**1. Primitive types**

String number,number bigInt,Boolean,undefied , null, symbol

**2. Non-primitive types**

* Primitive data types can store only a single value. To store multiple and complex values, non-primitive data types are used.
* Object - Used to store collection of data.
* **Note- It is important to remember that any data type that is not a primitive data type, is of Object type in javascript.**

### 2. Explain Hoisting in javascript.

Hoisting is the default behaviour of javascript where all the variable and function declarations are moved on top.

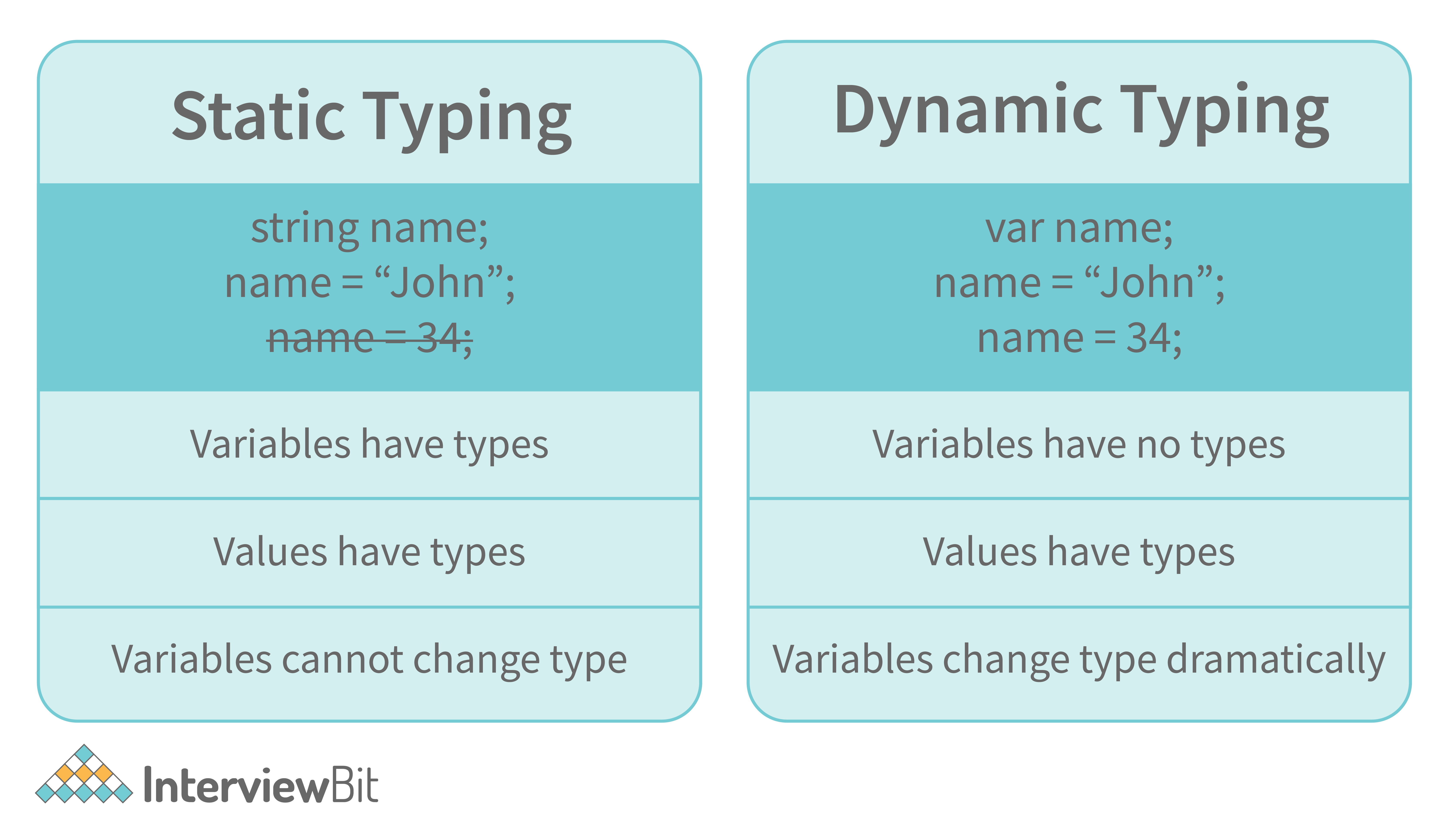
### 6. Explain Implicit Type Coercion in javascript.

Implicit type coercion in javascript is the automatic conversion of value from one data type to another. It takes place when the operands of an expression are of different data types.

String, Boolean , Equality Coercion

### 7. Is javascript a statically typed or a dynamically typed language?

JavaScript is a dynamically typed language. In a dynamically typed language, the type of a variable is checked during **run-time**in contrast to a statically typed language, where the type of a variable is checked during **compile-time.**



Since javascript is a loosely(dynamically) typed language, variab

### 8. What is NaN property in JavaScript?

NaN property represents the **“Not-a-Number”**value. It indicates a value that is not a legal number.

**typeof**of NaN will return a **Number**.

To check if a value is NaN, we use the **isNaN()**function,

#### Note- isNaN() function converts the given value to a Number type, and then equates to NaN.

isNaN("Hello") // Returns true

isNaN(345) // Returns false

isNaN('1') // Returns false, since '1' is converted to Number type which results in 0 ( a number)

isNaN(true) // Returns false, since true converted to Number type results in 1 ( a number)

isNaN(false) // Returns false

isNaN(undefined) // Returns true

### 10. What is an Immediately Invoked Function in JavaScript?(  Self Invoking Functions

### )

(()=>{})()

**An Immediately Invoked Function ( known as IIFE and pronounced as IIFY) is a function that runs as soon as it is defined.**

The benefit of self-invoking functions is that they enable us to execute code once without cluttering the global namespace (without declaring any globals).

Strictmode:

“use strict”

Variable need to be declared

Delete is not allowed

"use strict";  
let x = 3.14;  
delete x;                // This will cause an error

Duplicating a parameter name is not allowed:

Octal numeric literals are not allowed:

"use strict";  
let x = 010;

Writing to a read-only property is not allowed:

Const test={k:”1”}

Test[‘k’] =3//not allowed

The word eval cannot be used as a variable:

The word arguments cannot be used as a variable:

For security reasons, eval() is not allowed to create variables in the scope from which it was called.

The "use strict" directive is only recognized at the **beginning** of a script or a function.

aracteristics of strict mode in javascript

1. Duplicate arguments are not allowed by developers.
2. In strict mode, you won't be able to use the JavaScript keyword as a parameter or function name.
3. The 'use strict' keyword is used to define strict mode at the start of the script. Strict mode is supported by all browsers.
4. Engineers will not be allowed to create global variables in 'Strict Mode.

Higher order functions:

**function** **higherOrder**(fn) {

fn();

}

higherOrder(**function**() { console.log("Hello world") });

**function** **higherOrder2**() {

**return** **function**() {

**return** "Do something";

}

}

**var** x = higherOrder2();

x() // Returns "Do something

**The “this” keyword refers to the object that the function is a property of.**

### 15. Explain call(), apply() and, bind() methods.

**1. call():**

* It’s a predefined method in javascript.
* This method invokes a method (function) by specifying the owner object.
* Example 1:

**function** **sayHello**(){

**return** "Hello " + this.name;

}

**var** obj = {name: "Sandy"};

sayHello.call(obj);

// Returns "Hello Sandy"

* call() method allows an object to use the method (function) of another object.
* Example 2:

**var** person = {

age: 23,

getAge: **function**(){

**return** this.age;

}

}

**var** person2 = {age: 54};

person.getAge.call(person2);

// Returns 54

* call() accepts arguments:

**function** **saySomething**(message){

**return** this.name + " is " + message;

}

**var** person4 = {name: "John"};

saySomething.call(person4, "awesome");

// Returns "John is awesome"

**apply()**  
  
The apply method is similar to the call() method. The only difference is that,  
  
**call() method takes arguments separately whereas, apply() method takes arguments as an array.**

**function** **saySomething**(message){

**return** this.name + " is " + message;

}

**var** person4 = {name: "John"};

saySomething.apply(person4, ["awesome"]);

**2. bind():**

* This method returns a new function, where the value of **“this”**keyword will be bound to the owner object, which is provided as a parameter.
* Example with arguments:

**var** bikeDetails = {

displayDetails: **function**(registrationNumber,brandName){

**return** this.name+ " , "+ "bike details: "+ registrationNumber + " , " + brandName;

}

}

**var** person1 = {name: "Vivek"};

**var** detailsOfPerson1 = bikeDetails.displayDetails.bind(person1, "TS0122", "Bullet");

// Binds the displayDetails function to the person1 object

detailsOfPerson1();

//Returns Vivek, bike details: TS0122, Bullet

* We'll use **exec ()** to search a string for a specific pattern, and if it finds it, it'll return the pattern directly; else, it'll return an 'empty' result.
* We will use a**test ()** to find a string for a specific pattern. It will return the Boolean value 'true' on finding the given text otherwise, it will return 'false'.

### 17. What is currying in JavaScript?

**Currying is an advanced technique to transform a function of arguments n, to n functions of one or fewer arguments.**

Example of a curried function:

**function** **add** (a) {

**return** **function**(b){

**return** a + b;

}

}

add(3)(4)

For Example, if we have a function **f(a,b)**, then the function after currying, will be transformed to **f(a)(b).**  
  
By using the currying technique, we do not change the functionality of a function, we just change the way it is invoked.  
  
Let’s see currying in action:

**function** **multiply**(a,b){

**return** a\*b;

}

**function** **currying**(fn){

**return** **function**(a){

**return** **function**(b){

**return** fn(a,b);

}

}

}

**var** curriedMultiply = currying(multiply);

multiply(4, 3); // Returns 12

curriedMultiply(4)(3); // Also returns 12

As one can see in the code above, we have transformed the function **multiply(a,b)**to a function **curriedMultiply**, which takes in one parameter at a time.

### 19. Explain Scope and Scope Chain in javascript.

Scope in JS determines the accessibility of variables and functions at various parts of one’s code.  
  
In general terms, the scope will let us know at a given part of code, what are variables and functions we can or cannot access.  
  
There are three types of scopes in JS:

* Global Scope
* Local or Function Scope
* Block Scope

**Block Scope:**Block scope is related to the variables declared using let and const. Variables declared with var do not have block scope. Block scope tells us that any variable declared inside a block { }, can be accessed only inside that block and cannot be accessed outside of it.

Closures are an ability of a function to remember the variables and functions that are declared in its outer scope.

**var** Person = **function**(pName){

**var** name = pName;

this.getName = **function**(){

**return** name;

}

}

**var** person = **new** Person("Neelesh");

console.log(person.getName());

Let’s understand closures by example:

**function** **randomFunc**(){

**var** obj1 = {name:"Vivian", age:45};

**return** **function**(){

console.log(obj1.name + " is "+ "awesome"); // Has access to obj1 even when the randomFunc function is executed

}

}

**var** initialiseClosure = randomFunc(); // Returns a function

initialiseClosure();

**This ability of a function to store a variable for further reference even after it is executed is called Closure.**

There are many advantages of javascript. Some of them are

1. Javascript is executed on the client-side as well as server-side also. There are a variety of Frontend Frameworks that you may study and utilize. However, if you want to use JavaScript on the backend, you'll need to learn NodeJS. It is currently the only JavaScript framework that may be used on the backend.
2. Javascript is a simple language to learn.
3. Web pages now have more functionality because of Javascript.
4. To the end-user, Javascript is quite quick

### 22. What are object prototypes?

All javascript objects inherit properties from a prototype. For example,

* Date objects inherit properties from the Date prototype
* Math objects inherit properties from the Math prototype
* Array objects inherit properties from the Array prototype.
* On top of the chain is **Object.prototype.**Every prototype inherits properties and methods from the Object.prototype.
* **A prototype is a blueprint of an object. The prototype** allows us to use properties and methods on an object even if the properties and methods do not exist on the current object.

Let’s see prototypes help us use methods and properties:

### 23. What are callbacks?

A callback is a function that will be executed after another function gets executed. In javascript, functions are treated as first-class citizens, they can be used as an argument of another function, can be returned by another function, and can be used as a property of an object.

**Functions that are used as an argument to another function are called callback functions.**Example:

**function** **divideByHalf**(sum){

console.log(Math.floor(sum / 2));

}

**function** **multiplyBy2**(sum){

console.log(sum \* 2);

}

**function** **operationOnSum**(num1,num2,operation){

**var** sum = num1 + num2;

operation(sum);

}

operationOnSum(3, 3, divideByHalf); // Outputs 3

memoized:

**function** **memoizedAddTo256**(){

**var** cache = {};

**return** **function**(num){

**if**(num **in** cache){

console.log("cached value");

**return** cache[num]

}

**else**{

cache[num] = num + 256;

**return** cache[num];

}

}

}

**var** memoizedFunc = memoizedAddTo256();

memoizedFunc(20); // Normal return

memoizedFunc(20); // Cached return

### 27. What is the use of a constructor function in javascript?

Constructor functions are used to create objects in javascript.

When do we use constructor functions?

If we want to create multiple objects having similar properties and methods, constructor functions are used.

#### ****Note- The name of a constructor function should always be written in Pascal Notation: every word should start with a capital letter.****

Example:

**function** **Person**(name,age,gender){

this.name = name;

this.age = age;

this.gender = gender;

}

**var** person1 = **new** Person("Vivek", 76, "male");

console.log(person1);

**var** person2 = **new** Person("Courtney", 34, "female");

console.log(person2);

In the code above, we have created a constructor function named Person. Whenever we want to create a new object of the type Person, We need to create it using the new keyword:

**var** person3 = **new** Person("Lilly", 17, "female");

The above line of code will create a new object of the type Person. Constructor functions allow us to group similar objects

### 28. What is DOM?

* DOM stands for Document Object Model.  DOM is a programming interface for HTML and XML documents.
* When the browser tries to render an HTML document, it creates an object based on the HTML document called DOM. Using this DOM, we can manipulate or change various elements inside the HTML document.
* Example of how HTML code gets converted to DOM:

### 29. Which method is used to retrieve a character from a certain index?

The charAt() function of the JavaScript string finds a char element at the supplied index. The index number begins at 0 and continues up to n-1, Here n is the string length. The index value must be positive, higher than, or the same as the string length.

### 30. What do you mean by BOM?

Browser Object Model is known as BOM. It allows users to interact with the browser. A browser's initial object is a window. As a result, you may call all of the window's functions directly or by referencing the window. The document, history, screen, navigator, location, and other attributes are available in the window object.

### 31. What is the distinction between client-side and server-side JavaScript?

Client-side JavaScript is made up of two parts, a fundamental language and predefined objects for performing JavaScript in a browser. JavaScript for the client is automatically included in the HTML pages. At runtime, the browser understands this script.

### 32. What are arrow functions?

Arrow functions were introduced in the ES6 version of javascript. They provide us with a new and shorter syntax for declaring functions. Arrow functions can only be used as a function expression.  
  
Let’s compare the normal function declaration and the arrow function declaration in detail:

// Traditional Function Expression

**var** add = **function**(a,b){

**return** a + b;

}

// Arrow Function Expression

**var** arrowAdd = (a,b) => a + b;

Arrow functions are declared without the function keyword. If there is only one returning expression then we don’t need to use the return keyword as well in an arrow function as shown in the example above. Also, for functions having just one line of code, curly braces { } can be omitted.

// Traditional function expression

**var** multiplyBy2 = **function**(num){

**return** num \* 2;

}

// Arrow function expression

**var** arrowMultiplyBy2 = num => num \* 2;

If the function takes in only one argument, then the parenthesis () around the parameter can be omitted as shown in the code above.

**var** obj1 = {

valueOfThis: **function**(){

**return** this;

}

}

**var** obj2 = {

valueOfThis: ()=>{

**return** this;

}

}

obj1.valueOfThis(); // Will return the object obj1

obj2.valueOfThis(); // Will return window/global object

The biggest difference between the traditional function expression and the arrow function is the handling of **this**keyword. By general definition, **this**keyword always refers to the object that is calling the function. As you can see in the code above, **obj1.valueOfThis()**returns obj1 since **this**keyword refers to the object calling the function.

In the arrow functions, there is no binding of **this**keyword. Thiskeyword inside an arrow function does not refer to the object calling it. It rather inherits its value from the parent scope which is the window object in this case. Therefore, in the code above, **obj2.valueOfThis()**returns the window object.

### 34. Differences between declaring variables using var, let and const.

Before the ES6 version of javascript, only the keyword var was used to declare variables. With the ES6 Version, keywords let and const were introduced to declare variables.

|  |  |  |  |
| --- | --- | --- | --- |
| keyword | const | let | var |
| global scope | no | no | yes |
| function scope | yes | yes | yes |
| block scope | yes | yes | no |
| can be reassigned | no | yes | yes |

### 35. What is the rest parameter and spread operator?

Both rest parameter and spread operator were introduced in the ES6 version of javascript.  
  
**Rest parameter ( … ):**

* It provides an improved way of handling the parameters of a function.
* Using the rest parameter syntax, we can create functions that can take a variable number of arguments.
* Any number of arguments will be converted into an array using the rest parameter.
* It also helps in extracting all or some parts of the arguments.
* Rest parameters can be used by applying three dots (...) before the parameters.

**function** **extractingArgs**(...args){

**return** args[1];

}

// extractingArgs(8,9,1); // Returns 9

**function** **addAllArgs**(...args){

**let** sumOfArgs = 0;

**let** i = 0;

**while**(i < args.length){

sumOfArgs += args[i];

i++;

}

**return** sumOfArgs;

}

addAllArgs(6, 5, 7, 99); // Returns 117

addAllArgs(1, 3, 4); // Returns 8

**\*Note- Rest parameter should always be used at the last parameter of a function:**

// Incorrect way to use rest parameter

**function** **randomFunc**(a,...args,c){

//Do something

}

// Correct way to use rest parameter

**function** **randomFunc2**(a,b,...args){

//Do something

}

* **Spread operator (…):**Although the syntax of the spread operator is exactly the same as the rest parameter, the spread operator is used to spreading an array, and object literals. We also use spread operators where one or more arguments are expected in a function call.

**function** **addFourNumbers**(num1,num2,num3,num4){

**return** num1 + num2 + num3 + num4;

}

**let** fourNumbers = [5, 6, 7, 8];

addFourNumbers(...fourNumbers);

// Spreads [5,6,7,8] as 5,6,7,8

**let** array1 = [3, 4, 5, 6];

**let** clonedArray1 = [...array1];

// Spreads the array into 3,4,5,6

console.log(clonedArray1); // Outputs [3,4,5,6]

**let** obj1 = {x:'Hello', y:'Bye'};

**let** clonedObj1 = {...obj1}; // Spreads and clones obj1

console.log(obj1);

**let** obj2 = {z:'Yes', a:'No'};

**let** mergedObj = {...obj1, ...obj2}; // Spreads both the objects and merges it

console.log(mergedObj);

// Outputs {x:'Hello', y:'Bye',z:'Yes',a:'No'};

\*\*\*Note- Key differences between rest parameter and spread operator:

* Rest parameter is used to take a variable number of arguments and turns them into an array while the spread operator takes an array or an object and spreads it
* Rest parameter is used in function declaration whereas the spread operator is used in function calls
* Object.
* using Class.

# create Method.( Object.create())

* Object Literals.{}
* using Function.
* Object Constructor.

### 38. What are classes in javascript?

Introduced in the ES6 version, classes are nothing but syntactic sugars for constructor functions. They provide a new way of declaring constructor functions in javascript.  Below are the examples of how classes are declared and used:

// Before ES6 version, using constructor functions

**function** **Student**(name,rollNumber,grade,section){

this.name = name;

this.rollNumber = rollNumber;

this.grade = grade;

this.section = section;

}

// Way to add methods to a constructor function

Student.prototype.getDetails = **function**(){

**return** 'Name: ${this.name}, Roll no: ${this.rollNumber}, Grade: ${this.grade}, Section:${this.section}';

}

**let** student1 = **new** Student("Vivek", 354, "6th", "A");

student1.getDetails();

// Returns Name: Vivek, Roll no:354, Grade: 6th, Section:A

// ES6 version classes

**class** **Student**{

**constructor**(name,rollNumber,grade,section){

this.name = name;

this.rollNumber = rollNumber;

this.grade = grade;

this.section = section;

}

// Methods can be directly added inside the class

**getDetails**(){

**return** 'Name: ${this.name}, Roll no: ${this.rollNumber}, Grade:${this.grade}, Section:${this.section}';

}

}

**let** student2 = **new** Student("Garry", 673, "7th", "C");

student2.getDetails();

// Returns Name: Garry, Roll no:673, Grade: 7th, Section:C

Key points to remember about classes:

* Unlike functions, classes are not hoisted. A class cannot be used before it is declared.
* A class can inherit properties and methods from other classes by using the extend keyword.
* All the syntaxes inside the class must follow the strict mode(‘use strict’) of javascript. An error will be thrown if the strict mode rules are not followed.

Sets:

|  |  |
| --- | --- |
| **Method** | **Description** |
| new Set() | Creates a new Set |
| add() | Adds a new element to the Set |
| delete() | Removes an element from a Set |
| has() | Returns true if a value exists |
| clear() | Removes all elements from a Set |
| forEach() | Invokes a callback for each element |
| values() | Returns an Iterator with all the values in a Set |
| keys() | Same as values() |
| entries() | Returns an Iterator with the [value,value] pairs from a Set |

### 0. Explain WeakSet in javascript.

In javascript, a Set is a collection of unique and ordered elements. Just like Set, WeakSet is also a collection of unique and ordered elements with some key differences:

* Weakset contains only objects and no other type.
* An object inside the weakset is referenced weakly. This means, that if the object inside the weakset does not have a reference, it will be garbage collected.
* Unlike Set, WeakSet only has three methods, **add()**, **delete()**and **has()**.

**const** newSet = **new** Set([4, 5, 6, 7]);

console.log(newSet);// Outputs Set {4,5,6,7}

**const** newSet2 = **new** WeakSet([3, 4, 5]); //Throws an error

**let** obj1 = {message:"Hello world"};

**const** newSet3 = **new** WeakSet([obj1]);

console.log(newSet3.has(obj1)); // true

### 41. Why do we use callbacks?

A callback function is a method that is sent as an input to another function (now let us name this other function "thisFunction"), and it is performed inside the thisFunction after the function has completed execution.

JavaScript is a scripting language that is based on events. Instead of waiting for a reply before continuing, JavaScript will continue to run while monitoring for additional events. Callbacks are a technique of ensuring that a particular code does not run until another code has completed its execution.

### 42. Explain WeakMap in javascript.

In javascript, Map is used to store key-value pairs. The key-value pairs can be of both primitive and non-primitive types. WeakMap is similar to Map with key differences:

* The keys and values in weakmap should always be an object.
* If there are no references to the object, the object will be garbage collected.

**const** map1 = **new** Map();

map1.set('Value', 1);

**const** map2 = **new** WeakMap();

map2.set('Value', 2.3); // Throws an error

**let** obj = {name:"Vivek"};

**const** map3 = **new** WeakMap();

map3.set(obj, {age:23});

### 43. What is Object Destructuring?

Object destructuring is a new way to extract elements from an object or an array.

* **Object destructuring:**Before ES6 version:

**const** classDetails = {

strength: 78,

benches: 39,

blackBoard:1

}

**const** classStrength = classDetails.strength;

**const** classBenches = classDetails.benches;

**const** classBlackBoard = classDetails.blackBoard;

The same example using object destructuring:

**const** classDetails = {

strength: 78,

benches: 39,

blackBoard:1

}

**const** {strength:classStrength, benches:classBenches,blackBoard:classBlackBoard} = classDetails;

console.log(classStrength); // Outputs 78

console.log(classBenches); // Outputs 39

console.log(classBlackBoard); // Outputs 1

As one can see, using object destructuring we have extracted all the elements inside an object in one line of code. If we want our new variable to have the same name as the property of an object we can remove the colon:

**const** {strength:strength} = classDetails;

// The above line of code can be written as:

**const** {strength} = classDetails;

* **Array destructuring:**Before ES6 version:

**const** arr = [1, 2, 3, 4];

**const** first = arr[0];

**const** second = arr[1];

**const** third = arr[2];

**const** fourth = arr[3];

The same example using object destructuring:

**const** arr = [1, 2, 3, 4];

**const** [first,second,third,fourth] = arr;

console.log(first); // Outputs 1

console.log(second); // Outputs 2

console.log(third); // Outputs 3

console.log(fourth); // Outputs 4

### 44. Difference between prototypal and classical inheritance

Programers build objects, which are representations of real-time entities, in traditional OO programming. Classes and objects are the two sorts of abstractions. A class is a generalization of an object, whereas an object is an abstraction of an actual thing. A Vehicle, for example, is a specialization of a Car. As a result, automobiles (class) are descended from vehicles (object).

Classical inheritance differs from prototypal inheritance in that classical inheritance is confined to classes that inherit from those remaining classes, but prototypal inheritance allows any object to be cloned via an object linking method. Despite going into too many specifics, a prototype essentially serves as a template for those other objects, whether they extend the parent object or not.

### 45. What is a Temporal Dead Zone?

Temporal Dead Zone is a behaviour that occurs with variables declared using **let**and **const**keywords. It is a behaviour where we try to access a variable before it is initialized. Examples of temporal dead zone:

x = 23; // Gives reference error

**let** x;

**function** **anotherRandomFunc**(){

message = "Hello"; // Throws a reference error

**let** message;

}

anotherRandomFunc();

In the code above, both in the global scope and functional scope, we are trying to access variables that have not been declared yet. This is called the **Temporal Dead Zone**.

### 46. What do you mean by JavaScript Design Patterns?

JavaScript design patterns are repeatable approaches for errors that arise sometimes when building JavaScript browser applications. They truly assist us in making our code more stable.

They are divided mainly into 3 categories

1. Creational Design Pattern
2. Structural Design Pattern
3. Behavioral Design Pattern.

* **Creational Design Pattern:**The object generation mechanism is addressed by the JavaScript Creational Design Pattern. They aim to make items that are appropriate for a certain scenario.
* **Structural Design Pattern:**The JavaScript Structural Design Pattern explains how the classes and objects we've generated so far can be combined to construct bigger frameworks. This pattern makes it easier to create relationships between items by defining a straightforward way to do so.
* **Behavioral Design Pattern:**This design pattern highlights typical patterns of communication between objects in JavaScript. As a result, the communication may be carried out with greater freedom.

### 47. Is JavaScript a pass-by-reference or pass-by-value language?

The variable's data is always a reference for objects, hence it's always pass by value. As a result, if you supply an object and alter its members inside the method, the changes continue outside of it. It appears to be pass by reference in this case. However, if you modify the values of the object variable, the change will not last, demonstrating that it is indeed passed by value.

### 49. What are the primitive data types in JavaScript?

A primitive is a data type that isn't composed of other data types. It's only capable of displaying one value at a time. By definition, every primitive is a built-in data type (the compiler must be knowledgeable of them) nevertheless, not all built-in datasets are primitives. In JavaScript, there are 5 different forms of basic data. The following values are available:

1. Boolean
2. Undefined
3. Null
4. Number
5. String
6. <script src="demo\_defer.js" defer></script>
7. [Try it Yourself »](https://www.w3schools.com/tags/tryit.asp?filename=tryhtml_script_defer)

## **Definition and Usage**

The defer attribute is a boolean attribute.

If the defer attribute is set, it specifies that the script is downloaded in parallel to parsing the page, and executed after the page has finished parsing.

**Note:** The defer attribute is only for external scripts (should only be used if the src attribute is present).

**Note:** There are several ways an external script can be executed:

* If async is present: The script is downloaded in parallel to parsing the page, and executed as soon as it is available (before parsing completes)
* If defer is present (and not async): The script is downloaded in parallel to parsing the page, and executed after the page has finished parsing
* If neither async or defer is present: The script is downloaded and executed immediately, blocking parsing until the script is completed

### 50. What is the role of deferred scripts in JavaScript?

The processing of HTML code while the page loads are disabled by nature till the script hasn't halted. Your page will be affected if your network is a bit slow, or if the script is very hefty. When you use Deferred, the script waits for the HTML parser to finish before executing it. This reduces the time it takes for web pages to load, allowing them to appear more quickly

### 51. What has to be done in order to put Lexical Scoping into practice?

To support lexical scoping, a JavaScript function object's internal state must include not just the function's code but also a reference to the current scope chain.

### 52. What is the purpose of the following JavaScript code?

**var** scope = "global scope";

**function** **check**()

{

**var** scope = "local scope";

**function** **f**()

{

**return** scope;

}

**return** f;

}

Every executing function, code block, and script as a whole in JavaScript has a related object known as the Lexical Environment. The preceding code line returns the value in scope.