



**CRYSTALLIZE**  
**TECHNOLOGIES**

# FULL STACK WEB DEVELOPMENT

JS

## ABSTRACT

Full stack web development encompasses all aspects of building a website or web application, from the user interface to the back-end logic and database management. This guide provides an overview of the key technologies and concepts involved in full stack development.

**Madhan HK**  
Front-End JS

# JAVASCRIPT

**JavaScript** is the most powerful and versatile web programming language. It is used for making the websites interactive. JavaScript helps us add features like animations, interactive forms and dynamic content to web pages.

JavaScript is a programming language used for creating dynamic content on websites. It is a lightweight, cross-platform and single-threaded programming language. JavaScript is an interpreted language that executes code line by line providing more flexibility. It is a commonly used programming language to create dynamic and interactive elements in web applications.

It is also known as the scripting language for webpages.

JavaScript can be used for Client-side developments as well as Server-side developments.

- **Client-side:** It supplies objects to control a browser and its Document Object Model (DOM). Like if client-side extensions allow an application to place elements on an HTML form and respond to user events such as mouse clicks, form input, and page navigation. Useful libraries for the client side are AngularJS, ReactJS, VueJS, and so many others.
- **Server-side:** It supplies objects relevant to running JavaScript on a server. For if the server-side extensions allow an application to communicate with a database, and provide continuity of information from one invocation to another of the application, or perform file manipulations on a server. The useful framework which is the most famous these days is node.js

## How to Link JavaScript File in HTML ?

JavaScript can be added to HTML file in two ways:

**Internal JS:** We can add JavaScript directly to our HTML file by writing the code inside the `<script>` tag. The `<script>` tag can either be placed inside the `<head>` or the `<body>` tag according to the requirement.

External JS: We can write JavaScript code in another files having an extension.js and then link this file inside the <head> tag of the HTML file in which we want to add this code..

### **Syntax:**

```
<script>
  // JavaScript Code
</script>
```

### Features of JavaScript

According to a recent survey conducted by Stack Overflow, JavaScript is the most popular language on earth.

With advances in browser technology and JavaScript having moved into the server with Node.js and other frameworks, JavaScript is capable of so much more. Here are a few things that we can do with JavaScript:

- JavaScript was created in the first place for DOM manipulation. Earlier websites were mostly static, after JS was created dynamic Web sites were made.
- Functions in JS are objects. They may have properties and methods just like other objects. They can be passed as arguments in other functions.
- Can handle date and time.
- Performs Form Validation although the forms are created using HTML.
- No compiler is needed.

### Single Line Comments

A single-line comment in JavaScript is denoted by two forward slashes (//),

### **Syntax:**

```
// your comment here
```

### **Multi-line Comments**

A multiline comment in JavaScript is a way to include comments that span multiple lines in the source code.

### **Syntax:**

```
/*  
This is a multiline comment  
It can span multiple lines  
*/
```

## Variables

In JavaScript, variables are used to store and manage data. They are created using the var, let, or const keyword.

### var Keyword

The var keyword is used to declare a variable. It has a function-scoped or globally-scoped behavior.

```
var x = 10;
```

### let Keyword

The let keyword is a block-scoped variables. It's commonly used for variables that may change their value. Let can be updated but not redeclared.

```
let y = "Hello";
```

### const Keyword

The const keyword declares variables that cannot be reassigned or updated. It's block-scoped as well.

```
const PI = 3.14;
```

## Data Types

JavaScript is a dynamically typed (also called loosely typed) scripting language. In JavaScript, variables can receive different data types over time. The latest ECMAScript standard defines eight data types Out of which seven data types are Primitive (predefined) and one complex or Non-Primitive.

### Primitive Data Types

The predefined data types provided by JavaScript language are known as primitive data types. Primitive data types are also known as in-built data types.

1. **Number:** JavaScript numbers are always stored in double-precision 64-bit binary format IEEE 754. Unlike other programming languages, you don't need int, float, etc to declare different numeric values.
2. **Null:** This type has only one value that is null.
3. **String:** JavaScript Strings are similar to sentences. They are made up of a list of characters, which is essentially just an "array of characters, like "Hello GeeksforGeeks" etc.
4. **Symbol:** Symbols return unique identifiers that can be used to add unique property keys to an object that won't collide with keys of any other code that might add to the object.
- 5.
6. **Boolean:** Represent a logical entity and can have two values: true or false.
7. **BigInt:** BigInt is a built-in object in JavaScript that provides a way to represent whole numbers larger than  $2^{53}-1$ .
8. **Undefined:** A variable that has not been assigned a value is undefined.

## Non-Primitive Data Types

The data types that are derived from primitive data types of the JavaScript language are known as non-primitive data types.

- **Object:** It is the most important data type and forms the building blocks for modern JavaScript. Similar to dictionary in Python.

## JavaScript String

**JavaScript String** is a sequence of characters, typically used to represent text. It is enclosed in single or double quotes and supports various methods for text manipulation.

### Basic Terminologies of JavaScript String

- **String:** A sequence of characters enclosed in single (' ') or double (" ") quotes.
- **Length:** The number of characters in a string, obtained using the length property.
- **Index:** The position of a character within a string, starting from 0.
- **Concatenation:** The process of combining two or more strings to create a new one.

- **Substring:** A portion of a string, obtained by extracting characters between specified indices.

## Declaration of a String

### 1. Using Single Quotes

Single Quotes can be used to create a string in JavaScript. Simply enclose your text within single quotes to declare a string.

#### Syntax:

```
let str = 'String with single quote';
```

### 2. Using Double Quotes

Double Quotes can also be used to create a string in JavaScript. Simply enclose your text within double quotes to declare a string.

#### Syntax:

```
let str = "String with double quote";
```

### 3. String Constructor

You can create a string using the String Constructor. The String Constructor is less common for direct string creation, it provides additional methods for manipulating strings. Generally, using string literals is preferred for simplicity.

```
let str = new String('Create String with String Constructor');  
console.log(str);
```

### 4. Using Template Literals (String Interpolation)

You can create strings using Template Literals. Template literals allow you to embed expressions within backticks (``) for dynamic string creation, making it more readable and versatile.

#### Syntax:

```
let str = 'Template Literal String';  
let newStr = `String created using ${str}`;
```

## 5. Multiline Strings (ES6 and later)

You can create a multiline string using backticks (``) with template literals. The backticks allows you to span the string across multiple lines, preserving the line breaks within the string.

### Syntax:

```
let str = `
  This is a
  multiline
  string`;
```

## Basic JavaScript String Methods

JavaScript provides several built-in methods and properties for working with strings. Below is the list of JavaScript string functions that you need to know, for mastering JavaScript strings.

- JavaScript slice() Method
- JavaScript substring() Method
- JavaScript substr() Method
- JavaScript replace()
- JavaScript replaceAll()
- JavaScript toUpperCase()
- JavaScript toLowerCase()
- JavaScript concat() Method
- JavaScript trim() Method
- JavaScript trimStart() Method
- JavaScript trimEnd() Method
- JavaScript padStart() Method
- JavaScript padEnd() Method
- JavaScript charAt() Method
- JavaScript charCodeAt() Method
- JavaScript split() Method

## JavaScript Boolean

JavaScript Boolean represents true or false values. It's used for logical operations, condition testing, and variable assignments based on conditions. Values like 0, NaN, empty strings,

undefined, and null are false; non-empty strings, numbers other than 0, objects, and arrays are true.

Note: A variable or object which has a value is treated as a true boolean value. '0', 'NaN', empty string, 'undefined', and 'null' is treated as false boolean values.

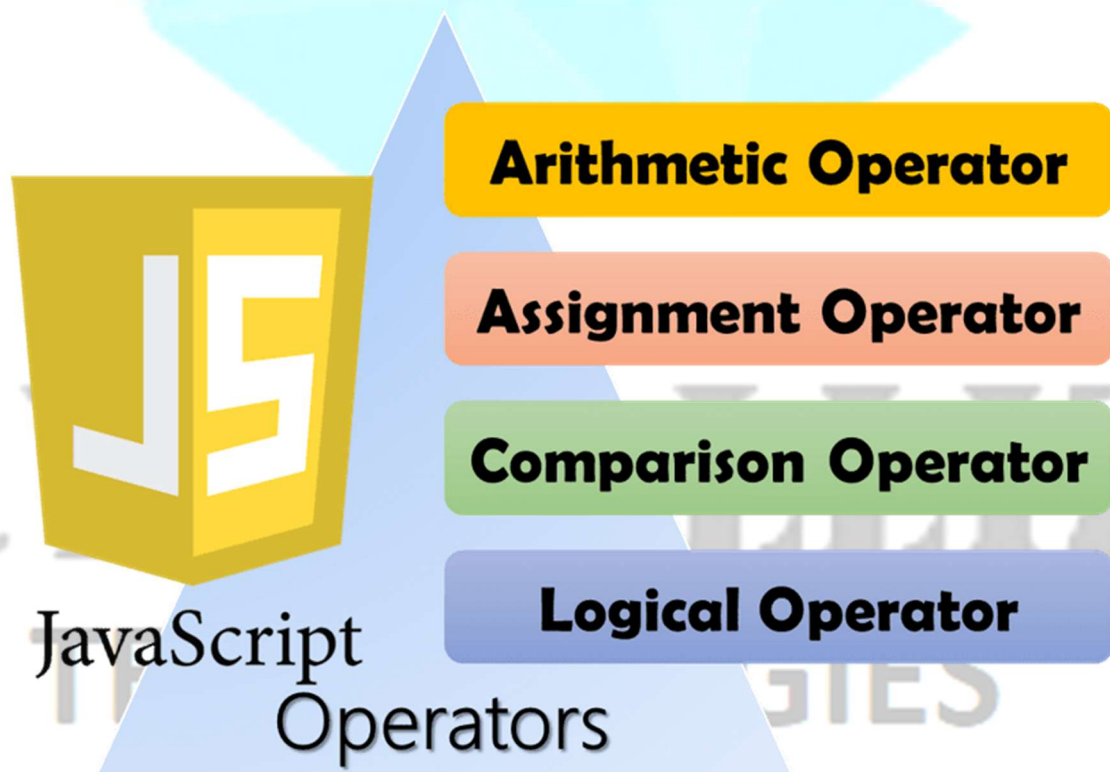
## **Global and Local variables in JavaScript**

### **Global Variables**

Global variables are those defined outside of any function, making them accessible from anywhere within the script. Regardless of the keyword used for declaration, they all behave similarly and possess a global scope. Even variables declared without a keyword inside a function are considered global.

### **Local Variables**

Local variables are defined within functions in JavaScript. They are confined to the scope of the function that defines them and cannot be accessed from outside. Attempting to access local variables outside their defining function results in an error.





// Arithmetic Operators

let a = 10;

let b = 4;

console.log("a + b = ", a + b)

console.log("a - b = ", a - b)

console.log("a / b = ", a / b)

console.log("a \*\* b = ", a \*\* b)

console.log("a % b = ", a % b)

console.log("++a = ", ++a)

console.log("a++ = ", a++)

console.log("--a = ", --a)

console.log("a-- = ", a--)

console.log("a = ", a)

console.log("a-- = ", a--)

// Assignment Operators

let assignment = 1;

assignment += 5 // same as assignment = assignment + 5

console.log("a is now = ", a)

assignment -= 5 // same as assignment = assignment - 5

console.log("a is now = ", a)

assignment \*= 5 // same as assignment = assignment \* 5

console.log("a is now = ", a)

assignment /= 5 // same as assignment = assignment / 5

```
console.log("a is now = ", a)
```

```
// Comparison Operators
```

```
let comp1 = 6;
```

```
let comp2 = 7;
```

```
console.log("comp1 == comp2 is ", comp1 == comp2)
```

```
console.log("comp1 != comp2 is ", comp1 != comp2)
```

```
console.log("comp1 === comp2 is ", comp1 === comp2)
```

```
console.log("comp1 !== comp2 is ", comp1 !== comp2)
```

```
console.log("comp1 > comp2 is ", comp1 > comp2)
```

```
// Logical Operators
```

```
let x = 5;
```

```
let y = 6;
```

```
console.log(x < y && x == 5)
```

```
console.log(x > y || x == 5)
```

```
console.log(!false)
```

```
console.log(!true)
```

## Conditional Statements

Very often when you write code, you want to perform different actions for different decisions.

You can use conditional statements in your code to do this.

In JavaScript we have the following conditional statements:

- Use if to specify a block of code to be executed, if a specified condition is true

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

```
1 let a = prompt("Hey whats you age?");
2 a = Number.parseInt(a); // Converting the string to a number
3 if(a<0){
4     alert("This is an invalid age");
5 }
6 else if(a<9){
7     alert("You are a kid and you cannot even think of driving");
8 }
9 else if(a<18 && a>=9){
10    alert("You are a kid and you can think of driving after 18");
11 }
12 else{
13    alert("You can now drive as you are above 18");
14 }
15 console.log("Done")
```

## The JavaScript Switch Statement

Use the **switch** statement to select one of many code blocks to be executed.

### Syntax

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

### Example

The **getDay()** method returns the weekday as a number between 0 and 6.

(Sunday=0, Monday=1, Tuesday=2 ..)

This example uses the weekday number to calculate the weekday name:

```
switch (new Date().getDay()) {  
  case 0:  
    day = "Sunday";  
    break;  
  case 1:  
    day = "Monday";  
    break;  
  case 2:  
    day = "Tuesday";  
    break;  
  case 3:  
    day = "Wednesday";  
    break;  
  case 4:  
    day = "Thursday";  
    break;  
  case 5:  
    day = "Friday";  
    break;  
  case 6:  
    day = "Saturday";  
}
```

## Different Kinds of Loops

JavaScript supports different kinds of loops:

- for - loops through a block of code a number of times
- for/in - loops through the properties of an object
- for/of - loops through the values of an iterable object
- while - loops through a block of code while a specified condition is true
- do/while - also loops through a block of code while a specified condition is true

### The For Loop

The for statement creates a loop with 3 optional expressions:

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

Expression 1 is executed (one time) before the execution of the code block.

Expression 2 defines the condition for executing the code block.

Expression 3 is executed (every time) after the code block has been executed.

## Example

```
for (let i = 0; i < 5; i++) {  
  text += "The number is " + i + "<br>";  
}
```

```
1 // Program to add first n natural numbers  
2 let sum = 0  
3 let n = prompt("Enter the value of n")  
4 n = Number.parseInt(n)  
5 for (let i = 0; i < n; i++) {  
6   sum += (i+1)  
7   console.log((i+1), "+")  
8 }  
9 console.log("Sum of first " + n + " natural numbers is " + sum)
```

## The For In Loop

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

### Syntax

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

### Example

```
const person = {fname:"John", lname:"Doe", age:25};  
  
let text = "";  
for (let x in person) {  
  text += person[x];  
}
```

### Example Explained

- The **for in** loop iterates over a **person** object
- Each iteration returns a **key** (x)
- The key is used to access the **value** of the key
- The value of the key is **person[x]**

## The For Of Loop

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

It lets you loop over iterable data structures such as Arrays, Strings, Maps, NodeLists, and more:

### Syntax

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

#### Example :

```
let language = "JavaScript";
```

```
let text = "";
```

```
for (let x of language) {
```

```
  text += x;
```

```
}
```

## The While Loop

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

### Syntax

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

## Example

```
while (i < 10) {  
  text += "The number is " + i;  
  i++;  
}
```

## The Do While Loop

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

## Syntax

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

## Example

The example below uses a do while loop. The loop will always be executed at least once, even if the condition is false, because the code block is executed before the condition is tested:

## Example

```
do {  
  text += "The number is " + i;  
  i++;  
}  
while (i < 10);
```

## Functions in JavaScript

A JavaScript function is a block of code designed to perform a particular task.

A JavaScript function is executed when "something" invokes it (calls it).

### Example

```
// Function to compute the product of p1 and p2  
function myFunction(p1, p2) {  
  return p1 * p2;  
}
```

## JavaScript Function Syntax

A JavaScript function is defined with the function keyword, followed by a **name**, followed by parentheses ().

Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).

The parentheses may include parameter names separated by commas: (*parameter1*, *parameter2*, ...)

The code to be executed, by the function, is placed inside curly brackets: {}

- Function **parameters** are listed inside the parentheses () in the function definition.
- Function **arguments** are the **values** received by the function when it is invoked.
- Inside the function, the arguments (the parameters) behave as local variables.

## Example

Calculate the product of two numbers, and return the result:

```
// Function is called, the return value will end up in x
```

```
let x = myFunction(4, 3);
```

```
function myFunction(a, b) {
```

```
// Function returns the product of a and b
```

```
  return a * b;
```

```
}
```

## JavaScript Arrays

### Creating an Array

Using an array literal is the easiest way to create a JavaScript Array.

Syntax:

```
const array_name = [item1, item2, ...];
```



## Accessing Array Elements

You access an array element by referring to the index number:

```
const cars = ["Saab", "Volvo", "BMW"];
let car = cars[0];
```

## Changing an Array Element

This statement changes the value of the first element in cars:

```
cars[0] = "Opel";
```

## Converting an Array to a String

The JavaScript method `toString()` converts an array to a string of (comma separated) array values.

### Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.toString();
```

```
1  // Array Methods
2  let num = [1, 2, 3, 34, 4]
3  let b = num.toString() // b is now a string
4  console.log(b, typeof b)
5  let c = num.join(" and ")
6  console.log(c, typeof c)
7  // let r = num.pop() // pop returns the popped element
8  // console.log(num, r)
9  // let r = num.push(56) // push returns the new array length
10 // console.log(num, r)
11 // let r = num.shift()
12 // console.log(r, num) // Removes an element from the start of the array
13
14 let r = num.unshift(78)
15 console.log(r, num)
16 console.log(r)
```

## Array Methods

```
1 // let num = [1, 2, 3, 4, 5, 6, 7, 8, 9]
2 // let num_more = [11, 12, 13, 14, 15, 16, 17, 18, 19]
3 // let num_even_more = [211, 212, 213, 214, 415, 416, 417, 418, 419]
4 // console.log(num.length)
5 // delete num[0]
6 // console.log(num.length)
7
8 // let newArray = num.concat(num_more, num_even_more)
9 // console.log(newArray)
10 // console.log(num, num_more)
11
12 // sort method
13 // let compare = (a, b)=>{
14 //   return b - a
15 // }
16 // let num = [551, 22, 3, 14, 5, 6, 7, 8, 229]
17 // num.sort(compare)
18 // num.reverse()
19 // console.log(num)
20
21 // Splice and Slice
22 let num = [551, 22, 3, 14, 5, 6, 7, 8, 229]
23 // let deletedValues = num.splice(2, 4, 1021, 1022, 1023, 1024, 1025)
24 // console.log(num)
25 // console.log(deletedValues)
26
27 // let newNum = num.slice(3)
28 let newNum = num.slice(3, 5)
29 console.log(newNum)
```

## Loopings In Array

```
1 let num = [3, 5, 1, 2, 4]
2
3 // for(let i=0; i<num.length;i++){
4 //   console.log(num[i])
5 // }
6
7 // ForEach Loop
8 num.forEach((element) => {
9   console.log(element * element)
10 })
11
12 // Array.from
13 let name = "Harry"
14 let arr = Array.from(name)
15 console.log(arr)
16
17 // for...of
18 for (let item of num){
19   console.log(item)
20 }
21
22 // for...in
23 for (let i in num){
24   console.log(num[i])
25 }
```

## Map, Filter and Reduce Functions in JS

The `map()`, `reduce()` and `filter()` are array functions that transform the array according to the applied function and return the updated array. They are used to write simple, short and clean codes for modifying an array instead of using the loops.

**map() method:** It applies a given function on all the elements of the array and returns the updated array. It is the simpler and shorter code instead of a loop. The map is similar to the following code:

```
arr = new Array(1, 2, 3, 6, 5, 4);  
  
for(let i = 0; i < 6; i++) {  
    arr[i] *= 3;  
}
```

### Syntax:

```
array.map(function_to_be_applied)  
  
array.map(function (args) {  
    // code;  
})
```

### Example:

```
function triple(n){  
    return n*3;  
}  
  
arr = new Array(1, 2, 3, 6, 5, 4);  
  
var new_arr = arr.map(triple)  
console.log(new_arr);
```

**reduce() method:** It reduces all the elements of the array to a single value by repeatedly applying a function. It is an alternative of using a loop and updating the result for every scanned element. Reduce can be used in place of the following code:

```
arr = new Array(1, 2, 3, 6, 5, 4);  
  
result = 1  
  
for(let i = 0; i < 6; i++) {  
    result = result * arr[i];  
}
```

**Syntax:**

```
array.reduce(function_to_be_applied)  
  
array.reduce(function (args) {  
    // code;  
})
```

**Example:**

```
function product(a, b){  
    return a * b;  
}  
  
arr = new Array(1, 2, 3, 6, 5, 4);
```

```
var product_of_arr = arr.reduce(product)  
console.log(product_of_arr)
```

**filter() method:** It filters the elements of the array that return false for the applied condition and returns the array which contains elements that satisfy the applied condition. It is a simpler and shorter code instead of the below code using a loop:

```
arr = new Array(1, 2, 3, 6, 5, 4);  
  
new_arr = {}  
  
for(let i = 0; i < 6; i++) {  
    if(arr[i] % 2 == 0) {  
        new_arr.push(arr[i]);  
    }  
}
```

**Syntax:**

```
array.filter(function_to_be_applied)  
  
array.filter(function (args) {  
    // condition;  
})
```


**Example:**

```
arr = new Array(1, 2, 3, 6, 5, 4);  
  
var new_arr = arr.filter(function (x){  
    return x % 2==0;  
});  
  
console.log(new_arr)
```

# JavaScript Objects

## Real Life Objects

In real life, **objects** are things like: houses, cars, people, animals, or any other subjects.

Car Object	Properties	Methods
	car.name = Fiat	car.start()
	car.model = 500	car.drive()
	car.weight = 850kg	car.brake()
	car.color = white	car.stop()

Objects are variables too. But objects can contain many values.

This code assigns **many values** (Fiat, 500, white) to an **object** named car:

### Example

```
const car = {type:"Fiat", model:"500", color:"white"};
```

Note: It is a common practice to declare objects with the const keyword.

## JavaScript Object Properties

An Object is an Unordered Collection of Properties

- Properties are the most important part of JavaScript objects.
- Properties can be changed, added, deleted, and some are read only.

## Accessing JavaScript Properties

The syntax for accessing the property of an object is:

```
// objectName.property
```

```
let age = person.age;
```

or

```
//objectName["property"]
```

```
let age = person["age"];
```

or

```
//objectName[expression]
```

```
let age = person[x];
```

## Adding New Properties

You can add new properties to an existing object by simply giving it a value:

### Example

```
person.nationality = "English";
```

## Deleting Properties

The delete keyword deletes a property from an object:

### Example

```
const person = {  
  firstName: "John",  
  lastName: "Doe",  
  age: 50,  
  eyeColor: "blue"  
};
```

```
delete person.age;
```

## Nested Objects

Property values in an object can be other objects:

### Example

```
myObj = {  
  name: "John",  
  age: 30,  
  myCars: {  
    car1: "Ford",  
    car2: "BMW",  
    car3: "Fiat"
```

```
}  
}
```

## JavaScript Object Methods

**Object methods** are actions that can be performed on objects.

A method is a **function definition** stored as a **property value**.

### Example

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

In the example above, this refers to the **person object**:

- **this.firstName** means the **firstName** property of **person**.
- **this.lastName** means the **lastName** property of **person**.

## Accessing Object Methods

You access an object method with the following syntax:

*objectName.methodName()*

## Adding a Method to an Object

Adding a new method to an object is easy:

### Example

```
person.name = function () {  
  return this.firstName + " " + this.lastName;  
};
```



## JavaScript Object Constructors

Sometimes we need to create many objects of the same **type**.

To create an **object type** we use an **object constructor function**.

It is considered good practice to name constructor functions with an upper-case first letter.

Object Type Person

```
function Person(first, last, age, eye) {  
  this.firstName = first;  
  this.lastName = last;  
  this.age = age;  
  this.eyeColor = eye;  
}
```

Note:

In the constructor function, this has no value.

The value of this will become the new object when a new object is created.

Now we can use new Person() to create many new Person objects:

### Example

```
const myFather = new Person("John", "Doe", 50, "blue");  
const myMother = new Person("Sally", "Rally", 48, "green");  
const mySister = new Person("Anna", "Rally", 18, "green");  
  
const myself = new Person("Johnny", "Rally", 22, "green");
```

**Data properties in JavaScript have four attributes:**

- **value:** The property's value.
- **writable:** When true, the property's value can be changed
- **enumerable:** When true, the property can be iterated over by “for-in” enumeration.  
Otherwise, the property is said to be non-enumerable.

- **configurable:** If false, attempts to delete the property, change the property to be an access-or property, or change its attributes (other than [[Value]], or changing [[Writable]] to false) will fail.

## JavaScript Object Prototypes

All JavaScript objects inherit properties and methods from a prototype.

### Using the **prototype** Property

The JavaScript prototype property allows you to add new properties to object constructors:

#### Example

```
function Person(first, last, age, eyecolor) {  
  this.firstName = first;  
  this.lastName = last;  
  this.age = age;  
  this.eyeColor = eyecolor;  
}
```

```
Person.prototype.nationality = "English";
```

The JavaScript prototype property also allows you to add new methods to objects constructors:

#### Example

```
function Person(first, last, age, eyecolor) {  
  this.firstName = first;  
  this.lastName = last;  
  this.age = age;  
  this.eyeColor = eyecolor;  
}  
  
Person.prototype.name = function() {  
  return this.firstName + " " + this.lastName;  
};
```

# JavaScript Popup Boxes

JavaScript has three kind of popup boxes: Alert box, Confirm box, and Prompt box.

## Alert Box

An alert box is often used if you want to make sure information comes through to the user.

When an alert box pops up, the user will have to click "OK" to proceed.

### Syntax

```
window.alert("sometext");
```

The `window.alert()` method can be written without the window prefix.

#### Example

```
alert("I am an alert box!");
```

## Confirm Box

A confirm box is often used if you want the user to verify or accept something.

When a confirm box pops up, the user will have to click either "OK" or "Cancel" to proceed.

If the user clicks "OK", the box returns **true**. If the user clicks "Cancel", the box returns **false**.

### Syntax

```
window.confirm("sometext");
```

The `window.confirm()` method can be written without the window prefix.

#### Example

```
if (confirm("Press a button!")) {  
    txt = "You pressed OK!";  
} else {  
    txt = "You pressed Cancel!";  
}
```

## Prompt Box

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

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

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

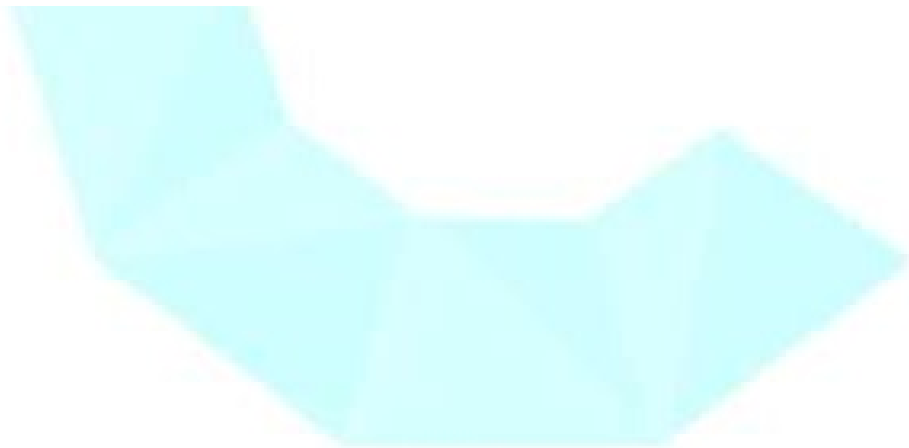
## Syntax

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

The `window.prompt()` method can be written without the window prefix.

### Example

```
let person = prompt("Please enter your name", "Harry Potter");
let text;
if (person == null || person == "") {
  text = "User cancelled the prompt.";
} else {
  text = "Hello " + person + "! How are you today?";
}
```



CRYSTALLIZE  
TECHNOLOGIES