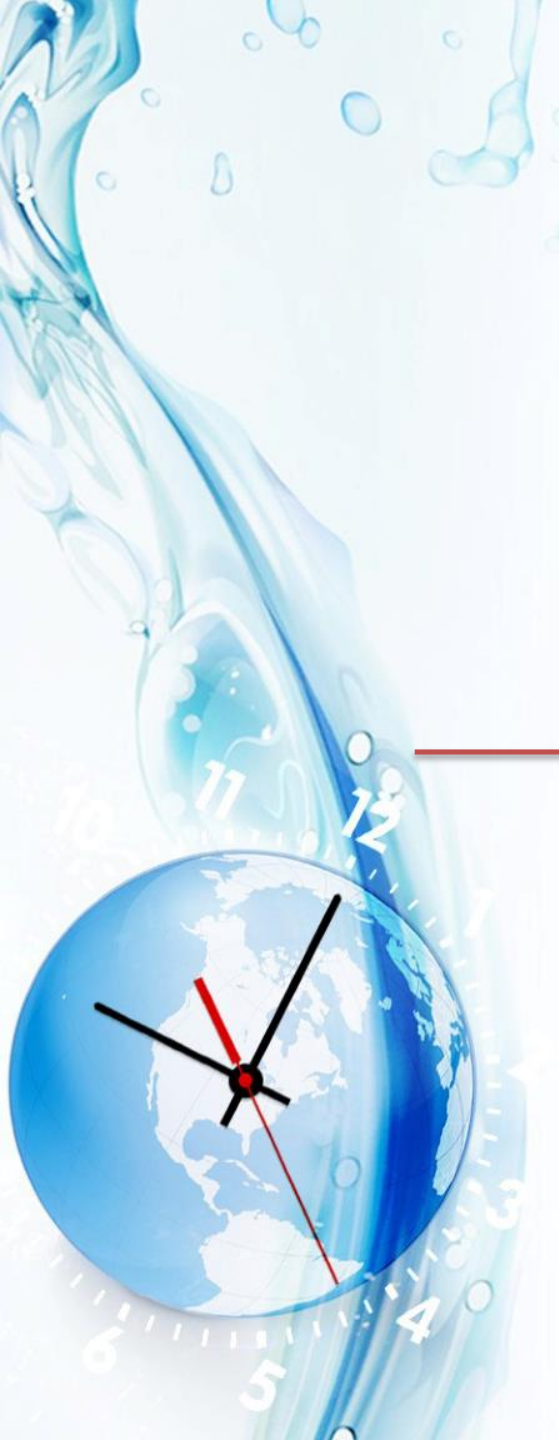
JS String Operators

Operator	Description	Example	Result
+	Concatenation	str1 + str2	Concatenation of str1 and str2
+=	Concatenation assignment	str1 += str2	Appends the str2 to the str1

JS String Operators

Example

```
var str1 = "Hello";

var str2 = " World!";

console.log(str1 + str2);        //  Hello World!

str1 += str2;

console.log(str1);               //  Hello World!
```



Js Incrementing Operators

The assignment operators are used to assign values to variables.

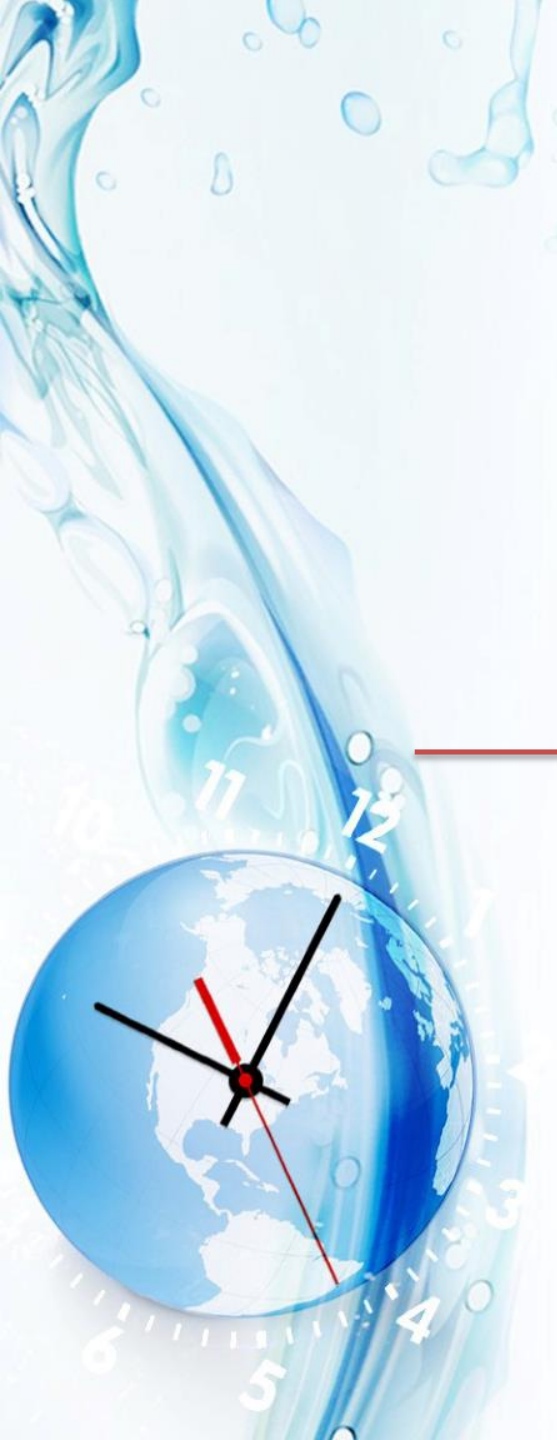
Js Incrementing Operators

Operator	Name	usage	Description
++x	Pre-increment	y=++x;	Increments x by one, then returns x ➔ x=x+1; y=x;
x++	Post-increment	y=x++;	Returns x, then increments x by one ➔ y=x; x=x+1;
--x	Pre-decrement	y=--x;	Decrements x by one, then returns x ➔ x=x-1; y=x;
x--	Post-decrement	y=x--;	Returns x, then decrements x by one ➔ y=x; x=x-1;

Js Incrementing Operators

Example

```
var x = 10;
console.log(++x);          // 11
console.log(x);            // 11

x = 10;
console.log(x++);          // 10
console.log(x);            // 11

x = 10;
console.log(--x);          // 9
console.log(x);            // 9

x = 10;
console.log(x--);          // 10
console.log(x);            // 9
```

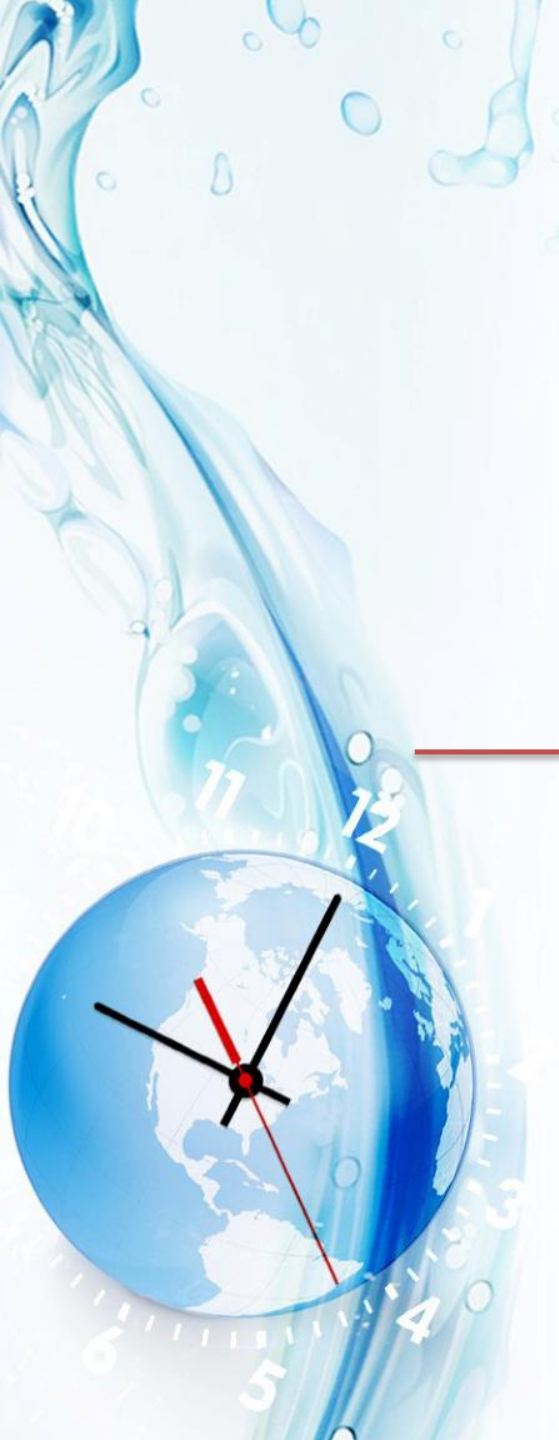


JS Logical Operators

The logical operators are typically used to combine conditional statements.

JS Logical Operators

Operator	Name	Example	Result
Assume a = 6 and b = 3			
&&	And	x && y	True if both x and y are true Ex: (a>3 && b<8) ➔ True (a>8 && b<8) ➔ False
||	Or	x || y	True if either x or y is true Ex: (a>3 || b<1) ➔ True (a>8 || b<1) ➔ False
!	Not	!x	True if x is not true !(a<8) ➔ False



JS Comparison Operators

The comparison operators are used to compare two values in a Boolean fashion.

JS Comparison Operators

Operator	Name	Example	Output
==	Equal	x == y	True if x is equal to y
===	Identical	x === y	True if x is equal to y, and they are of the same type
!=	Not equal	x != y	True if x is not equal to y
!==	Not identical	x !== y	True if x is not equal to y, or they are not of the same type
<	Less than	x < y	True if x is less than y
>	Greater than	x > y	True if x is greater than y
>=	Greater than or equal to	x >= y	True if x is greater than or equal to y
<=	Less than or equal to	x <= y	True if x is less than or equal to y

JS Comparison Operators

Example

```
var x = 25;
var y = 35;
var z = "25";

console.log(x == z);          // true
console.log(x === z);         // false
console.log(x != y);          // true
console.log(x !== z);         // true
console.log(x < y);           // true
console.log(x > y);           // false
console.log(x <= y);          // true
console.log(x >= y);          // false
```

Conditional Operator

- assigns a value to a variable based on some condition.

Syntax

```
variableName = (condition) ? value1 : value2
```

Example

```
x = (y>20) ? "ok" : "not ok"
```

Global Method: String()

- A global method used to convert numbers to strings.

Syntax

```
String(variableName)
```

Example

```
var x = 10;
x = String(x);
console.log(typeof x);          // String
```

Global Method: Number()

- A global method used to convert strings to numbers.

Syntax

```
Number(variableName)
```

Example

```
var x = "10";
x = Number(x);
console.log(typeof x);          // Number
```

Global Method: isNaN()

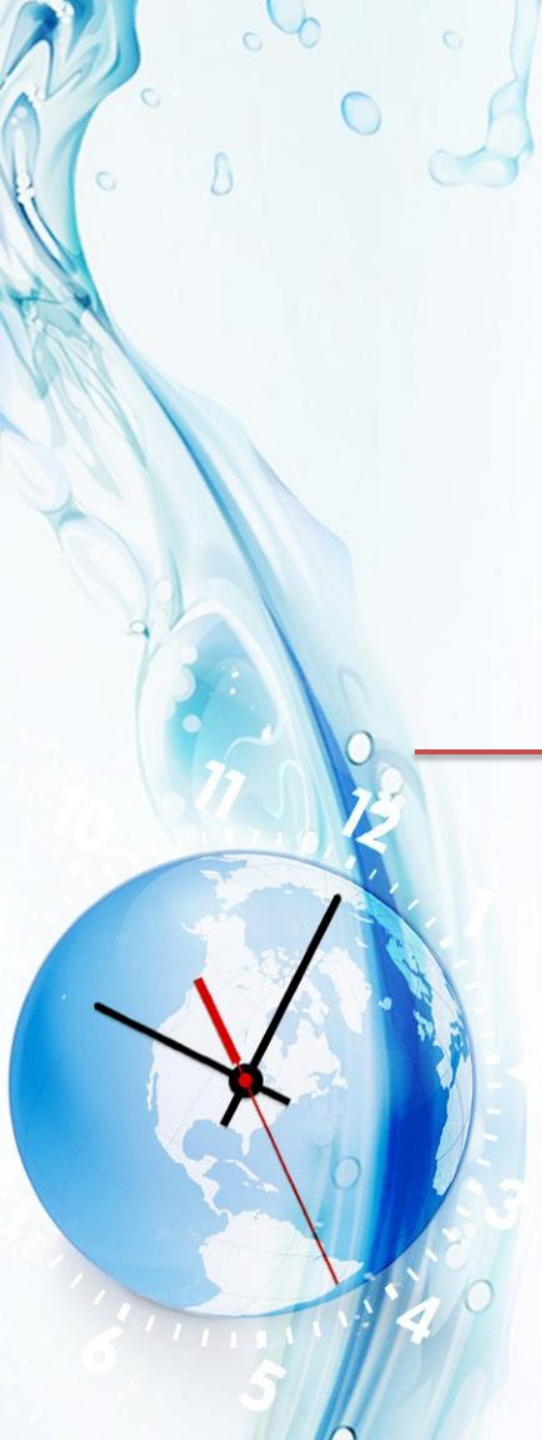
- A global method converts a variable data type to a number and returns true if the value is equal NaN.

Syntax

```
isNaN(variableName)
```

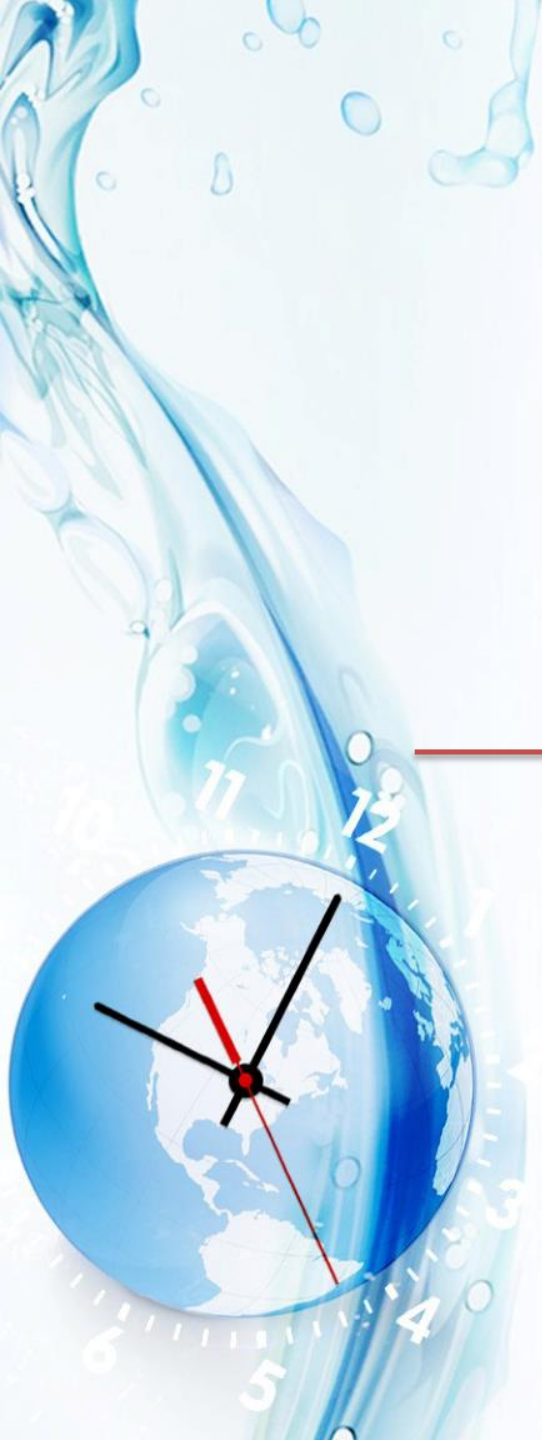
Example

```
isNaN('NTI')        // true
isNaN('123')        // false
```



JS Conditional Statements

Conditional statements are used to decide the flow of execution based on different conditions. If the condition is true, an action is performed and if the condition is false, another action is performed.



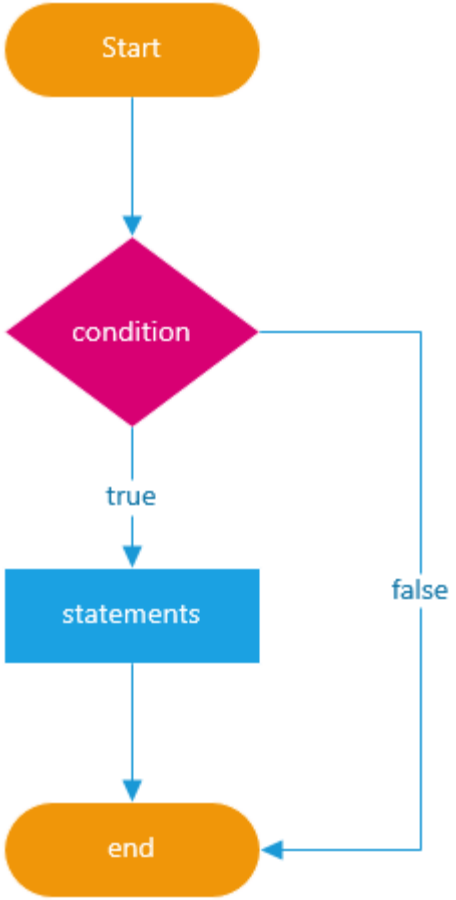
If Statement

The if statement is used to execute a block of code only if a specified condition is true.

If Statement

Syntax

```
if (condition) {
        // Code to be executed
    }
```



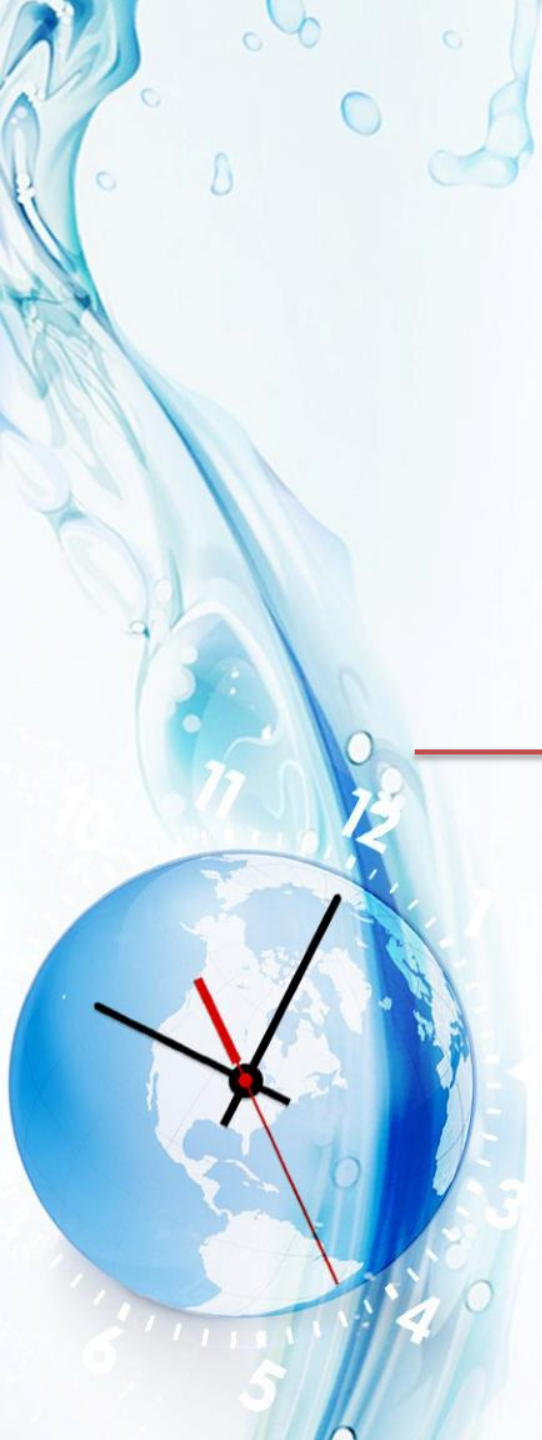
If Statement

Syntax

```
if (condition)  {
        // Code to be executed
    }
```

Example

```
if (salary<2000)  {
        var tax = 0.1;    // tax will be 0.1 if the salary < 2000
    }
```



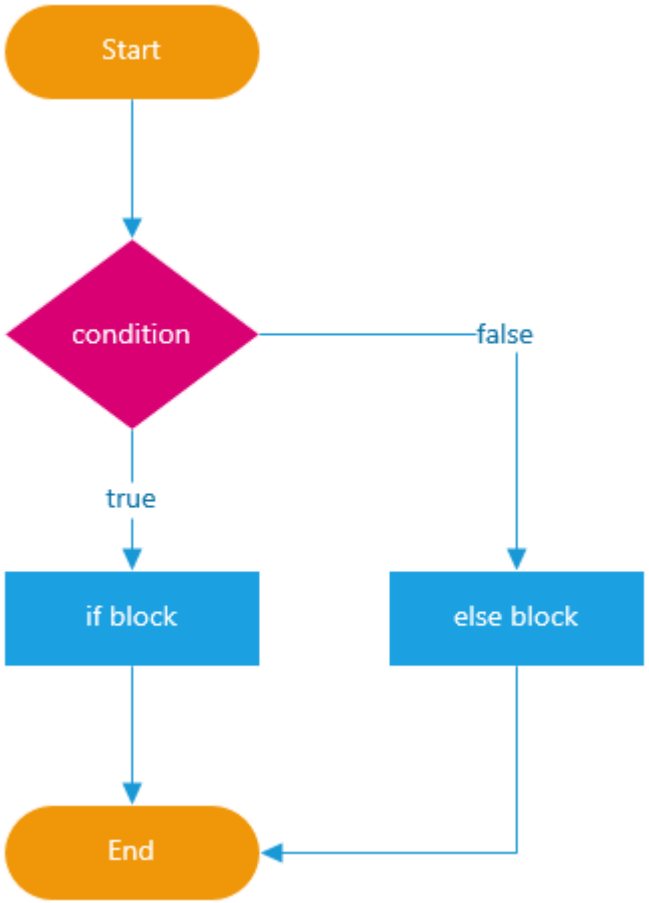
If...else Statement

When the condition inside the if statement returns "false", the code inside the else statement is executed.

If...else Statement

Syntax

```
if (condition) {
        // Code to be executed
}  else  {
        // Code to be executed
}
```



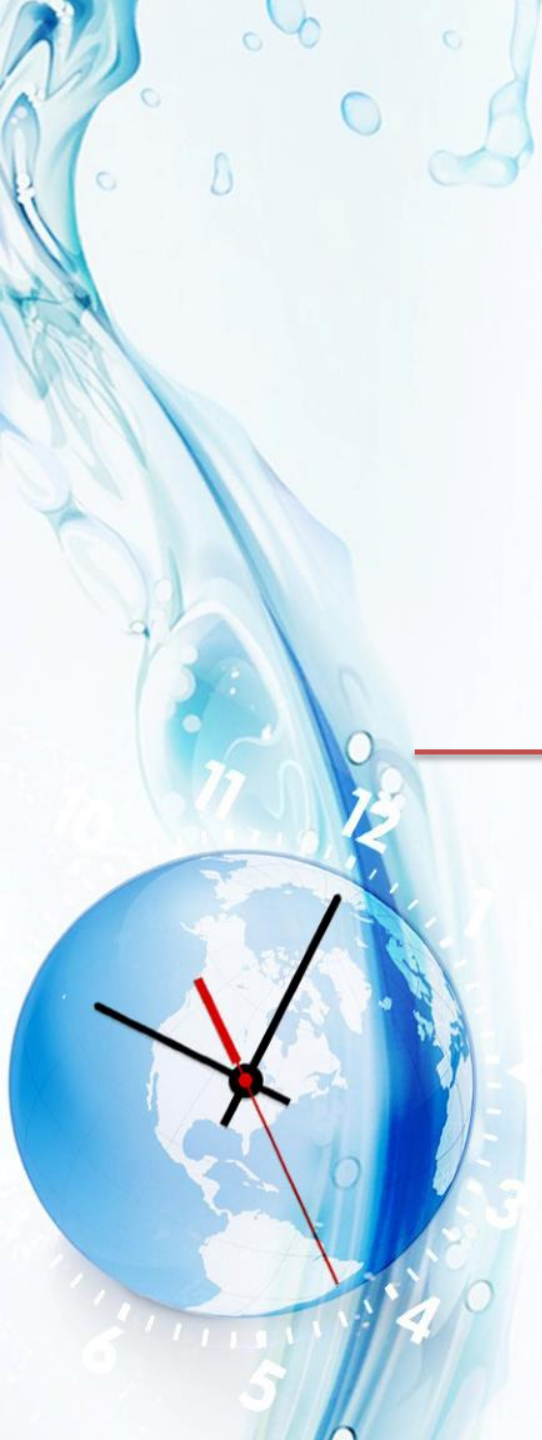
If...else Statement

Syntax

```
if (condition) {
        code to be executed if condition is true
}  else  {
        code to be executed if condition is not true
}
```

Example

```
if (salary<2000)  {
        var tax = 0.1;        // tax will be 0.1 if the salary < 2000
 } else  {
        var tax = 0.2;        // tax will be 0.2 if the salary not < 2000
}
```



If...else if...else Statement

If….Else If…. Statement is used to check more than one condition and execute a different set of codes.

If...else if...else Statement

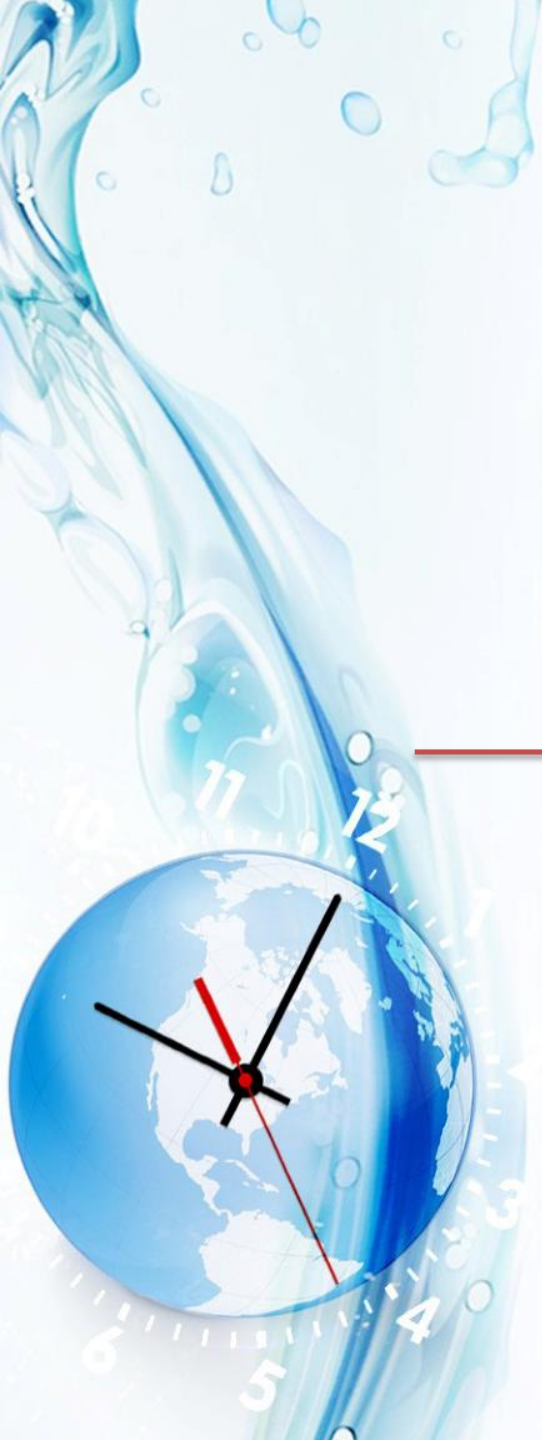
Syntax

```
if (condition1) {
    code to be executed if condition 1 is true
} else if (condition2) {
    code to be executed if condition1 is false and condition2 is true
} else {
    code to be executed if both condition1 and condition2 are false
}
```

If...else if...else Statement

Example

```
if (salary<2000) {
        var tax = 0.1;          // tax will be 0.1 if the salary < 2000
}  else if (salary>5000)  {
        var tax = 0.3;          // tax will be 0.3 if the salary < 5000
}   else  {
        var tax = 0.2;
           // tax will be 0.2 if the salary not <2000 and not >5000
}
```



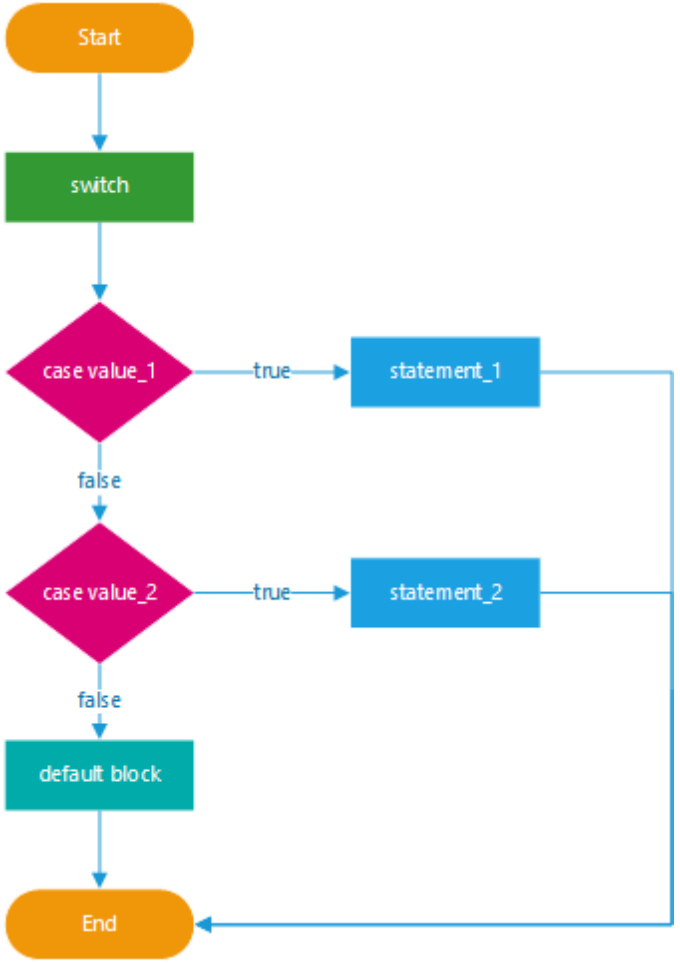
Switch Statement

The switch statement is an alternative to the if...else statement, it tests a variable against a series of values until it finds a match, and then executes the block of code corresponding to that match.

Switch Statement

Syntax

```
switch(variable)
    {
        case value:
            // execute code block 1
            break;
        Case value:
            // execute code block 2
            break;
        default:
            // execute code block 3
            break;
    }
```



Switch Statement

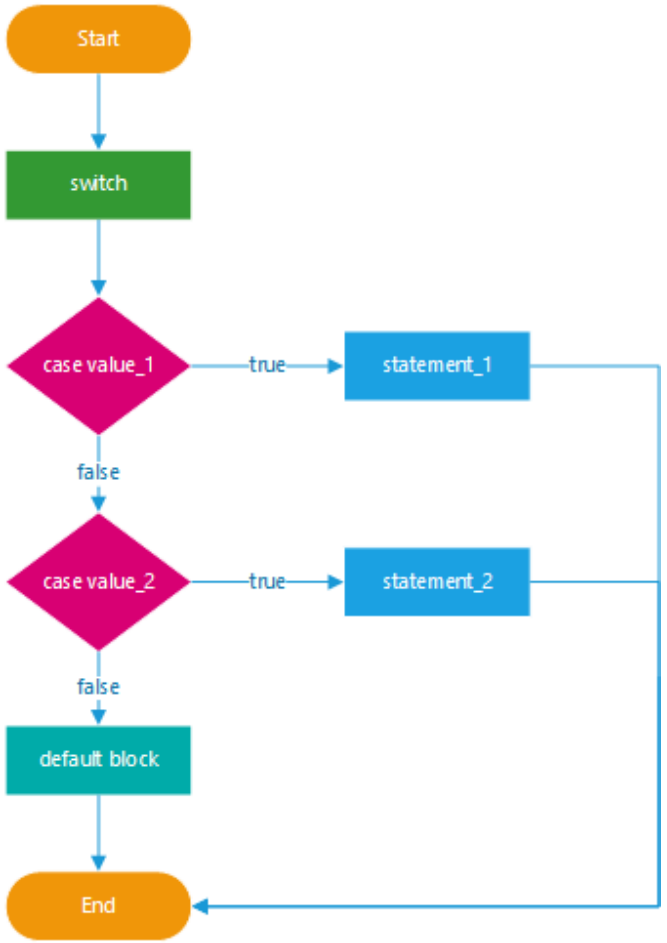
"break" is used to prevent the code from running into the next case automatically.

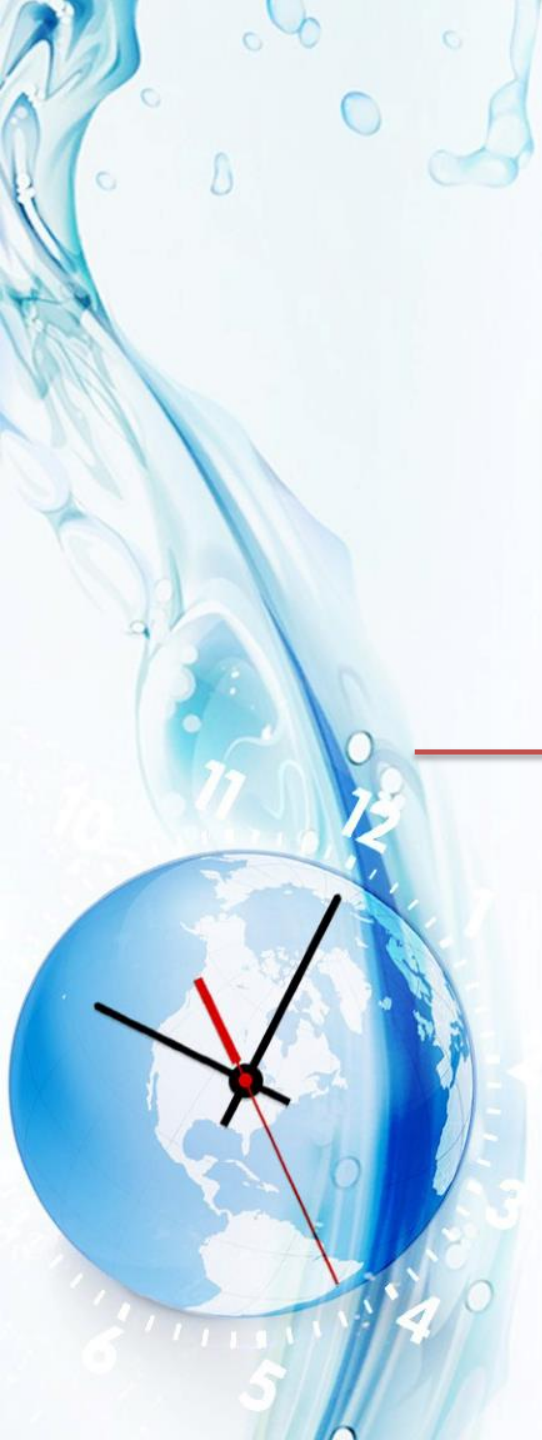
Example

```
switch(x)
    {
        case 10:
            y=30;
            break;
        Case 20:
            y=50;
            break;
        default:
            y=100;
            break;
    }
```

Example: Multiple Cases Sharing Same Action

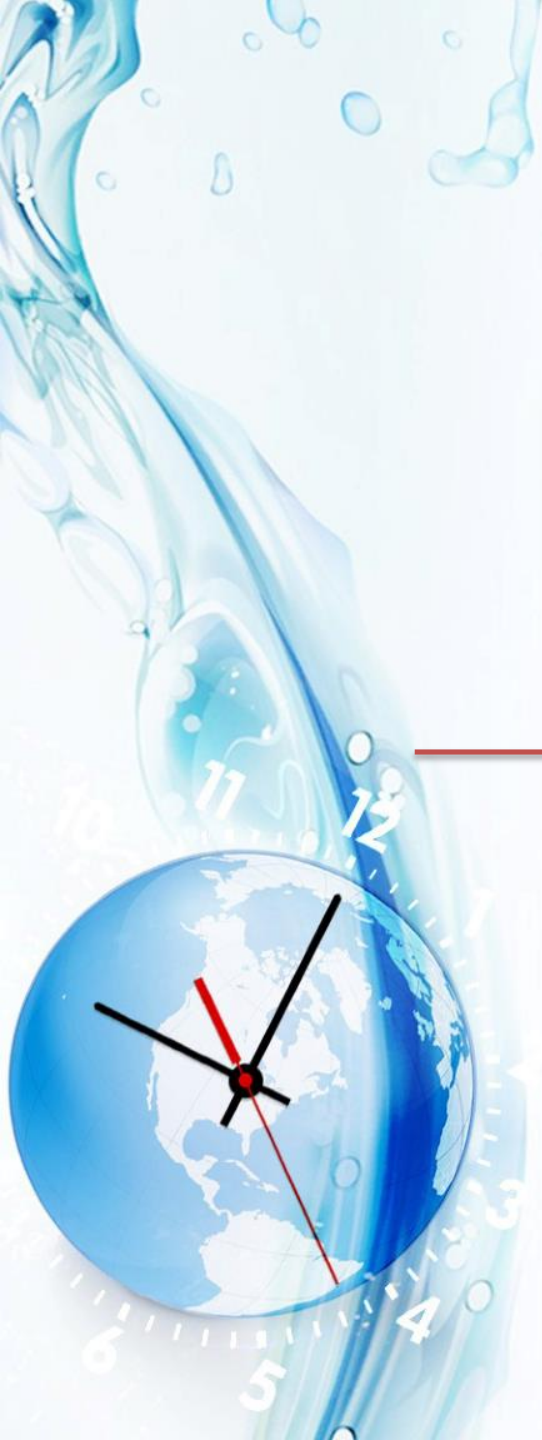
```
switch(x)
    {
        case 10:
        case 20:
        case 25:
            y=30;
            break;
        case 30:
            y=50;
            break;
        default:
            y=100;
            break;
    }
```





JS Loops

Loops are used to execute the same block of code again and again, as long as a certain condition is met.



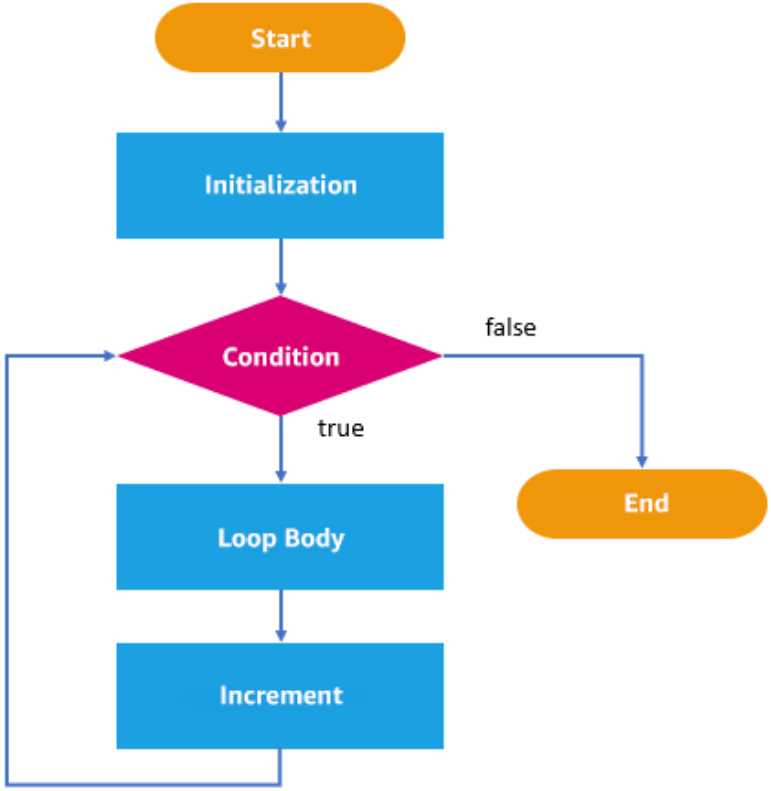
For Loop

The “for” loop is the most common type of JavaScript loops because it runs "for" a specific number of times.

For Loop

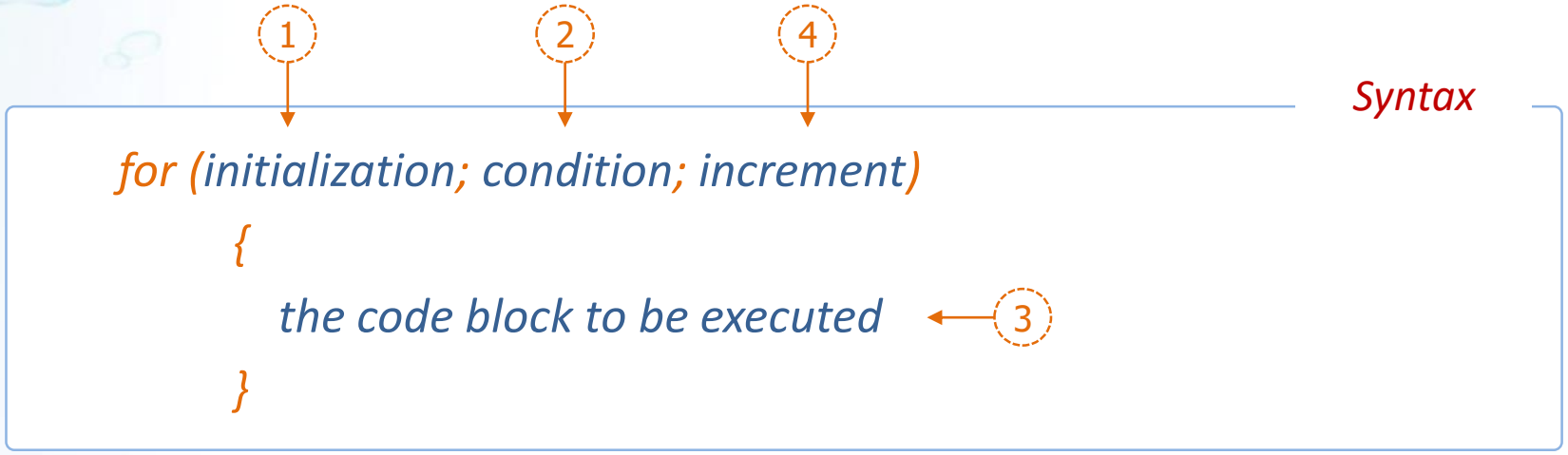
Syntax

```
for (initialization; condition; increment)
        {
          code block
        }
```



For Loop

Syntax



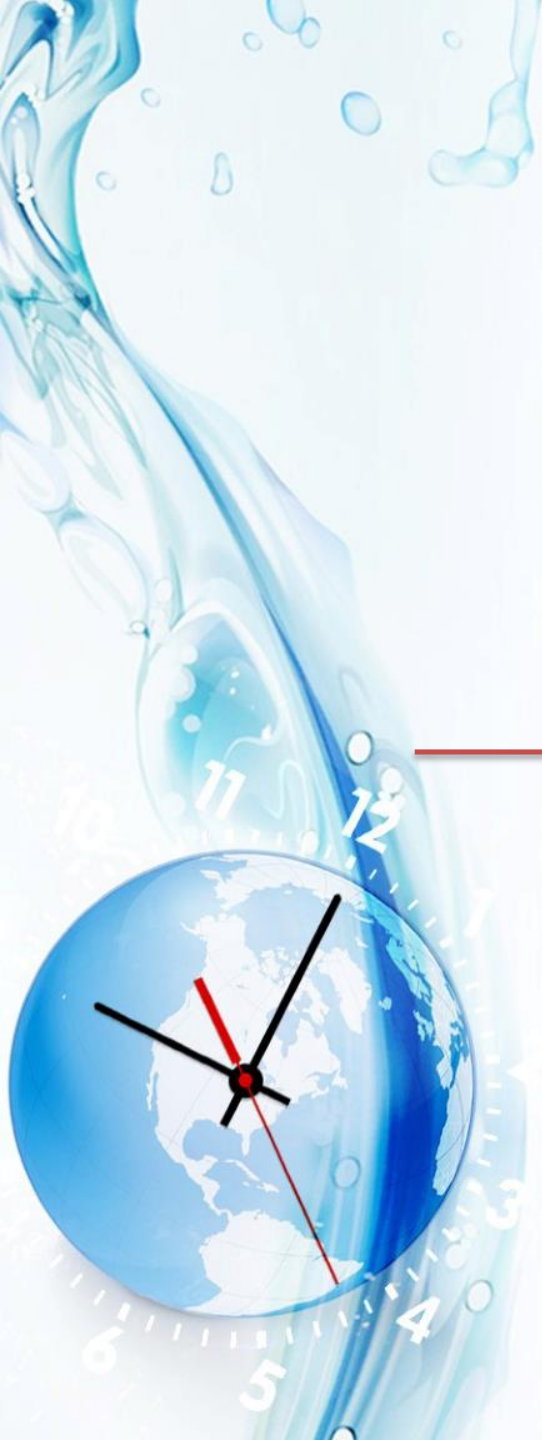
- *Initialization:* used to initialize the counter variables, and evaluated once before the first execution of the body of the loop.
- *Condition:* it is evaluated at the beginning of each iteration. If it evaluates to true, the loop statements execute.
- *Increment:* updates the loop counter with a new value each time the loop runs.

For Loop

Example

```
for (var  i = 0; i < 10 ; i++)
        {
           document.write(i + "<br>");
        }
```

1. Declares variable i and assign value 0 to it.
2. Defines the condition for the loop (variable i must be less than 10).
3. Executes the block code is the condition is true.
4. Increment variable i with 1. (i++ same as i=i+1).



For..in.. Loop

The for-in loop is a special type of a loop that iterates over the properties of an object, or the elements of an array.

For..in.. Loop

Array Syntax

```
for (var variableName in array) {
        // Code to be executed
}
```

Array Example

```
var arr = new Array("a", "b", "c");
arr[5] = "s";
for (var i in arr) {
   document.write(i+"<br>");
}
```

0
1
2
5

For..in.. Loop

Object Syntax

```
for (var variableName in object) {
  // Code to be executed
}
```

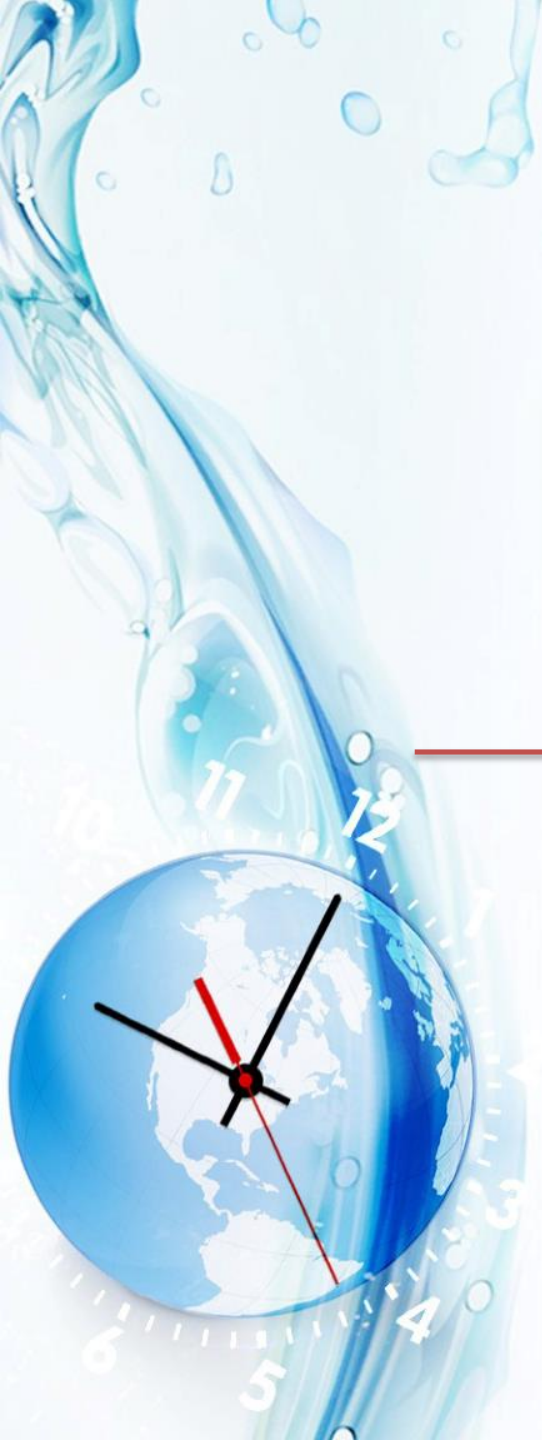
Object Example

```
var person={ firstName : 'Ahmed' , lastName : 'Ali' , age : 30 };

for (var prop in person) {

   document.write("<p>" + prop + " = " + person[prop] + "</p>");

}
```

firstName = Ahmed

lastName = Ali

age = 30



For..of.. Loop

ES6 introduces a new for-of loop which allows to iterates over the values of an array.

For..of.. Loop

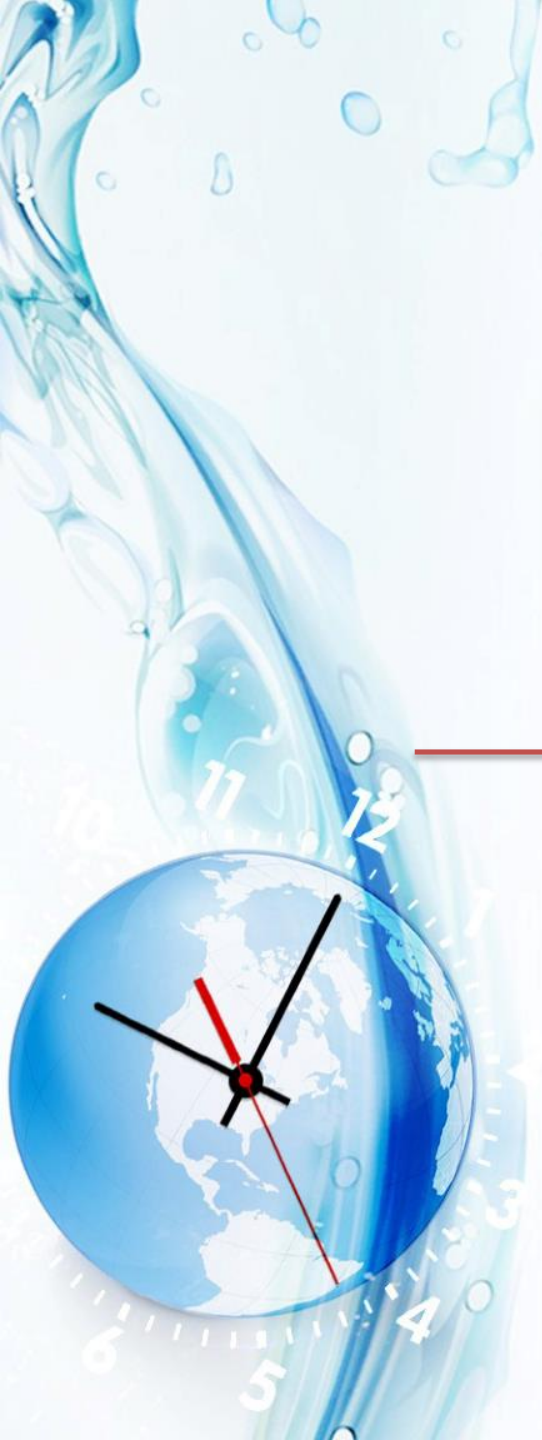
Syntax

```
for (var variableName of array) {
        // Code to be executed
}
```

Example

```
var arr=new Array("a", "b", "c");
arr[5]="s";
for (var i of arr) {
   document.write(i + "<br>");
}
```

```
a
b
c
undefined
undefined
s
```



While Loop

This is the simplest looping statement provided by JavaScript.

The while loop loops through a block of code as long as the specified condition evaluates to true. As soon as the condition fails, the loop is stopped.

While Loop

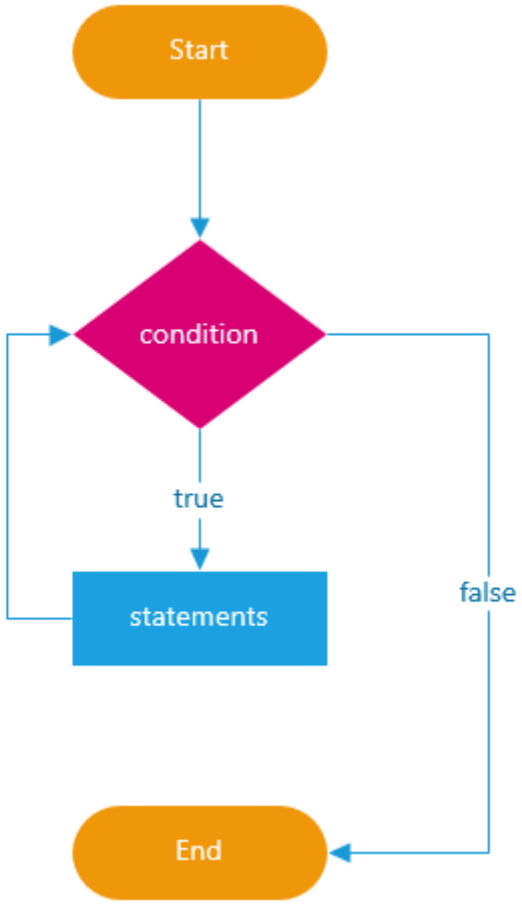
Syntax

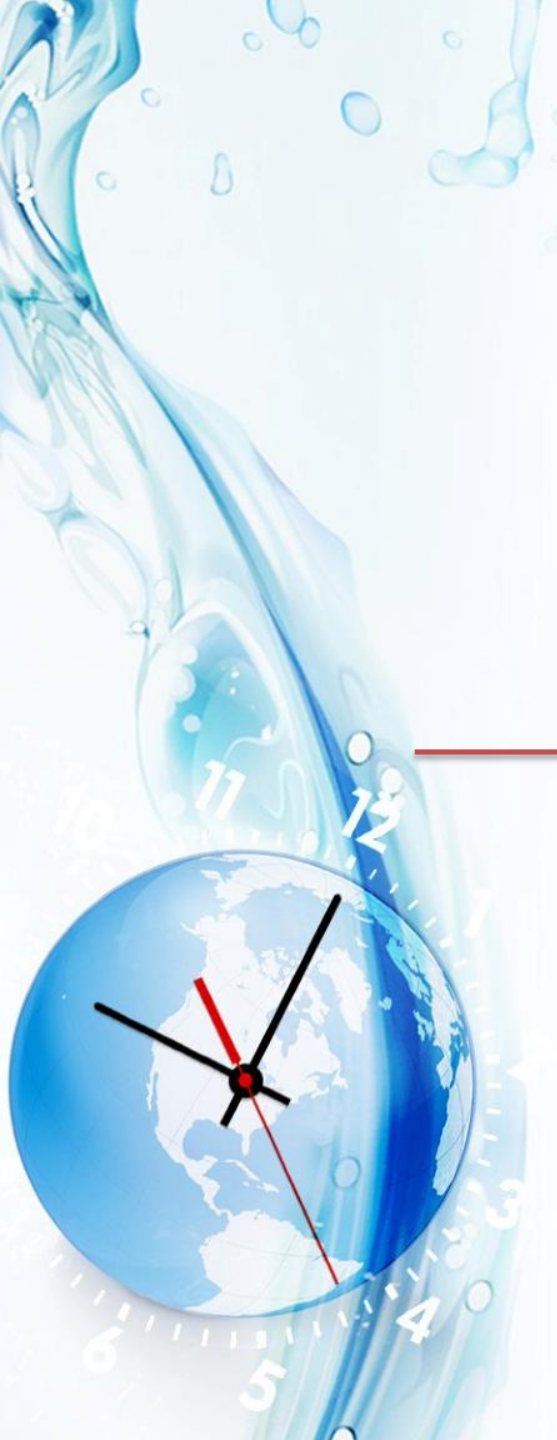
```
while (condition)
    {
    // code to be executed
    }
```

Example

```
var i = 0;
while (i < 3)
    {
    document.write(i);
    i = i + 1;
}
```

012





Do/While Loop

Do/While Loop

- The do/while loop is a variant of the while loop.
- The code block is executed once, before checking the condition.
- The loop block will be executed again and again as long as the condition is true.

Syntax

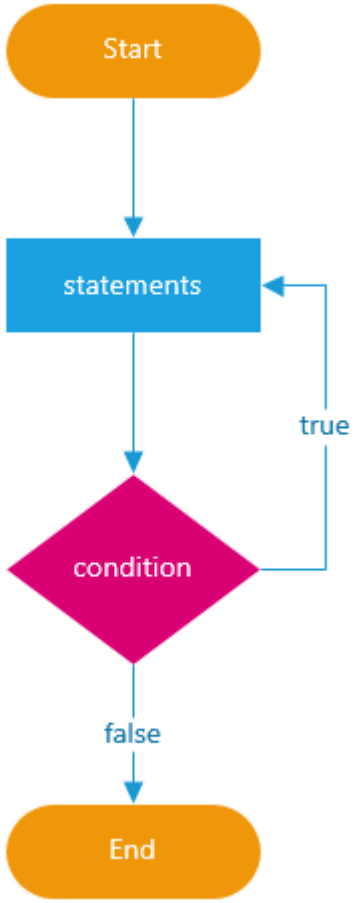
```
do
    {
        // code to be executed
    }
while (condition);
```

Do/While Loop

Example

```
<script>
        var i = 0;
        do
            {
                    document.write(i);
                    i++;
            }
        while (i < 3);
</script>
```

012



Quiz

- *Create an array containing three or more values.*
- *Print the values of the array on a new html document using "for loop".*
- *Print the values of the array on another new html document using "while loop".*

Answer Using “while” loop

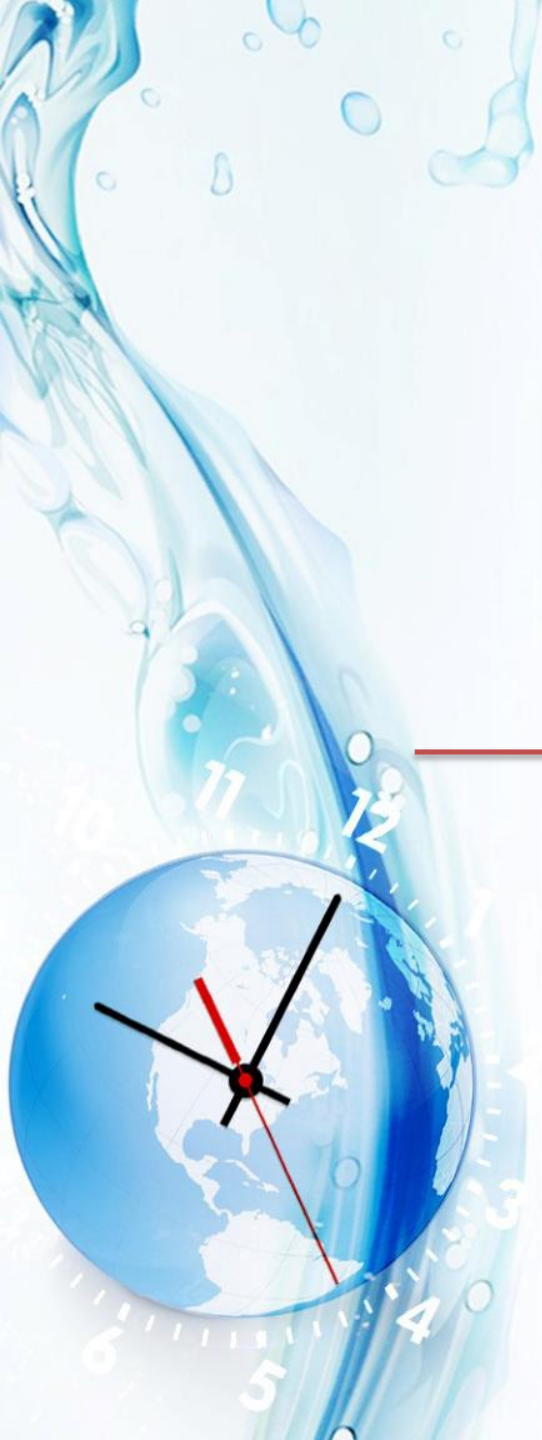
Answer

```
<html><body>
<script>
        colors=["red" , "green" , "blue"];
                var i = 0;
                while (colors[i])
                        {
                        document.write(colors[i] + "<br>");
                        i++;
                        }
</script>
</body></html>
```

Answer Using “for” loop

Answer

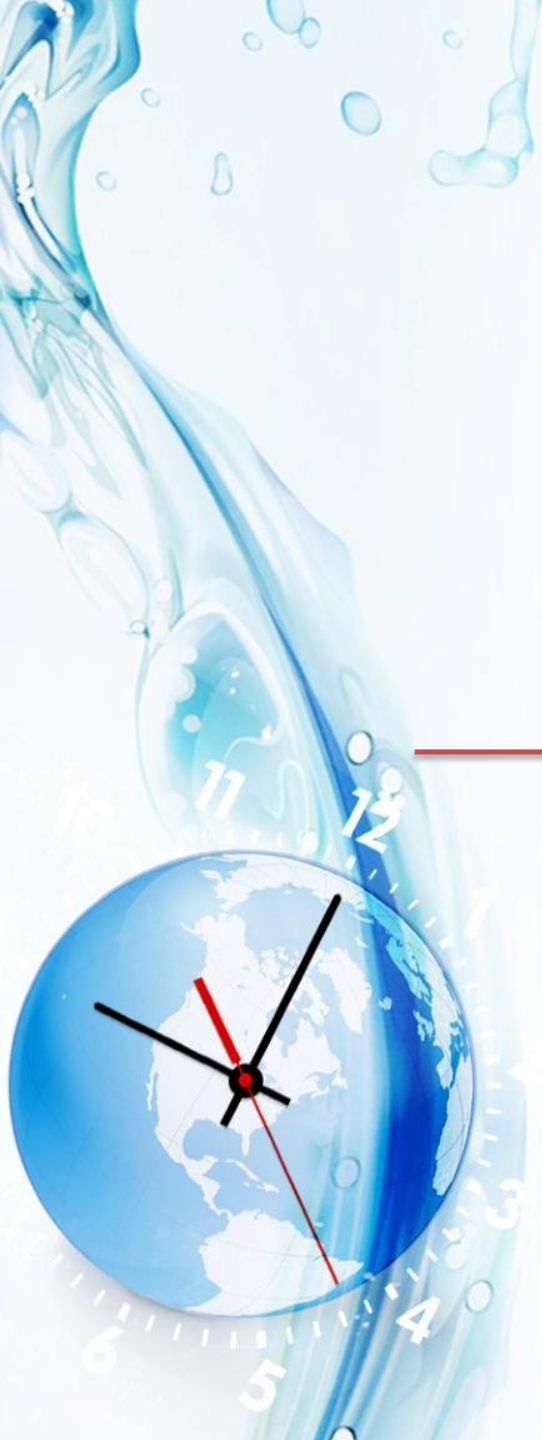
```
<html><body>
<script>
        colors=["red" , "green" , "blue"];
                var i = 0;
                for (var i = 0 ; colors[i] ; i++)
                        {
                        document.write(colors[i] + "<br>");
                        }
</script>
</body></html>
```



JS Loop Control

JavaScript provides "break" and "continue" statements to provide loops full control.

There may be a situation when you need to come out of a loop without reaching its bottom or to skip a part of the code block and start the next iteration of the loop.



JS Break Statement

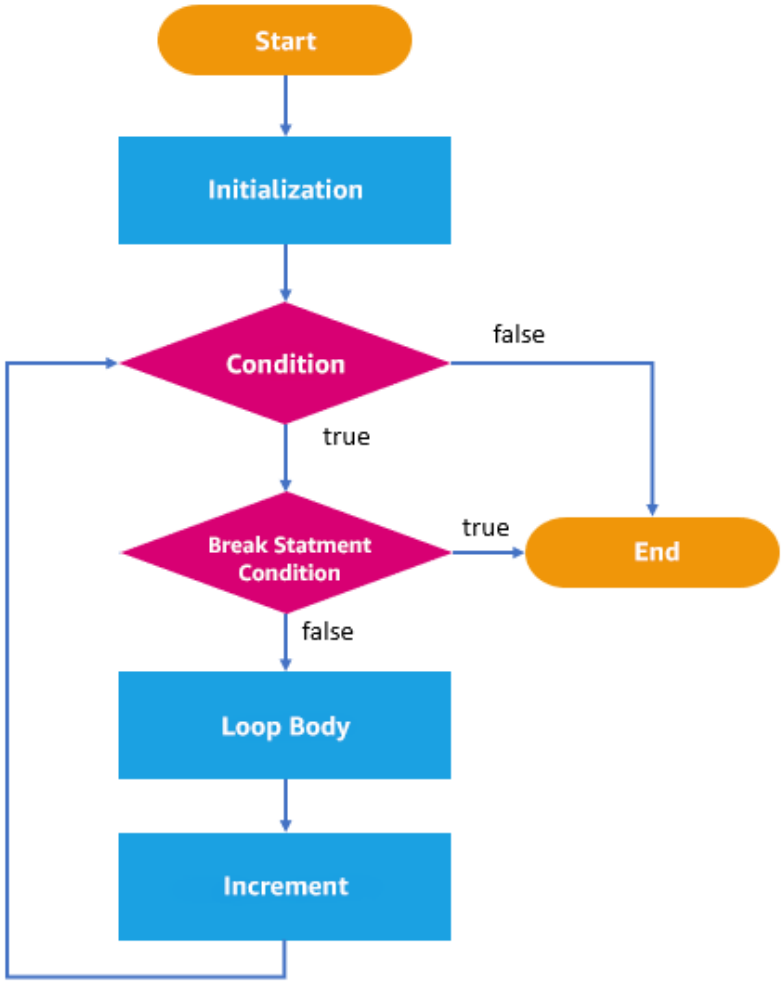
The break statement, which was briefly introduced with the switch statement, is used to exit a loop early, breaking out of the enclosing curly braces.

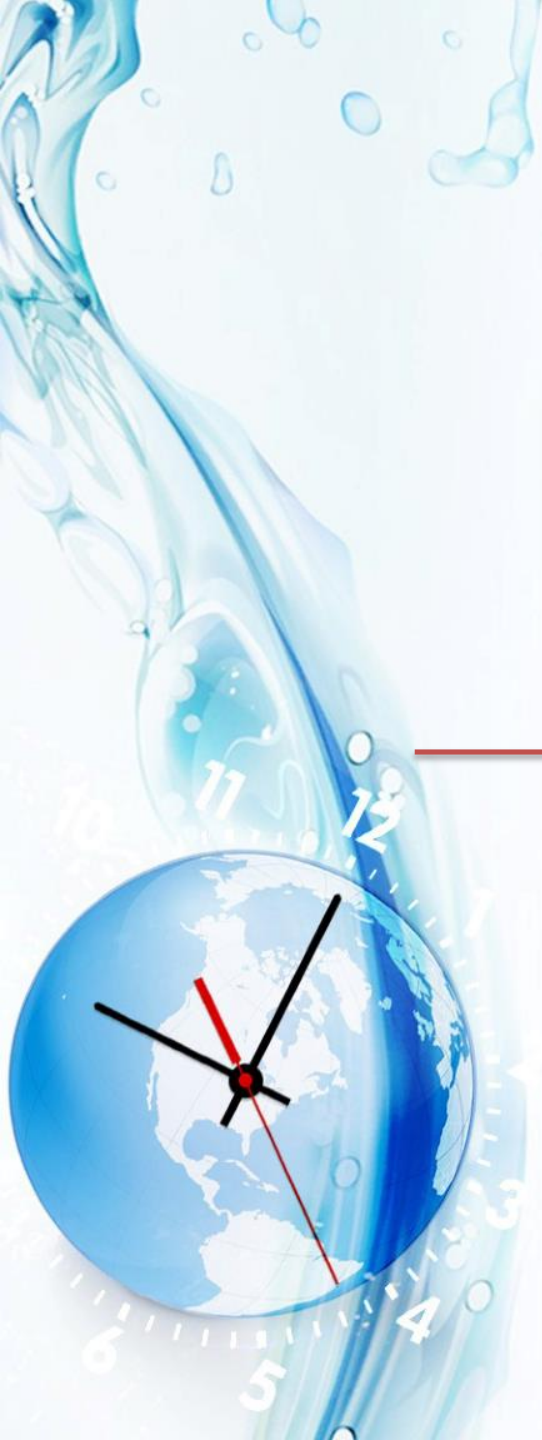
JS Break Statement

Example

```
for (i=0 ; I < 10 ; i++)
 {
      if (i==3)
              {
              break;
              }
     console.log(i);
 }
```

012





JS Continue Statement

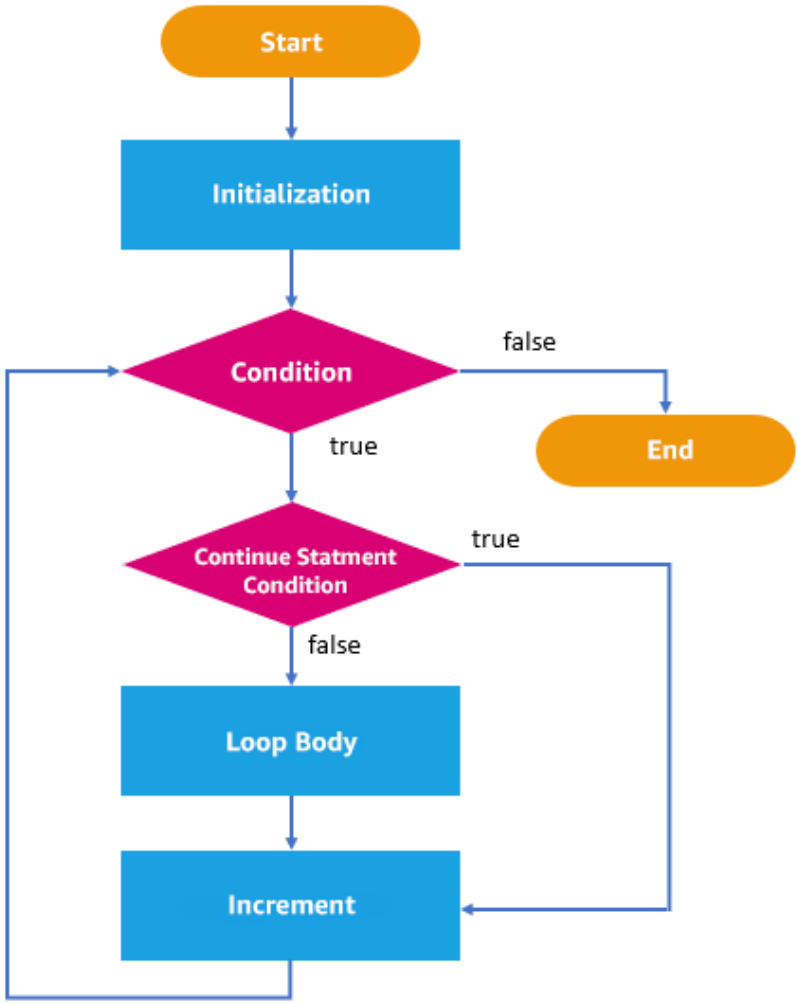
The continue statement tells the interpreter to immediately start the next iteration of the loop and skip the remaining code block.

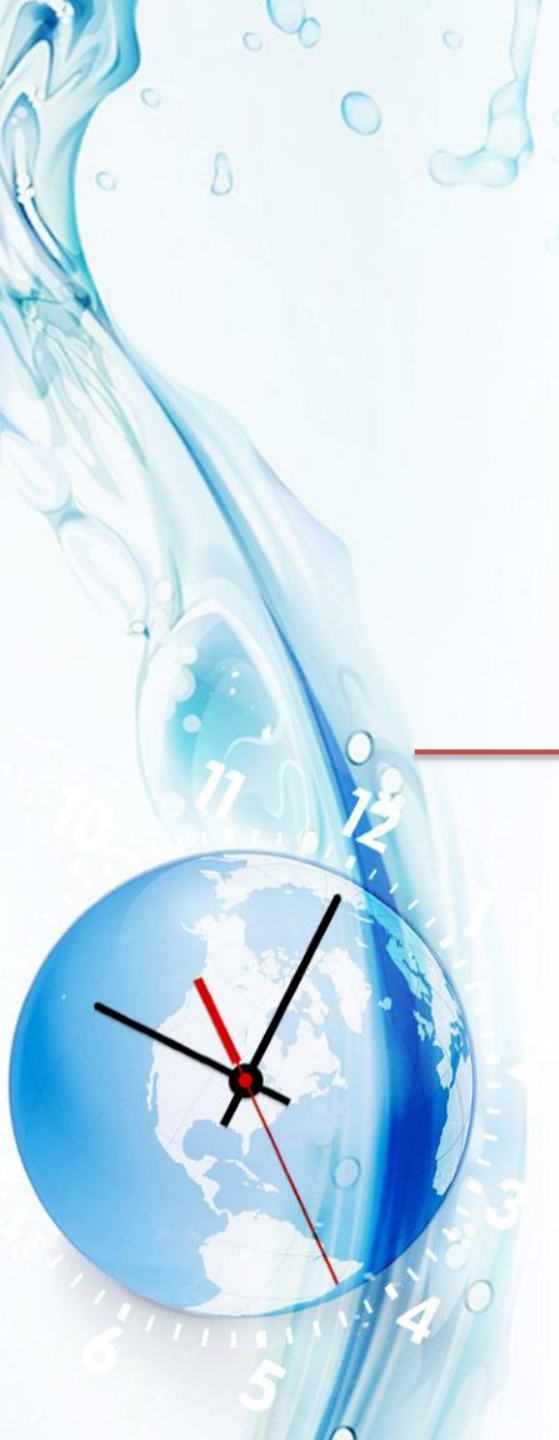
JS Continue Statement

Example

```
for (i=0;i<10;i++)
 {
      if (i==3)
              {
              continue;
              }
     document.write(i);
 }
```

012456789





JS Objects

JavaScript Objects

- *JavaScript is an object-based language; i.e. everything is an object in JavaScript.*
- *JS object is a non-primitive (composite) data type. It is like any other variable, the only difference is that an object holds multiple values in terms of properties and methods.*
- *JavaScript as an OOP it provides four basic capabilities to developers: Encapsulation, Aggregation, Inheritance, and Polymorphism.*

- ✓ *Encapsulation*
  - » *The capability to store related information, whether properties or methods, together in an object.*
- ✓ *Aggregation*
  - » *The capability to store one object inside another object.*
- ✓ *Inheritance*
  - » *The capability of a class to rely upon another class (or number of classes) for some of its properties and methods.*
- ✓ *Polymorphism*
  - » *The capability to write one function or method that works in a variety of different ways.*

JS Object Properties

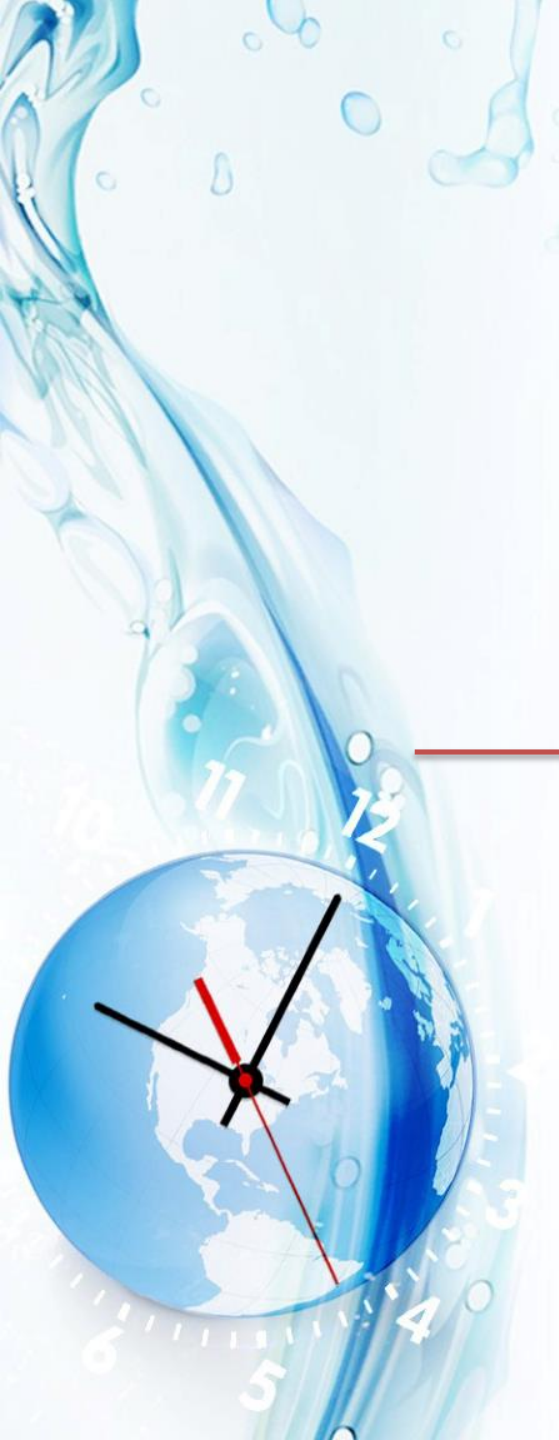
- *Object properties can be any of the primitive data types, or any of the abstract data types, such as another object.*
- *Each property is composed of a key (name) and a value.*
- *The property's value can be placed as a function, in this case we call the property as a "method".*

JS Object Methods

- *Methods are the functions that let the object do something or let something be done to it.*
- *The difference between a function and a method is that a function is a standalone unit of statements and a method is attached to an object and can be referenced by the "this" keyword.*

In JavaScript there are many types of objects:

- *User-Defined Objects*
  - ✓ *Custom objects created by the JavaScript programmer.*
- *Native Objects*
  - ✓ *Predefined in JavaScript language (string, number, Date).*
- *Hosts Objects*
  - ✓ *Objects used inside the environment itself like window and navigator.*
- *Document Objects*
  - ✓ *These are part of the object document model (DOM), in which each element on the page is defined as an object.*



User-Defined Objects

Creating Objects in JavaScript

- *JavaScript is template based not class based. i.e., "class" is not created to get the "object", but "objects" are directly created.*
- *There are three ways to create objects:*
  - ✓ *By object literal.*
  - ✓ *By creating instance of Object directly (using new keyword)*
  - ✓ *By using an object constructor.*

Creating Object by object literal

Syntax

```
var objectName = { property : value , property : value };
```

Example

```
<script>
    var emp = { id:102 , name : "Ahmed", salary : 4000 };
    console.log(emp.id + " " + emp.name + " " + emp.salary);
</script>
```

```
102  Ahmed  4000
```

Creating Object by creating instance of Object

Syntax

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

Example

```
var emp = new Object();

emp.id  = 101;

emp.name = "Ahmed";

emp.salary = 4000;

console.log(emp.id + " " + emp.name + " " + emp.salary);
```

```
101  Ahmed  4000
```

Creating Object by using an Object constructor

Example

```
<script>
    function emp(a , b , c){
        this.id=a;
        this.name=b;
        this.salary=c;
    }

    var x=new emp(101,"Ahmed",4000);
    console.log(x.id + " " + x.name + " " + x.salary);
</script>
```

101 Ahmed 4000

Clone Objects in JavaScript

Objects are reference types. So when the equal sign (=) is used, it copies the pointer to the memory space it occupies. Reference types don't hold values, they are a pointer to the value in memory.

Example

```
var emp={ id:102 , name : "Ahmed", salary : 4000 };
var user = Object.assign({}, emp);        // clone aoject emp to user
emp.id=500;                    // change the id property in emp object
console.log(emp);
console.log(user);
```

```
{id: 500, name: "Ahmed", salary: 4000}
{id: 102, name: "Ahmed", salary: 4000}
```

Defining method in JavaScript Object

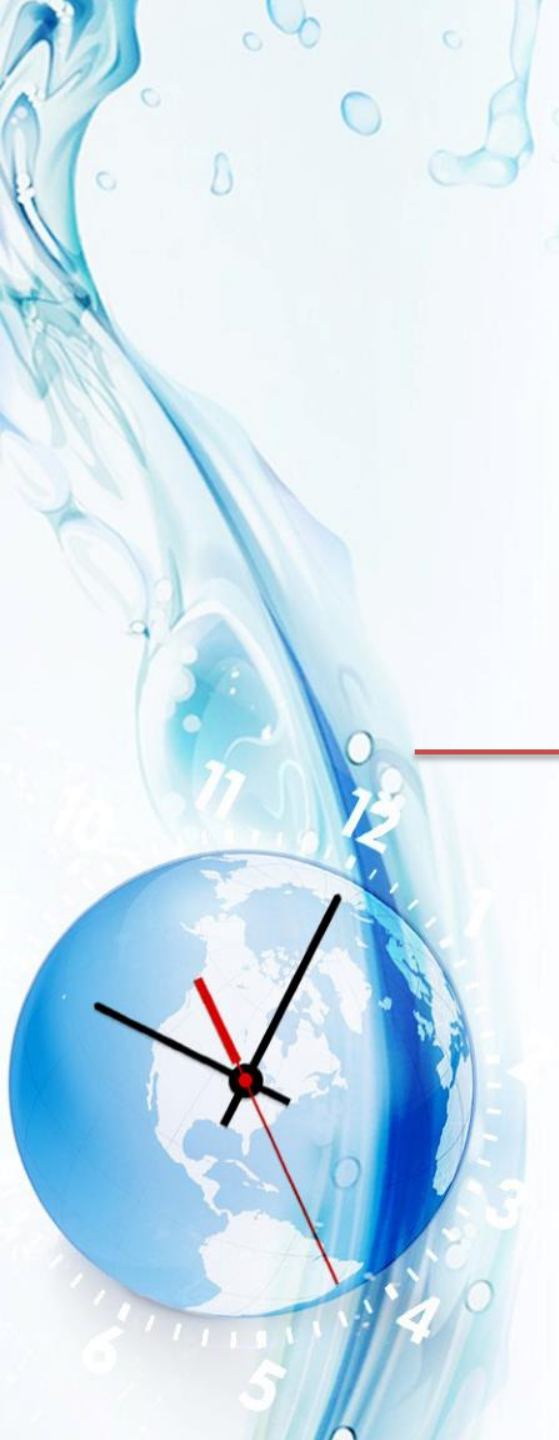
Method can be defines in JavaScript object. But before defining method, we need to add a property in the constructor function with same name as the method.

Defining method in JavaScript Object

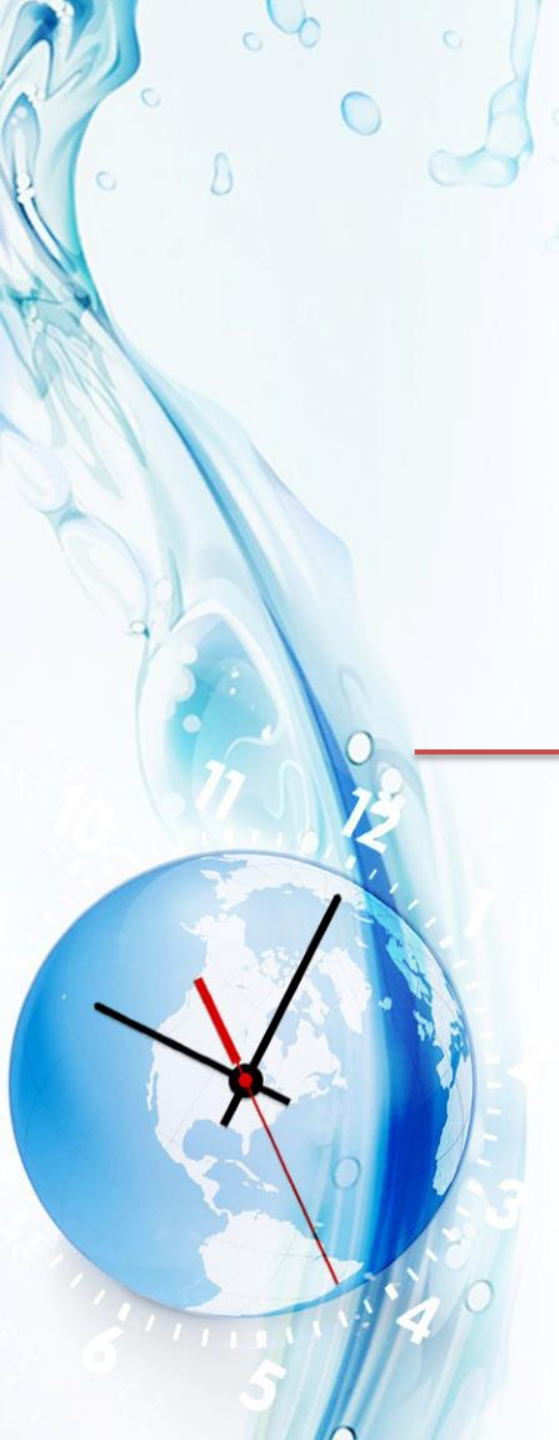
Example

```
function emp(a,b,c){
        this.id=a;
        this.name=b;
        this.salary=c;
        this.changeSalary=changeSalary;
              function changeSalary(s){
              this.salary=s;
              }
}
var x=new emp(101,"Ahmed",4000);
x.changeSalary(5000);
console.log( x.id + " " + x.name + " " + x.salary);
```

```
101  Ahmed  5000
```



JavaScript Native Objects



String Properties and Methods

Length Property

- Returns the length (number of characters) of a string.

Syntax

```
variableName.length;
```

Example

```
x="Hello";

console.log (x.length);
```

```
5
```

toUpperCase()

- Converts a string to uppercase letters.

Syntax

```
variableName.toUpperCase();
```

Example

```
var x= "Hello World!";

var y= x.toUpperCase();

Console.log (y);
```

HELLO WORLD!

toLowerCase()

- Converts a string to lowercase letters.

Syntax

```
variableName.toLowerCase();
```

Example

```
var x= "Hello World!";

var y= x.toLowerCase();

console.log (y);
```

hello world!

concat()

- Joins two or more strings together.

Syntax

```
variableName.concat(str1 , str2);
```

Example

```
var x= "Hello ";
var y= "world!";
var z= " Have a nice day!";
var m= x.concat(y , z);
console.log(m);
```

Hello world! Have a nice day!

indexOf()

- Returns the position of the first location of a specified value in a string. (case-sensitive), If the value is not present in a string, it returns -1.

Syntax

```
variableName.indexOf(str);
```

Example

```
var x = "Hello world, welcome.";
var y = x.indexOf("welcome");
console.log(y);
```

13

lastIndexOf()

- Returns the position of the last location of a specified value in a string. (case-sensitive), If the value is not present in a string, it returns -1.

Syntax

```
variableName.lastIndexOf(str);
```

Example

```
var x = "planet earth is a great planet.";
var y = x.lastIndexOf("planet");
console.log(y);
```

24

replace()

- Replaces a part of a given string with a new substring and return the new string. (case sensitive, replaces only first match)

Syntax

```
variableName.replace(str1,str2);
```

Example

```
var x= "Visit Cairo";
var y= x.replace("Cairo","Egypt");
console.log(y);
```

Visit Egypt

search()

- Searches a string for a value, and returns the position of the match. (case sensitive)

Syntax

```
variableName.search(str);
```

Example

```
var x = "Visit Cairo!";
var y = x.search("Cairo");
console.log(y);
```

6

slice()

- Extracts a part of a string and returns a new string.

Syntax

```
variableName.slice(startIndex,endIndex);
```

Example

```
var x= "Hello-world!";
var y = x.slice(1,5);
console.log(y);  // the last index is not included
```

```
ello
```

slice(-ve)

- Using negative number as an index causes the method to starts fetching from the end of the string.

Syntax

```
variableName.slice(-ve index);
```

Example

```
var x= "Hello-world!";
var y = x.slice(-5);
console.log(y);
```

```
orld!
```

split()

- Split a string into an array of substrings using a splitter letter.

Syntax

```
variableName.split("splitter");
```

Example

```
var x= "How are you doing today?";
var y= x.split(" ");
document.write(y[0],y[1]);
```

Howare

trim()

- Removes whitespace from both ends of a string.

Syntax

```
variableName.trim();
```

Example

```
var x = "      Hello World!      ";
y = x.trim();
console.log(y);
```

Hello World!

charAt()

- Returns the character at the specified index.

Syntax

```
variableName.charAt(index);
```

Example

```
var x = "NTI";
y = x.charAt(0);
console.log(y);
```

N

charCodeAt()

- Returns a number indicating the Unicode value of the character at the given index.

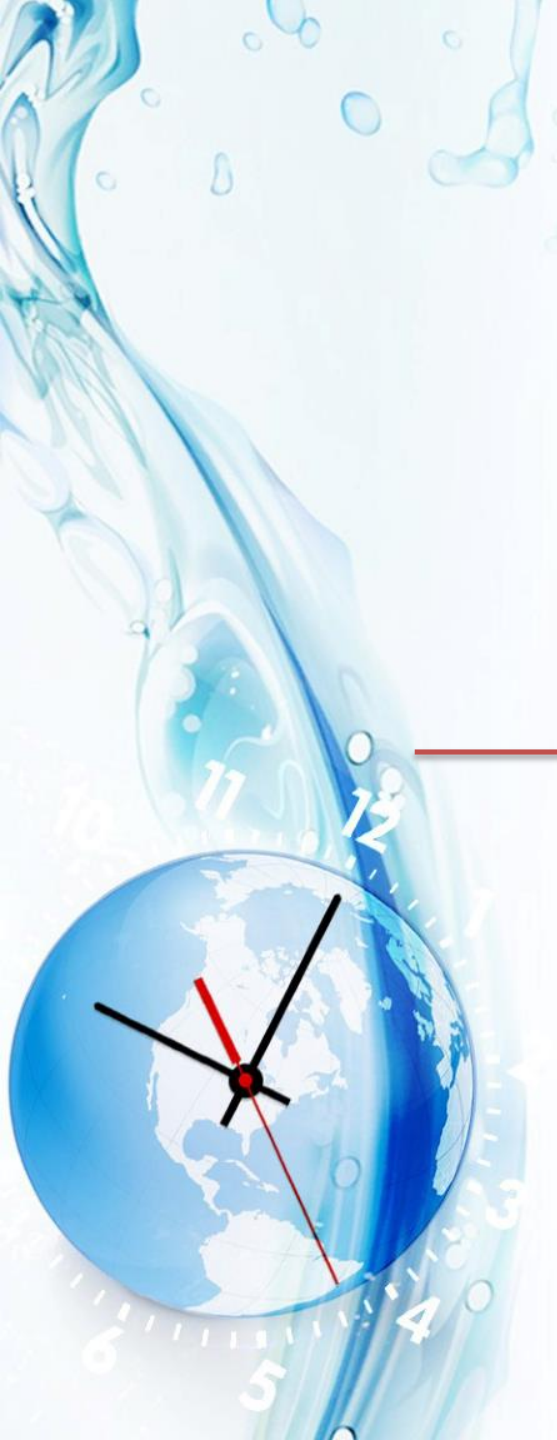
Syntax

```
variableName.charCodeAt(index);
```

Example

```
var x = "NTI";
y = x.charCodeAt(0);
console.log(y);
```

```
78
```



Number Methods

toString()

- Converts numbers to strings.

Syntax

```
variableName.toString();
```

Example

```
var x=10;

y=x.toString();

console.log(typeof y);
```

string

toFixed()

- Returns a string that represents a number with exact digits after a decimal point.

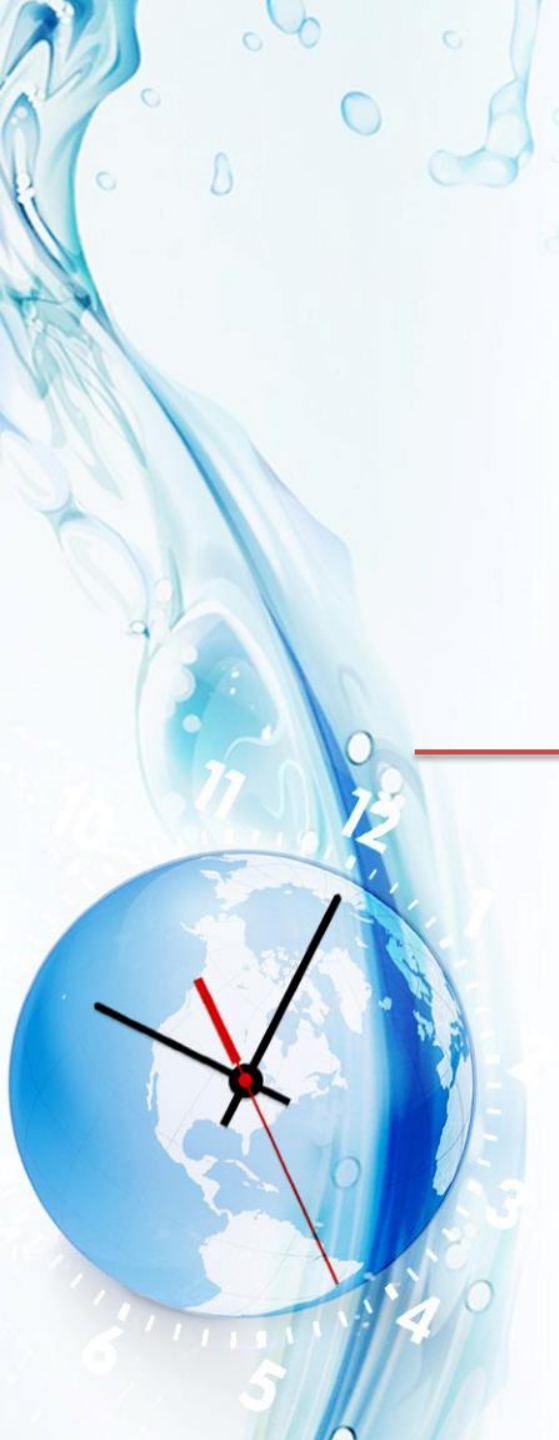
Syntax

```
variableName.toFixed(no of digits);
```

Example

```
var x=10.56789;
console.log(x.toFixed());
console.log(x.toFixed(3));
console.log(x.toFixed(2));
```

```
11
10.568
10.57
```



Array Properties

length

- Returns the length of an array; number of elements.

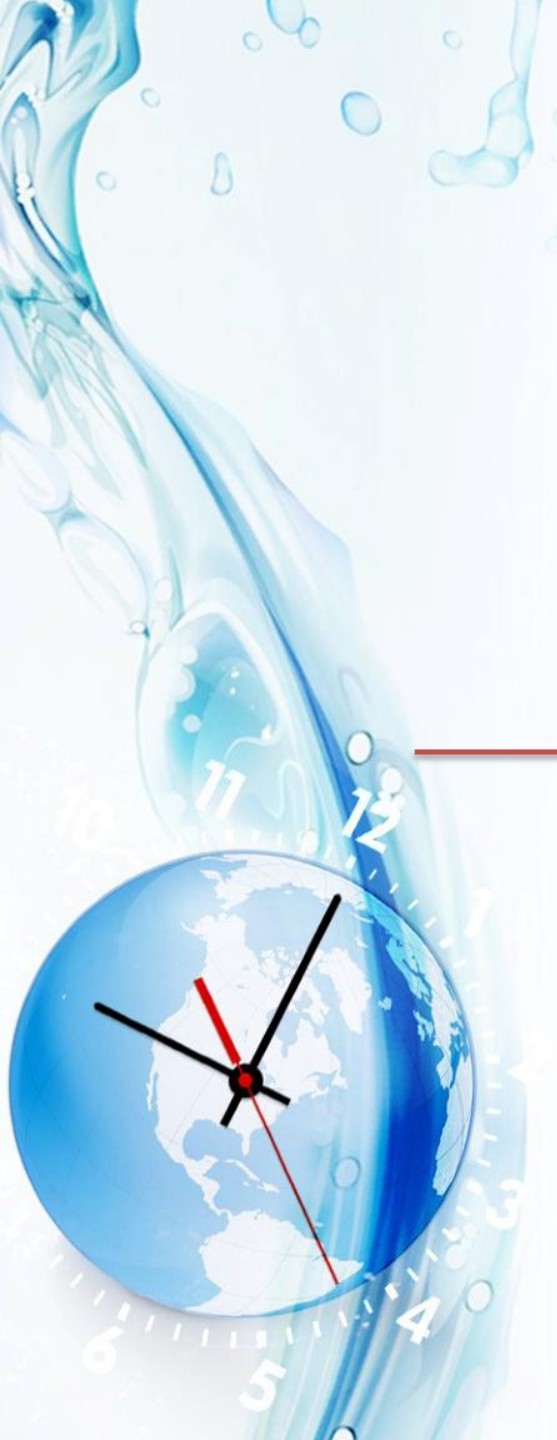
Syntax

```
arrayName.length;
```

Example

```
var colors = ["red", "green", "blue"];

console.log(colors.length);
```

```
3
```



Array Methods

toString()

- Returns the array values into a comma separated string.

Syntax

```
arrayName.toString();
```

Example

```
var colors = ["red", "green", "blue"];
var x=colors.toString();
console.log(x);
console.log(typeof x);
```

```
red,green,blue
string
```

join()

- Returns the array values joined as a string using custom joiner.

Syntax

```
arrayName.join("joiner");
```

Example

```
var colors = ["red", "green", "blue"];
var x= colors.join("*");
console.log(x);
console.log(typeof x);
```

```
red*green*blue
string
```

pop()

- Removes the last element from an array and returns the removed element.

Syntax

```
arrayName.pop();
```

Example

```
var colors = ["red", "green", "blue"];
var x= colors.pop()
console.log(x);
console.log(colors);
```

```
Blue
["red", "green"]
```

push()

- Adds an element to the end of an array and returns the new array length.

Syntax

```
arrayName.push("newElement");
```

Example

```
var colors = ["red", "green", "blue"];
var x= colors.push("black");
console.log(x);
console.log(colors);
```

```
4
["red", "green", "blue", "black"]
```

shift()

- Removes the first element from an array and "shifts" all other elements to a lower index and returns the removed element value.

Syntax

```
arrayName.shift();
```

Example

```
var colors = ["red", "green", "blue"];
var x= colors.shift();
console.log(x);
console.log(colors);
```

```
red
["green", "blue"]
```

unshift()

- Adds an element at the beginning of an array and "shifts" all other elements to a higher index and return the new array length.

Syntax

```
arrayName.unshift("newElement");
```

Example

```
var colors = ["red", "green", "blue"];
var x= colors.unshift("yellow");
console.log(x);
console.log(colors);
```

```
4
["yellow", "red", "green", "blue"]
```

reverse()

- Reverses the sequence of elements in an array and returns the reversed sequence.

Syntax

```
arrayName.reverse();
```

Example

```
var colors = ["red", "green", "blue"];
colors.reverse();
console.log(colors);
```

```
["blue", "green", "red"]
```

sort()

- Sorts the elements in an array alphabetically.

Syntax

```
arrayName.sort();
```

Example

```
var colors = ["red", "green", "blue"];
colors.sort();
console.log(colors);
```

```
["blue", "green", "red"]
```

You can use the sort method with the reverse method to sort an array in descending order.

Example

```
var colors = ["red", "green", "blue"];
colors.sort();          // Sorts the array in ascending order.
colors.reverse();       // Reverses the order of the sorted array
console.log(colors);
```

```
["red", "green", "blue"]
```

forEach()

- Calls a function once for each element in an array, in order.

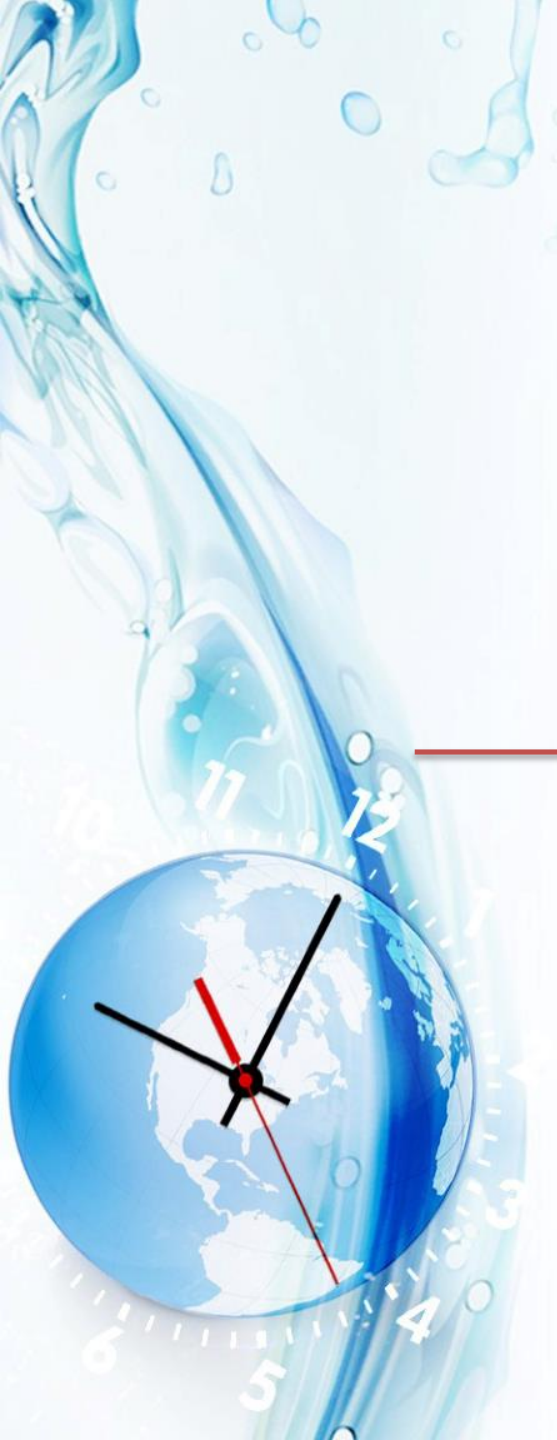
Syntax

```
arrayName.forEach(functionName);
```

Example

```
var arr = [10, 20, 30, 40];
arr.forEach(myFun1);
function myFun1(x, y) {
        // param x for values and param y for index
  document.write(y,"==> ",x,"<br>");
}
```

```
0 ==> 10
1 ==> 20
2 ==> 30
3 ==> 40
```



JS Date Object

JS Date Object

- The Date object is used to work with dates and times.
- Date objects are created with new Date() constructor.

Example: To get Today's Date

```
var myDate = new Date();
console.log(myDate);
```

```
Sat Oct 10 2020 20:35:55 GMT+0200 (Eastern European Standard Time)
```

JS Date Object

- *Also Date and time can be assigned in order to work with different date and time.*

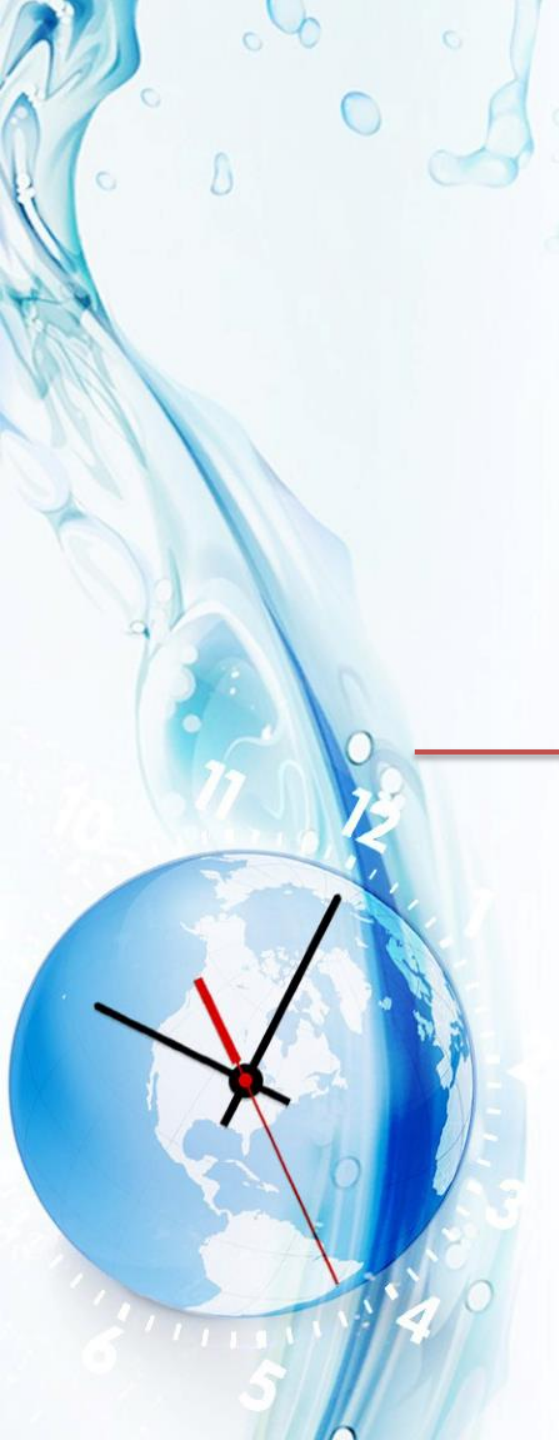
Syntax

```
var varName=new Date(year, mon_Index, day, hour, min, sec, ms);
```

Example: Date and time assigned (10/11/2018-8:35:55)

```
var myDate=new Date(2018, 10, 10, 20, 35, 55, 100);
console.log(myDate);
```

Sat Nov 10 2018 20:35:55 GMT+0200 (Eastern European Standard Time)



JavaScript Date Methods

JavaScript Date Methods

Method	Description
getDate()	Output ➔ 12 Returns the day of the month (from 1-31)
getDay()	Output ➔ 4 Returns the day of the week (from 0-6) (0 ➔ Sunday)
getFullYear()	Output ➔ 2018 Returns the year
getHours()	Output ➔ 22 Returns the hour (from 0-23)
getMilliseconds()	Output ➔ 650 Returns the milliseconds (from 0-999)

JavaScript Date Methods

Method	Description
getMinutes()	Output ➔ 35 Returns the minutes (from 0-59)
getMonth()	Output ➔ 5 Returns the month index (from 0-11)
getSeconds()	Output ➔ 33 Returns the seconds (from 0-59)
getTimezoneOffset()	Output ➔ -120 Returns the time difference between UTC time and local time, in minutes

JavaScript Date Methods

Method	Description
toDateString()	Output ➔ Sun Apr 15 2018 Converts the date portion of a Date object into a readable string
toLocaleDateString()	Output ➔ 4/15/2018 Returns the date portion of a Date object as a string, using locale conventions
toLocaleTimeString()	Output ➔ 9:39:01 AM Returns the time portion of a Date object as a string, using locale conventions

JavaScript Date Methods

Method	Description
toLocaleString()	Output ➔ 4/15/2018, 9:39:23 AM Converts a Date object to a string, using locale conventions
toString()	Output ➔ Thu Apr 15 2018 20:39:54 GMT+0200 (Egypt Standard Time) Converts a Date object to a string
toTimeString()	Output ➔ 20:41:11 GMT+0200 (Egypt Standard Time) Converts the time portion of a Date object to a string

JavaScript Date Methods

Example: No date assigned (work as today's date)

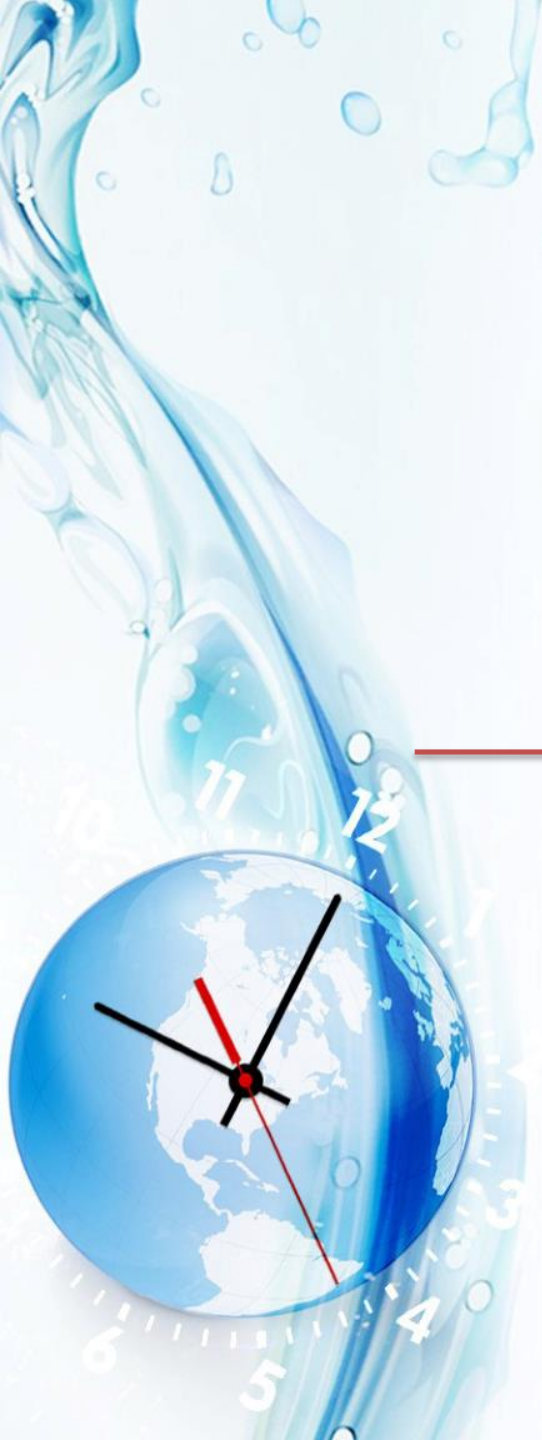
```
var d=new Date();
var x=d.toDateString();
document.write(x);
```

Thu Apr 15 2018

Example: Date and time assigned (10/11/2018-8:35:55)

```
var d=new Date(2018, 10, 10, 20, 35, 55, 100);
var x=d.toDateString();
document.write(x);
```

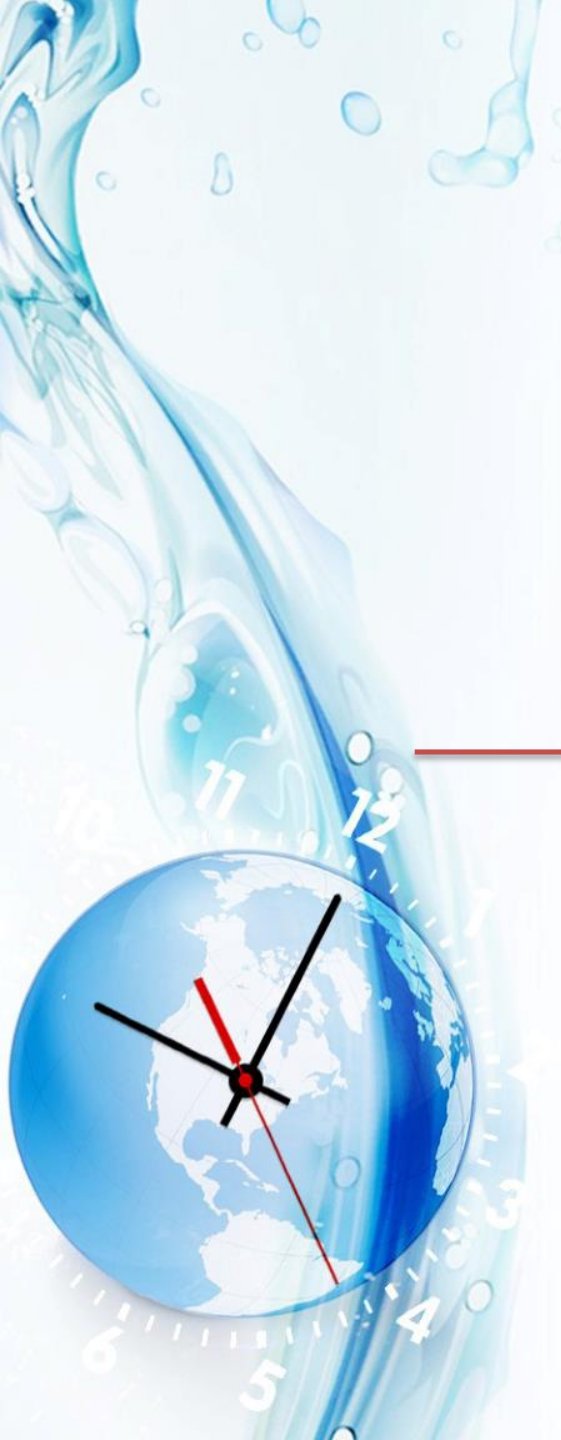
Sat Nov 10 2018



JavaScript Math Object

The Math object allows to perform mathematical tasks.

All properties/methods of Math object can be called by using the Math object, without creating it.



JavaScript Math Object Properties

Math PI

- Returns the value of PI. (22/7)

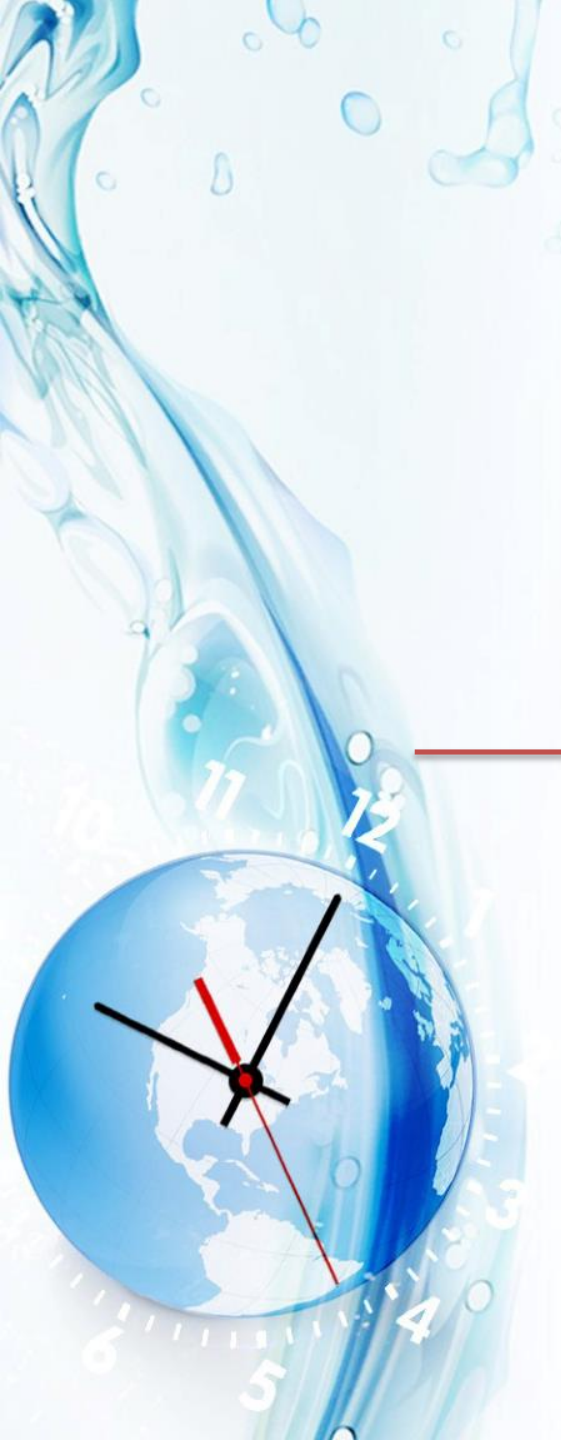
Syntax

```
Math.PI;
```

Example

```
var x = Math.PI;

console.log(x);
```

```
3.141592653589793
```



JS Math Object Methods

JS Math Object Methods

Method	Description
Math.abs(-50.5)	Output ➔ 50.5 Returns the absolute value of a number.
Math.cbrt(125)	Output ➔ 5 Returns the cubic root of a number.
Math.ceil(70.01)	Output ➔ 71 Round a number upward to its nearest integer.
Math.floor(1.99)	Output ➔ 1 Round a number downward to its nearest integer.
Math.cos(3)	Output ➔ -0.9899924966004454 Returns a numeric value between -1 and 1, which represents the cosine of the angle. (x is in radians)

JS Math Object Methods

Method	Description
Math.pow(4, 3)	Output ➔ 64 (4^3) Returns the value of the first number to the power of second number.
Math.random()	Output ➔ Ex: 0.5228992054729398 Return a random number between 0 and 1.
Math.round(10.5)	Output ➔ 11 Round a number to nearest integer.
Math.sin(3)	Output ➔ 0.1411200080598672 Returns the sine of a number (x is in radians).
Math.sqrt(9)	Output ➔ 3 Return the square root of a number.

JS Math Object Methods

Method	Description
Math.tan(90)	Output ➔ -1.995200412208242 Return the tangent of a number (angle).
Math.trunc(8.76)	Output ➔ 8 Return the integer part of a number.
Math.min(10,20,5,30)	Output ➔ 5 Returns the lowest value in a list of parameters.
Math.max(10,20,5,30)	Output ➔ 30 Returns the highest value in a list of parameters.