## **What is JavaScript?**

JavaScript is a client-side as well as server side scripting language that can be inserted into HTML pages and is understood by web browsers. JavaScript is also an Object based Programming language

## **What are JavaScript Data Types?**

* Number
* String
* Boolean
* Object
* Undefined

## **Which company developed JavaScript?**

Netscape is the software company who developed JavaScript.

## **What are the features of JavaScript?**

Following are the **features** of JavaScript:

* It is a **lightweight, interpreted** programming language.
* It is designed for creating **network-centric** applications.
* It is complementary to and **integrated** with Java.
* It is an **open** and **cross-platform** scripting language.

## 

## 

## **Is JavaScript a case-sensitive language?**

Yes, JavaScript is a **case sensitive** language. The language keywords, variables, function names, and any other identifiers must always be typed with a consistent capitalization of letters.

## **What is the difference between undefined and not defined in JavaScript?**

In JavaScript, if you try to use a variable that doesn't exist and has not been declared, then JavaScript will throw an error var name is not defined and script will stop executing. However, if you use typeof undeclared\_variable, then it will return undefined.

Before getting further into this, let's first understand the difference between declaration and definition.

Let's say var x is a declaration because you have not defined what value it holds yet, but you have declared its existence and the need for memory allocation.

> var x; // declaring x

> console.log(x); //output: undefined

Here var x = 1 is both a declaration and definition (also we can say we are doing an initialisation). In the example above, the declaration and assignment of value happen inline for variable x. In JavaScript, every variable or function declaration you bring to the top of its current scope is called hoisting.

The assignment happens in order, so when we try to access a variable that is declared but not defined yet, we will get the result undefined.

var x; // Declaration

if(typeof x === 'undefined') // Will return true

If a variable that is neither declared nor defined, when we try to reference such a variable we'd get the result not defined.

> console.log(y); // Output: ReferenceError: y is not defined

## **What is a “closure” in JavaScript? Provide an example**

A closure is a function defined inside another function (called the parent function), and has access to variables that are declared and defined in the parent function scope.

The closure has access to variables in three scopes:

* Variables declared in their own scope
* Variables declared in a parent function scope
* Variables declared in the global namespace

var globalVar = "abc";

// Parent self invoking function

(function outerFunction (outerArg) { // begin of scope outerFunction

// Variable declared in outerFunction function scope

var outerFuncVar = 'x';

// Closure self-invoking function

(function innerFunction (innerArg) { // begin of scope innerFunction

// variable declared in innerFunction function scope

var innerFuncVar = "y";

console.log(

"outerArg = " + outerArg + "\n" +

"outerFuncVar = " + outerFuncVar + "\n" +

"innerArg = " + innerArg + "\n" +

"innerFuncVar = " + innerFuncVar + "\n" +

"globalVar = " + globalVar);

}// end of scope innerFunction)(5); // Pass 5 as parameter

}// end of scope outerFunction )(7); // Pass 7 as parameter

innerFunction is closure that is defined inside outerFunction and has access to all variables declared and defined in the outerFunction scope. In addition, the function defined inside another function as a closure will have access to variables declared in the global namespace.

Thus, the output of the code above would be:

outerArg = 7

outerFuncVar = x

innerArg = 5

innerFuncVar = y

globalVar = abc

## **What will be the output of the following code?**

var output = (function(x){

delete x;

return x; })(0);

console.log(output);

The output would be 0. The delete operator is used to delete properties from an object. Here x is not an object but a **local variable**. delete operators don't affect local variables.

## **What are the advantages of JavaScript?**

Following are the **advantages** of using JavaScript −

* **Less server interaction** − You can validate user input before sending the page off to the server. This saves server traffic, which means less load on your server.
* **Immediate feedback to the visitors** − They don’t have to wait for a page reload to see if they have forgotten to enter something.
* **Increased interactivity** − You can create interfaces that react when the user hovers over them with a mouse or activates them via the keyboard.
* **Richer interfaces** − You can use JavaScript to include such items as drag-and-drop components and sliders to give a Rich Interface to your site visitors.

## **How can you create an object in JavaScript?**

JavaScript supports **Object** concept very well. You can create an object using the **object literal** as follows −

1

2

3

4

var emp = {

name: "Daniel",

age: 23

};

## **How can you create an Array in JavaScript?**

You can define arrays using the **array literal** as follows-

1

2

var x = [];

var y = [1, 2, 3, 4, 5];

## **What is the difference between Java & JavaScript?**

**Java** **JavaScript** Java is an OOP programming language. JavaScript is an OOP scripting language. It creates applications that run in a virtual machine or browser. The code is run on a browser only. Java code needs to be compiled. JavaScript code are all in the form of text.

## **What are the scopes of a variable in JavaScript?**

The scope of a variable is the **region** of your program in which it is **defined**. JavaScript variable will have only two scopes.

• **Global Variables** − A global variable has global scope which means it is visible everywhere in your JavaScript code.

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

## **What is the purpose of ‘This’ operator in JavaScript?**

The JavaScript **this** keyword refers to the object it belongs to. This has different values depending on where it is used. In a method, this refers to the owner object and in a function, this refers to the global object.

## **How does TypeOf Operator work?**

The **typeof** operator is used to get the data type of its operand. The operand can be either a **literal** or a **data structure** such as a variable, a function, or an object. It is a **unary** operator that is placed before its single operand, which can be of any type. Its value is a string indicating the data type of the operand.

## **What is the difference between Attributes and Property?**

**Attributes-** provide more details on an element like id, type, value etc.

**Property-** is the value assigned to the property like type=”text”, value=’Name’ etc.

## **List out the different ways an HTML element can be accessed in a JavaScript code.**

Here are the list of ways an HTML element can be accessed in a Javascript code:

(i) **getElementById(‘idname’):** Gets an element by its ID name

(ii) **getElementsByClass(‘classname’):** Gets all the elements that have the given classname.

(iii) **getElementsByTagName(‘tagname’):** Gets all the elements that have the given tag name.

(iv) **querySelector():** This function takes css style selector and returns the first selected element.

## **In how many ways a JavaScript code can be involved in an HTML file?**

There are 3 different ways in which a JavaScript code can be involved in an HTML file:

* **Inline**
* **Internal**
* **External**

An **inline** function is a JavaScript function, which is assigned to a variable created at runtime. You can differentiate between Inline Functions and Anonymous since an inline function is assigned to a variable and can be easily reused. When you need a JavaScript for a function, you can either have the script **integrated** in the page you are working on, or you can have it placed in a **separate** file that you call, when needed. This is the difference between an **internal** script and an **external** script.

## **What are the ways to define a variable in JavaScript?**

The three possible ways of defining a variable in JavaScript are:

* **Var** – The JavaScript variables statement is used to declare a variable and, optionally, we can initialize the value of that variable. Example: var a =10; Variable declarations are processed before the execution of the code.
* **Const** – The idea of const functions is not allow them to modify the object on which they are called. When a function is declared as const, it can be called on any type of object.
* **Let** – It is a signal that the variable may be reassigned, such as a counter in a loop, or a value swap in an algorithm. It also signals that the variable will be used only in the block it’s defined in.

## **What is the difference between the operators ‘==‘ & ‘===‘?**

The main difference between “==” and “===” operator is that formerly compares variable by making **type correction** e.g. if you compare a number with a string with numeric literal, == allows that, but === doesn’t allow that, because it not only checks the value but also type of two variable, if two variables are not of the same type “===” return false, while “==” return true.

## **What is the difference between null & undefined?**

Undefined means a variable has been **declared** but has not yet been **assigned** a value. On the other hand, null is an assignment value. It can be assigned to a variable as a representation of no value. Also, undefined and null are two distinct types: undefined is a type itself (undefined) while null is an object.

## **Name some of the JavaScript Frameworks**

A [JavaScript framework](https://www.edureka.co/blog/top-10-javascript-frameworks/) is an application framework written in JavaScript. It differs from a JavaScript library in its control flow. There are many JavaScript Frameworks available but some of the most commonly used frameworks are:

* [Angular](https://www.edureka.co/angular-training)
* [React](https://www.edureka.co/blog/reactjs-tutorial)
* Vue

## **How can you remove an element from an array? Or can you use delete to remove an element in an array or not?**

First, we should not use delete to remove an element from an array. Because, delete removes only the value of the array element, not the entire element. So the index still have undefined. And the length does not change. delete is supposed to use on an Object to remove its property.

The best way to remove an element is using splice method. The syntax is:

arr.splice(startIndex, deleteCount, itemToInser);

and

[1, 2, 3].splice(1, 1);

result in [1, 3].

## **What is instanceof in JavaScript?**

This operator tests whether an object has the prototype property of a given constructor in its prototype chain. For example,

function Car(name)

{

this.name = name;

}

var mycar = new Car('Honda');

mycar instanceof Car; //true

mycar instanceof Car; returns true since the mycar is constructed from the Car.

## **How can you get the list of all properties in an Object?**

The easy way is using Object.keys(). This will return an array of given object’s own enumerable properties.

If we want all properties, even not-enumerable properties also, we can use Object.getOwnPropertyNames().

## **What is shift() and push? How they are differing from each other?**

”The shift() removes the first element from an array and return it.’’ On the otherhand, unshift() adds the given element at the start of an array.” The push() adds the given element at the end of the array. “And, pop() removes the last element from an array.”

## **What is Number and String Objects in JavaScript?**

JavaScript have Number and String objects to work with the primitive types number and string. We don’t need to explicitly create and use Number and String objects. JavaScript uses them to convert the primitive types to object when we are accessing any properties or methods on the primitive types. For example,

var str = 'Javascript';

str.length //10

Here, we are declaring a variable str and assign a primitive string to that. When accessing the length property, the primitive types actually does not have the length property, but JavaScript compiler convert this primitive string to Object string, and the return the length property of it. To say more clearly, JavaScript returns new String(str).length. And this String object will be removed immediately from the memory once it returns the required value. So we won’t see any difference. Remember, declaring a string as var str = ‘Javascript’; is different from declaring as String object as

var str = new String('Javascript');. The latter will return object.

var str = new String('Javascript');

typeof str; //object

## **What is DOM**

Document Object Model .

## **What is prototype in javascript?**

Every object has prototype in javascript, so you can add the property to an object based on prototype. You can create the instant.

## **What is undefined?**

In javascript, undefined means, variable is declared but doesn’t contain value.

## **What is not defined?**

Varaible not yet declared.

## **What is closure?**

variable have first its own scope, parent scope and global sope.

## **What is the difference for let and const?**

let will scope for block level and const values never change.

## **What is variable hoisting?**

All the variable will go the top of the function and declare the variable.

## **How to create an Object?**

Object.Create();

## **What is a prompt box in JavaScript?**

A prompt box is a box which allows the user to pass the input in provided text area ,prompt displays dialog box which will be contain “OK” and “Cancel” to process further after entering the input value.

## **Explain equality in JavaScript**

JavaScript has both strict and type–converting comparisons:

* **Strict comparison (e.g., ===)** checks for value equality without allowing *coercion*
* **Abstract comparison (e.g. ==)** checks for value equality with *coercion* allowed

var a = "42";

var b = 42;

a == b; // true

a === b; // false

Some simple equalityrules:

* If either value (aka side) in a comparison could be the true or false value, avoid == and use ===.
* If either value in a comparison could be of these specific values (0, "", or [] -- empty array), avoid == and use ===.
* In all other cases, you're safe to use ==. Not only is it safe, but in many cases it simplifies your code in a way that improves readability.

## **What is IIFEs (Immediately Invoked Function Expressions)?**

It’s an Immediately-Invoked Function Expression, or IIFE for short. It executes immediately after it’s created:

(function IIFE(){

console.log( "Hello!" );

})();

// "Hello!"

This pattern is often used when trying to avoid polluting the global namespace, because all the variables used inside the IIFE (like in any other normal function) are not visible outside its scope.

## **When should I use Arrow functions in ES6?**

I'm now using the following rule of thumb for functions in ES6 and beyond:

* Use function in the global scope and for Object.prototype properties.
* Use class for object constructors.
* Use => everywhere else.

Why use arrow functions almost everywhere?

* **Scope safety**: When arrow functions are used consistently, everything is guaranteed to use the same thisObject as the root. If even a single standard function callback is mixed in with a bunch of arrow functions there's a chance the scope will become messed up.
* **Compactness**: Arrow functions are easier to read and write. (This may seem opinionated so I will give a few examples further on).
* **Clarity**: When almost everything is an arrow function, any regular function immediately sticks out for defining the scope. A developer can always look up the next-higher function statement to see what the thisObject is.

## **What are the differences between ES6 class and ES5 function constructors?**

// ES5 Function Constructor

function Person(name) {

this.name = name;

}

// ES6 Class

class Person {

constructor(name) {

this.name = name;

}

}

For simple constructors, they look pretty similar.

The main difference in the constructor comes when using inheritance.

## **Explain the difference between Object.freeze() vs const**

const and Object.freeze are two completely different things.

* const applies to **bindings** ("variables"). It creates an immutable binding, i.e. you cannot assign a new value to the binding
* Object.freeze works on **values**, and more specifically, *object values*. It makes an object immutable, i.e. you cannot change its properties.

## **How do you check if a variable is an object**

You can use typeof to determine if variable is an object, however bear in mind that null is actually an object! However null object is 'falsy' thus the following will work:

if(bar && typeof bar === "object") {

console.log('bar is object and is not null');

}

## **Explain hoisting in JavaScript.**

As some might not be familiar with the term 'hoisting' yet have the relevant experience this question could be asked indirectly

In JavaScript function declarations ( function foo() {} ) and variable declarations ( var bar ) are 'hoisted' i.e. are silently moved to the very top of the scope. Consider the following code:

(function() {

console.log(bar); //returns 'undefined'

//console.log(baz) // error: baz is not defined

foo(); // outputs 'aloha' to the console

//function declaration AND its body is hoisted

function foo() {

console.log('aloha');

}

//variable declaration is hoisted but value assignment stays here

var bar = 1;

baz = 2; //defines baz in global scope

})();

## **Explain prototypal/differential inheritance**

Conceptually this is very simple: A new object can inherit properties of an old object.

While JavaScript has always been a prototype-oriented language, tools to work with prototypes were somewhat missing.

Object.create

used in the code snipped above has been added in ECMAScript 5

## **What is Strict Mode in JavaScript**

Strict Mode has been introduced as part of ECMAScript 5 and introduces new, restricted variant of JavaScript which has following aims:

* Throws errors for actions that are rather silly but previously didn't throw an error
* Throws errors for potentially unsafe actions
* Disables functions that are poorly thought out
* Potentially code in strict mode could run faster by eliminating mistakes that would make it difficult for JavaScript engines to perform optimizations

## **What is a Callback function?**

A callback function is a function passed into another function as an argument, which is then invoked inside the outer function to complete some kind of routine or action.

function greeting(name) {

alert('Hello ' + name);

}

function processUserInput(callback) {

var name = prompt('Please enter your name.');

callback(name);

}

processUserInput(greeting);