**JavaScript**

JavaScript is dynamically typed scripting language.

JavaScript supports 3 primitive datatypes:

* **Numbers**: 5, 6.5, 7 etc.
* **String**: “Hello GeeksforGeeks” etc.
* **Boolean**: true, false.

JavaScript also supports two trivial datatypes:

* Null
* undefined

composite datatype

* object.

Syntax:

var var\_name;

**identifiers =variable name**

JavaScript is also known as **untyped** language.

**Variable Scope in JavaScript**

Scope of a variable is the part of the program from where the variable may directly be accessible.

In JavaScript, there are two types of scopes:

1. Global Scope – Scope outside the outermost function attached to Window
2. Local Scope – Inside the function being executed

If you **don’t use the var keyword**, then the **variables are by default created in the global** **scope.**

Always remember, JavaScript has function scope and not block scope.

function fun(){

if(true){

var i=100;

}

Console.log(i);

}

fun();

//output 100

you want to implement something like “private variables of class in C++/Java” in JavaScript.



if-else Statement in JavaScript.

Decision Making

A programming language uses control statements to control the flow of execution of the program based on certain conditions.

JavaScript’s conditional statements:

* [if](https://www.geeksforgeeks.org/else-statement-javascript/#if)
* [if-else](https://www.geeksforgeeks.org/else-statement-javascript/#if-else)
* [nested-if](https://www.geeksforgeeks.org/else-statement-javascript/#nested-if)
* [if-else-if](https://www.geeksforgeeks.org/else-statement-javascript/#if-else-if)
* [**if**](https://www.geeksforgeeks.org/else-statement-javascript/#if)

Syntax:

if(condition)

{

// Statements to execute if

// condition is true

}

* [if-else](https://www.geeksforgeeks.org/else-statement-javascript/#if-else)

if (condition)

{

// Executes this block if

// condition is true

}

else

{

// Executes this block if

// condition is false

}

* [nested-if](https://www.geeksforgeeks.org/else-statement-javascript/#nested-if)

if (condition1)

{

// Executes when condition1 is true

if (condition2)

{

// Executes when condition2 is true

}

}

* [if-else-if](https://www.geeksforgeeks.org/else-statement-javascript/#if-else-if)

if (condition)

statement;

else if (condition)

statement;

.

.

else

statement;

# Switch Case in JavaScript

The **switch case** statement in JavaScript is also used for decision making purposes.

The switch case statement is a multiway branch statement.

switch (expression)

{

case value1:

statement1;

break;

case value2:

statement2;

break;

.

.

case valueN:

statementN;

break;

default:

statementDefault;

}

# Loops in JavaScript

Looping in programming languages is a feature which facilitates the execution of a set of instructions/functions repeatedly while some condition evaluates to true.

<script type = "text/javascript">

var i;

for (i = 0; i < 10; i++)

{

document.write("Hello World!\n");

}

< /script>

**There are mainly two types of loops:**

1. **Entry Controlled loops**: In this type of loops the test condition is tested before entering the loop body. **For Loop** and **While Loop** are entry controlled loops.
2. **Exit Controlled Loops**: In this type of loops the test condition is tested or evaluated at the end of loop body. Therefore, the loop body will execute atleast once, irrespective of whether the test condition is true or false. **do – while loop** is exit controlled loop.

for (initialization condition; testing condition;

increment/decrement)

{

statement(s)

}

while (boolean condition)

{

loop statements...

}

**for…in loop**

JavaScript also includes another version of for loop also known as the for..in Loops. The for..in loop provides a simpler way to iterate through the properties of an object.

for (variableName in Object)

{

statement(s)

}

**do while:**

do

{

statements..

}

while (condition);

# Functions in JavaScript

A function is a set of statements that take inputs, do some specific computation and produces output. Basically, a function is a set of statements that performs some tasks or does some computation and then returns the result to the use

function functionName(Parameter1, Parameter2, ..)

{

// Function body

}

# JavaScript Backend basics

**JavaScript Engine**

Each browser has its own JavaScript engine which is used to support the JavaScript scripts in order for them to work properly. Below are the names of the JavaScript engines used in some of the most popular browsers out there.

* Chrome: V8
* Firefox: SpiderMonkey
* Safari: JavaScript Core
* Microsoft Edge/ Internet Explorer: Chakra

[**ECMA Script Standard**](https://en.wikipedia.org/wiki/ECMAScript)**:**The ECMA Script standard is a trademark scripting-language specification standardized by European Computer Manufactures Association. The ECMA Script standard sets the standard for JavaScript which is used throughout the web browsers available out there.

**Types definitions in JavaScript**

**Dynamic Typing**: The interpreter figures out the type of the variable dynamically based on certain conditions.

**Primitive Data Types**: The primitive data types are the data types which have no methods attached to it i.e. some defined methods cannot be used with them and they are used in isolation. Though there are ways to use those methods by wrapping these primitive data type variables (covered in the next article). The following are the data types that comes under the primitive category:

1. **undefined**:- If variable exists but is not defined the it is categorized under undefined.
2. **null**:- If variable exists but is not explicitly set the it comes under null category.
3. **boolean**:- The Boolean data type is to define if a variable is True or False.
4. **number**:- number is the data type to define a number which can be integer, floating point, double. The only problem here is that we have to allocate a memory equivalent to a double variable every time we define a number.
5. **string**:- This is used to define string values of a character.
6. **symbol**:- This is a special data type which is new in ECMA Script 6. The data type “symbol” is a primitive data type having the quality that values of this type can be used to make object properties that are anonymous.

**Object**

Everything in JavaScript is an object. That is each variable, string, array or any other structure that we know comes under the category of object. Java Script object can be understood by almost every language and are easy to read.

**Creating objects**: – There are 3 ways to create objects:  
**1. Using the Object() Constructor**

|  |
| --- |
| var obj = new Object();  obj.firstName = "Geeks";  obj.middleName = "for";  obj.lastName = "Geeks";  obj.isTeaching = "Javascipt";  obj.greet = function() { console.log("Hi"); } |

**2.Using Object literal**

|  |
| --- |
| var obj = {} //obj is an empty object  obj.firstName = "Geeks";  obj.middleName = "for";  obj.lastName = "Geeks";  obj.isTeaching = "Javascipt";  obj.greet = function() { console.log("Hi"); } |

**3. Directly specifying the values**

|  |
| --- |
| var obj = {      obj.firstName = "Geeks";      obj.middleName = "for";      obj.lastName = "Geeks";      obj.isTeaching = "Javascipt";      obj.greet = function() { console.log("Hi"); }  } |

**Coercion**

typecasting is called coercion in JavaScript.

**Explicit Coercion**

var x = 42;

var explicit = String(x); // explicit is set to "42"

**Implicit Coercion**

var x = 42;

var implicit = x + " "; // interpreter automatically sets implicit as "42

**Hoisting:** – Function definitions are hoisted but not variable declarations. This means that when a function is declared, it is usable from anywhere within your code.

**Example of hoisting**: –

JavaScript only hoists declarations, not the initializations.

|  |
| --- |
| function greet(){      console.log("Hi");  }  greet(); |

greet();

function greet(){

    console.log("Hi");

}

variable hoisting is not possible in JavaScript.

x(); // calling x

var x = function(){

    console.log("Hi");

}

/\* the above is an error as x is not a function because due to the '=' operator,

x is a variable initialization  to which a function is assigned.

---------------------------------------

console.log(x);

var x = 42;

// Output ==> undefined

-----------------------------------------------------------------------

# Strict mode in JavaScript

Strict Mode is a new feature in ECMAScript 5 that allows you to place a program, or a function, in a “strict” operating context. The statement “use strict”; instructs the browser to use the Strict mode, which is a reduced and safer feature set of JavaScript.

Strict mode can be used in two ways – used in global scope for the entire script and can be applied to individual functions. Strict mode doesn’t work with block statements enclosed in {} braces.

// Whole-script strict mode syntax

function strict() {

// Function-level strict mode syntax

'use strict';

function nested() { return 'Javascript on GeeksforGeeks'; }

return "strict mode function! " + nested();

}

function notStrict() { return "non strict function"; }

'use strict';

var v = "strict mode script!";

# Object Oriented Programming in JavaScript

To be more precise, JavaScript is a prototype based object oriented language, which means it doesn’t have classes rather it define behaviors using constructor function and then reuse it using prototype.

Let’s use ES6 classes then we will look into traditional way of defining Object and simulate them as classes.

Traditional Way.

|  |
| --- |
| // Defining class in a Traditional Way.  function Employee(Ename, Eid) {      this.Ename = Ename;      this.Eid = Eid;  };    Employee.prototype.getDetails = function(){      return "Employee name = " + this.Ename +              ", Employee id = " + this.Eid;  }    var Emp1 = new Employee("Sumit", "1234");  console.log(Emp1.getDetails()); |

**Output:**

Employee name= Sumit ,Employee id=1234

// Define class using ES6

class Employee {

// Defining connstructor

// to initialize the property

constructor(Ename, Eid) {

this.Ename = Ename;

this.Eid = Eid;

}

// Method returns employee details

getDetails() {

return "Employee name = " + this.Ename +

", Employee id = " + this.Eid;

}

}

// Creating an Employee Object

var Emp1 = new Employee("Sumit", "1234");

// Printing the Employee Details

console.log(Emp1.getDetails());

Output:

Employee name= Sumit ,Employee id=1234

**Encapsulation** – The process of **wrapping property and function**within a **single unit** is known as encapsulation.

Sometimes encapsulation refers to **hiding of data** or **data Abstraction** which means representing essential features hiding the background detail.

class Person {

    constructor(name, id) {

        this.name = name;

        this.id = id;

    }

    add\_Address(add) {

        this.add = add;

    }

    details() {

        return "Name is = " + this.name +

            ", Student id = " + this.id + ", Address = " + this.add;

    }

}

var person = new Person("Sunny", "14783");

person.add\_Address("Delhi");

console.log(person.details());

Output:

Name is = Sunny ,Student id=14783, Address = Delhi

**Inheritance**– It is a concept in which some property and methods of an Object is being used by another Object. Unlike most of the OOP languages where classes inherit classes, JavaScript Object inherits Object i.e. certain features (property and methods)of one object can be reused by other Objects.

# JavaScript | Polyfilling & Transpiling

**Transpiling**

**Transformation** + **Compiling**.

The following is a small list of such tools.

* [Babel](https://babeljs.io/)
* [Traceur](https://github.com/google/traceur-compiler)

# this Identifier

In JavaScript, ‘this’ identifier can be used in different contexts and scopes.

**‘this’ in Global Scope Functional Scope**

// Declaring variable in global context.

var a = "GFG";

console.log(a);

// Using this we refer to the Global Context.

// And update the value of a we declared previously.

this.a = "GeeksforGeeks";

console.log(a);

Output:

GFG

GeeksforGeeks

// Function that contains this.

function myFunc() {

console.log( this.a );

}

var a = "Global";

// Owner of the function.

var myObj1 = {

a: "myObj1",

myFunc: myFunc

};

// Object other than the owner.

var myObj2 = {

a: "myObj2"

};

// Call the funtion in Global Scope.

myFunc();

// Call the funtion from the reference of owner.

myObj1.myFunc();

// Call the function from the reference

// of object other than the owner.

myFunc.call( myObj2 );

// Create a new undefined object.

new myFunc();

Output:

Global

myObj1

myObj2

undefined

**‘this’ Inside an Event Handler**

<div id="clickMe">Welcome to GFG!</div>

function clickedMe() {

console.log(this.innerHTML);

}

clickedMe(); // undefined because global object.

var myElem = document.getElementById('clickMe');

myElem.onclick = clickedMe;

myElem.onclick(); // Welcome to GFG!

Output:

undefined

Welcome to GFG!

# JavaScript | Rest Operator

**Rest Operator** is an improved way to handle function parameter, allowing us to more easily handle various input as parameters in a function.

Rest operator is added in ES2015 or ES6 which improved the ability to handle parameter.

Syntax:

function functionname[...parameters]//... is the rest operator

{

statement;

}

It stores n number of parameters as an array.

Javascript code for display parameter using rest operator

<script>

// Calling of function

function onlyMath(operator, ...numbers) {

document.write(operator +"<br>");

document.write(numbers);

}

onlyMath('Hello',1,2,3,4,5);

</script>

Run on IDE

Output

Hello

1,2,3,4,5

Javascript code demonstrating addition of numbers using rest operator.

|  |
| --- |
| <script>       //function passed with parameters using rest operator.     function total(...args) {     var result = 0;       //args is used as a variable to add numbers.     for(var i = 0; i < args.length; i++) {      result += args[i];     }     return result;       //scope of args ends here.     }     document.write(total(1, 6, 8));    </script> |