IMPORTANT

JAVASCRIPT CONCEPTS

FOR INTERVIEWS

1 NEW FEATURES IN ES6 VERSION.

The new features introduced in ES6 version of JavaScript are:

- Let and const keywords.
- Arrow functions.
- Multi-line Strings.
- The destructuring assignment.
- Enhanced object literals.
- Promises.

2 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.

Static Typing	Dynamic Typing
Variables have types	Variables have no types
Variables cannot change type	Variables can change type

3 EXPLAIN SCOPE AND SCOPE CHAIN JAVASCRIPT

JavaScript has the following kinds of scopes:

- **Global Scope**: A variable with global scope is one that can be accessed from anywhere in the application. It is the default scope for all code running in script mode.

Example :

```
{
var x = 2;
}
console.log(x) //global scope
```

- **Local Scope**: Any declared variable inside of a function is referred to as having local scope. Within a function, a local variable can be accessed. It throws an error if you attempt to access any variables specified inside a function from the outside or another function.The scope for code running in module mode.

Example :

```
// This part of code cannot use x
function myFunction() {
let x = 1;
// This part of code can use x
}
This part of code cannot use x
```

- **Function Scope**: In JavaScript, every new function results in the generation of a fresh scope. Variables declared inside a function cannot be accessed from outside the function or from another function. When used inside of a function, var, let, and const all act similarly. The scope created with a function.

Example :

```
function myFunction() {
const firstName = "Krishna"; // Function Scope
}
```

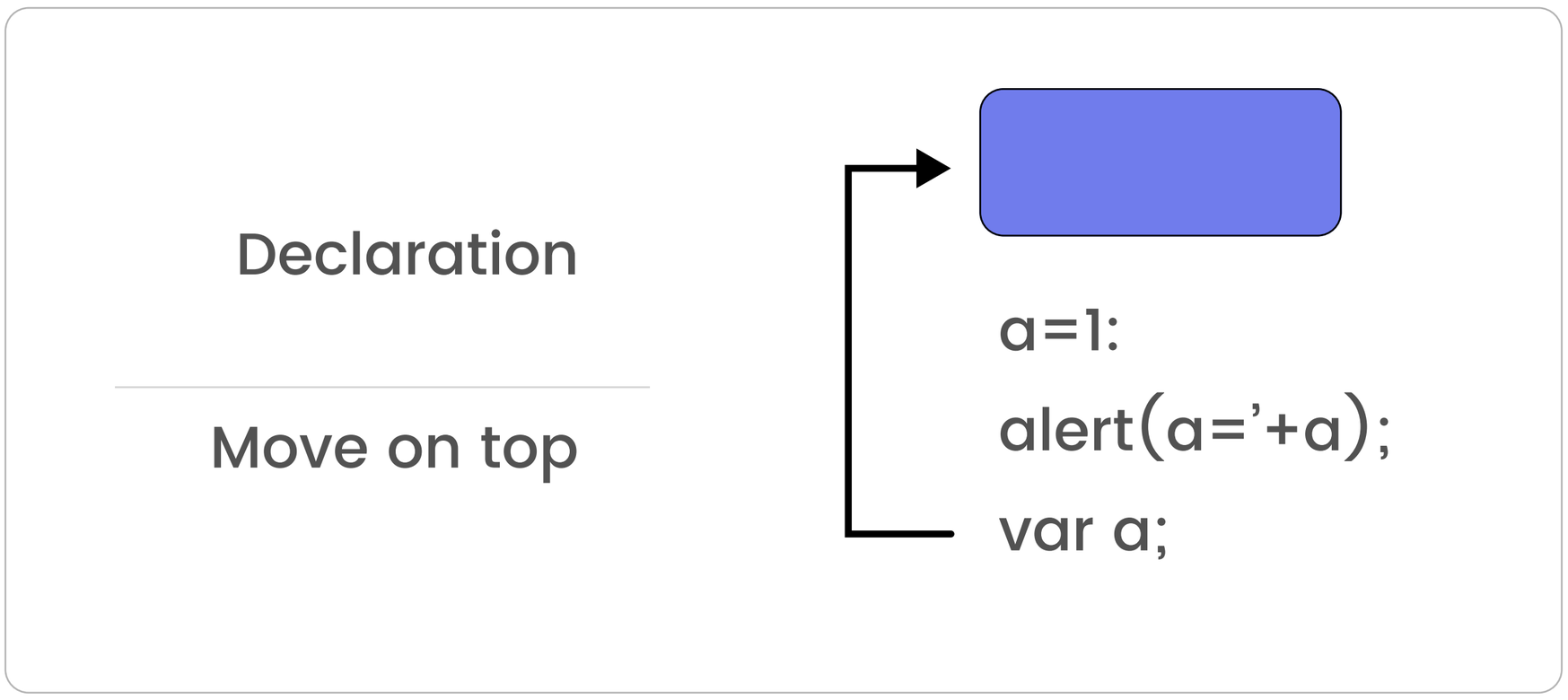
Scope Chain refers to the situation when one variable, which may have a global, local, function, or block scope, is used by another variable, function, or block, which may also have a global, local, function, or block scope. This entire chain construction continues till the user decides to halt it in accordance with the need.

4 WHAT ARE THE DIFFERENCES BETWEEN VAR, CONST & LET IN JAVASCRIPT?

var	let	const
The scope of a var variable is functional scope.	The scope of a let variable is block scope.	The scope of a let variable is block scope.
It can be updated and redeclared into the scope.	It can be updated but cannot be re-declared into the scope.	It cannot be updated or redeclared into the scope.
It can be declared without initialization.	It can be declared without initialization.	It cannot be declared without initialization.
It can be accessed without initialization as its default value is “undefined”.	It cannot be accessed without initialization otherwise it will give ‘referenceError’.	It cannot be accessed without initialization, as it cannot be declared without initialization.
Hoisting done, with initializing as ‘default’ value	Hoisting is done, but not initialized (this is the reason for the error when we access the let variable before declaration/initialization	Hoisting is done, but not initialized (this is the reason for the error when we access the const variable before declaration/ initialization

5 WHAT IS HOISTING IN JAVASCRIPT?

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



This means that irrespective of where the variables and functions are declared, they are moved on top of the scope. The scope can be both local and global.

6 EXPLAIN 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:

```
num = 23; // Gives reference error
let num;
function func(){
greeting = "Hi"; // Throws a reference error
let greeting;
}
func();
```

7 WHAT IS CLOSURE IN JAVASCRIPT?

A closure consists of references to the surrounding state (the lexical environment) and a function that has been wrapped (contained). In other words, a closure enables inner functions to access the scope of an outside function. Closures are formed whenever a function is created in JavaScript, during function creation time.

8 WHAT ARE DIFFERENCES BETWEEN “==” & “===”?

The == operator performs a loose equality comparison that, if necessary to enable the comparison, applies type coercion.

On the other hand, the === operator conducts a strict equality comparison without type coercion and necessitates that the operands be of the same type (as well as the same value).

9 WHAT IS NAN PROPERTY?

The NaN property is a global property that represents "Not-a-Number" value. i.e, It indicates that a value is not a legal number. It is very rare to use NaN in a program but it can be used as return value for few cases

For E.g.:

```
Math.sqrt(-1);
parseInt("Hello");
```

10 WHAT IS THE DIFFERENCE BETWEEN NULL AND UNDEFINED?

null	undefined
It is an assignment value which indicates that variable points to no object.	It is not an assignment value where a variable has been declared but has not yet been assigned a value.
Type of null is object	Type of undefined is undefined
The null value is a primitive value that represents the null, empty, or non-existent reference.	The undefined value is a primitive value used when a variable has not been assigned a value.
Indicates the absence of a value for a variable	Indicates absence of variable itself
Converted to zero (0) while performing primitive operations	Converted to NaN while performing primitive operations

11 WHAT ARE THE TERMS BOM AND DOM IN JAVASCRIPT?

DOM stands for **Document Object Model** and BOM for **Browser Object Model**.

DOM: An element may be added, changed, or removed from a document using the Document Object Model (DOM), a programming interface for HTML and XML documents. It specifies how a document is accessed and handled, as well as its logical structure. The DOM allows the webpage to be represented as a structured hierarchy, making it simple to access and modify HTML tags, IDs, classes, attributes, and elements using the Document object's provided commands and methods. This makes it easier for programmers and users to understand the document.

DOM provides several methods to find & manipulate the behavior of the HTML element:

- getElementById() Method
- getElementsByClassName() Method
- getElementsByName() Method
- getElementsByTagName() Method
- querySelector() Method
- querySelectorAll() Method

BOM: **Browser Object Model (BOM)** is a browser-specific convention referring to all the objects exposed by the web browser. The BOM allows JavaScript to “interact with” the browser. The window object represents a browser window and all its corresponding features. A window object is created automatically by the browser itself. JavaScript’s window.screen object contains information about the user’s screen.

Window properties of BOM are:

- screen.width
- screen.height
- screen.availWidth
- screen.availHeight
- screen.colorDepth
- screen.pixelDepth

Window methods of BOM are:

- window.open() Method
- window.close() Method
- window.moveTo() Method
- window moveBy() Method
- window.resizeTo() Method

12 WHAT IS CRITICAL RENDERING PATH?

The Critical Rendering Path is the sequence of steps the browser goes through to convert the HTML, CSS, and JavaScript into pixels on the screen. Optimizing the critical render path improves render performance. The critical rendering path includes the Document Object Model (DOM), CSS Object Model (CSSOM), render tree and layout.

13 WHAT ARE BASIC JAVASCRIPT ARRAY METHODS?

Some of the basic JavaScript methods are:

push() method: adding a new element to an array. Since JavaScript arrays are changeable objects, adding and removing elements from an array is simple. Additionally, it alters itself when we change the array's elements.

```
Syntax: Array.push(item1, item2 ...)
```

pop() method: This method is used to remove elements from the end of an array.

```
Syntax: Array.pop()
```

slice() method: This method returns a new array containing a portion of the original array, based on the start and end index provided as arguments

```
Syntax: Array.slice (startIndex , endIndex);
```

map() method: The map() method in JavaScript creates an array by calling a specific function on each element present in the parent array. It is a non-mutating method. Generally, the map() method is used to iterate over an array and call the function on every element of an array.

```
Syntax: Array.map(function(currentValue, index, arr), thisValue)
```

reduce() method: The array reduce() method in JavaScript is used to reduce the array to a single value and executes a provided function for each value of the array (from left to right) and the return value of the function is stored in an accumulator.

```
Syntax: Array.reduce(function(total, currentValue, currentIndex, arr), initialValue)
```

14 WHAT IS THE REST PARAMETER AND SPREAD OPERATOR?

Rest parameter (...):

- It offers a better method of managing a function's parameters.
- We can write functions that accept a variable number of arguments using the rest parameter syntax.
- The remainder parameter will turn any number of inputs into an array.
- Additionally, it assists in extracting all or some of the arguments.
- Applying three dots (...) before the parameters enables the use of rest parameters.

Syntax:

```
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.

Spread operator(...): Although the spread operator's syntax is identical to that of the rest parameter, it is used to spread object literals and arrays. Spread operators are also used when a function call expects one or more arguments.

Syntax:

```
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'};
```

15 EXPLAIN THIS KEYWORD

In JavaScript, the this keyword always refers to an object. The thing about it is that the object it refers to will vary depending on how and where this is being called. If we call this by itself, meaning not within a function, object, or whatever, it will refer to the global window object. The majority of the time, the value of this depends on the runtime binding used to call a function. It may change every time the function is called and cannot be changed by assignment while the function is being executed. Although arrow functions don't give their own this binding (it keeps the this value of the enclosing lexical context), the bind() method can set the value of a function's this regardless of how it's called

16 EXPLAIN CALL(), APPLY() AND, BIND() METHODS.

We use call, bind and apply methods to set the this keyword independent of how the function is called. This is especially useful for the callbacks.Every function object is also given a few unique methods and attributes by JavaScript. These are the methods that every JavaScript function inherits. Every function inherits certain methods, such as call, bind, and apply.

bind(): The bind method creates a new function and sets the this keyword to the specified object.

Syntax:

```
function.bind(thisArg, optionalArguments)
```

For example:

Let’s suppose we have two person objects.

```
const john = {
name: 'John',
age: 24,
};
const jane = {
name: 'Jane',
age: 22,
};
```

Let’s add a greeting function:

```
function greeting() {
console.log(`Hi, I am ${this.name} and I am
${this.age} years old`);
}
```

We can use the bind method on the greeting function to bind the this keyword to john and jane objects.

For example:

```
const greetingJohn = greeting.bind(john);
// Hi, I am John and I am 24 years old
greetingJohn();
const greetingJane = greeting.bind(jane);
// Hi, I am Jane and I am 22 years old
greetingJane();
```

Here greeting.bind(john) creates a new function with this set to john object, which we then assign to greetingJohn variable. Similarly for greetingJane.

call(): The call method initializes the this inside the function and launches it right away.
In contrast to bind(), which produces a copy of the function and sets the this keyword, call() sets the this keyword, executes the function instantly, and does not create a new copy of the function.

Syntax: function.call(thisArg, arg1, agr2, ...)

For example:

```
function greeting() {
console.log(`Hi, I am ${this.name} and I am
${this.age} years old`);
}
const john = {
name: 'John',
age: 24,
};
const jane = {
name: 'Jane',
age: 22,
};
// Hi, I am John and I am 24 years old
greeting.call(john);
// Hi, I am Jane and I am 22 years old
greeting.call(jane);
```

Above example is similar to the bind() example except that call() does not create a new function. We are directly setting the this keyword using call().

apply(): The apply() method is similar to call(). The difference is that the apply() method accepts an array of arguments instead of comma separated values.

Syntax: function.apply(thisArg, [argumentsArr])

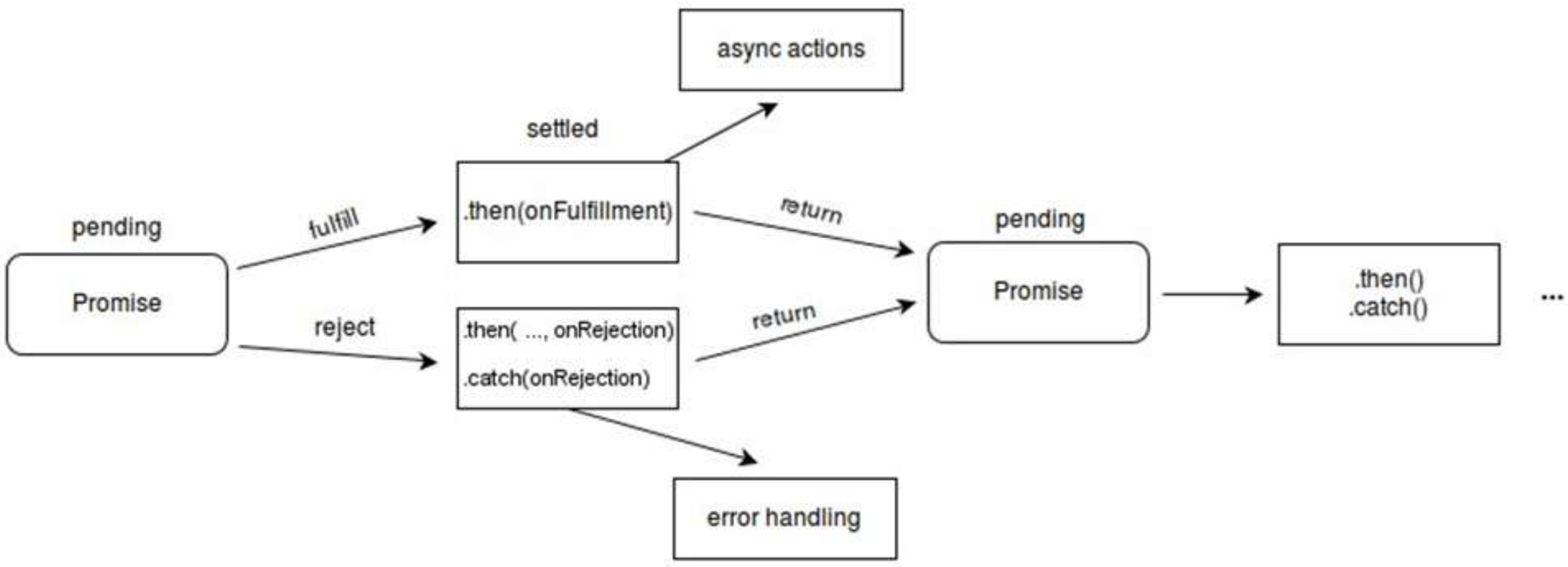
For example:

```
function greet(greeting, lang) {
console.log(lang);
console.log(`${greeting}, I am ${this.name}
and I am ${this.age} years old`);
}
const john = {
name: 'John',
age: 24,
};
const jane = {
name: 'Jane',
age: 22,
};
// Hi, I am John and I am 24 years old
greet.apply(john, ['Hi', 'en']);
// Hi, I am Jane and I am 22 years old
greet.apply(jane, ['Hola', 'es']);
```

17 IS JAVASCRIPT SINGLE-THREADED, IF YES THEN HOW IT WORKS AS AN MULTI-THREADED LANGUAGE? OR WHAT IS EVENT LOOP IN JAVASCRIPT?

JavaScript is a single-threaded asynchronous programming language. But what does it mean? What is this event loop in JavaScript that we all keep talking about? To know more click here

18 WHAT ARE PROMISES, ASYNC-AWAIT AND CALLBACK?



A Promise is a proxy for a value not necessarily known when the promise is created. It allows you to associate handlers with an asynchronous action's eventual success value or failure reason.

This lets asynchronous methods return values like synchronous methods: instead of immediately returning the final value, the asynchronous method returns a promise to supply the value at some point in the future.

A Promise is in one of these states:

- pending: initial state, neither fulfilled nor rejected.
- fulfilled: meaning that the operation was completed successfully.
- rejected: meaning that the operation failed.

A promise is said to be settled if it is either fulfilled or rejected, but not pending.

Async simply allows us to write promises-based code as if it was synchronous and it checks that we are not breaking the execution thread. It operates asynchronously via the event loop. Async functions will always return a value. It makes sure that a promise is returned and if it is not returned then JavaScript automatically wraps it in a promise which is resolved with its value.

Syntax:

```
const func1 = async() => {
return “Hello World!”;
}
```

Await function is used to wait for the promise. It could be used within the async block only. It makes the code wait until the promise returns a result. It only makes the async block wait.

Syntax:

```
const func2 = async () = {
let x= await func1();
console.log(x);
}
```

19 WHAT IS CALLBACK HELL?

Callback Hell is an anti-pattern with multiple nested callbacks which makes code hard to read and debug when dealing with asynchronous logic.

The callback hell looks like below,

Syntax:

```
async1(function(){
async2(function(){
async3(function(){
async4(function(){
....
});
});
});
});
```

20 WHAT ARE OBSERVABLES?

Observables in JavaScript are a way to handle asynchronous events. They are functions that return a stream of values, which can be used to represent data streams such as DOM events, mouse events, or HTTP requests.

Observables work by providing a way to subscribe to a stream of values, and then receiving those values as they become available. This allows you to respond to events in a more reactive way, without having to wait for the entire event stream to complete before processing it.

To use observables in JavaScript, you can use the RxJS library.

```
import { Observable } from 'rxjs';
const observable = new Observable(subscriber => {
subscriber.next(1);
subscriber.next(2);
subscriber.next(3);
subscriber.complete();
});
observable.subscribe(value => {
console.log(value);
});
```

21 WHAT ARE THE DIFFERENCES BETWEEN PROMISES AND OBSERVABLES?

Promises	Observables
Emits only a single value at a time	Emits multiple values over a period of time(stream of values ranging from 0 to multiple)
Eager in nature; they are going to be called immediately	Lazy in nature; they require subscription to be invoked
Promise is always asynchronous even though it resolved immediately	Observable can be either synchronous or asynchronous
Doesn't provide any operators	Provides operators such as map, forEach, filter, reduce, retry, and retryWhen etc
Cannot be cancelled	Cancelled by using unsubscribe() method

21 WHAT IS THE DIFFERENCE BETWEEN SETTIMEOUT, SETIMMEDIATE AND PROCESS.NEXTTICK?

setTimeout(): Using the setTimeout() method, a callback function can be scheduled to run once after a millisecond delay.

setImmediate(): Use the setImmediate function to run a function immediately following the conclusion of the current event loop.

process.nextTick(): If process.nextTick() is invoked in a given phase, all the callbacks passed to process.nextTick() will be resolved before the event loop continues. This will block the event loop and create I/O Starvation if process.nextTick() is called recursively.

23 WHAT IS MICROTASK IN JAVASCRIPT?

A microtask is a short function which is executed after the function or program which created it exits and only if the JavaScript execution stack is empty, but before returning control to the event loop being used by the user agent to drive the script's execution environment.

24 WHAT PURE FUNCTIONS IN JAVASCRIPT?

A function or section of code that always yields the same outcome when the same arguments are supplied is known as a pure function. It is independent of any state or data changes that occur while a program is running. Instead, it just relies on the arguments it is given.

Additionally, a pure function does not result in any side effects that can be seen, such as network queries, data alteration, etc.

25 WHAT IS AN ERROR OBJECT AND ITS DIFFERENT ERROR NAME OBJECT?

When an error happens, an error object—a built-in error object—provides error information. There are two attributes: name and message. For instance, the following function records error information,

Syntax:

```
try {
greeting("Welcome");
} catch (err) {
console.log(err.name + "<br>" + err.message);
}
```

There are 6 different types of error names returned from error object,

Error name	Description
EvalError	An error has occurred in the eval() function
RangeError	An error has occurred with a number "out of range"
ReferenceError	An error due to an illegal reference
SyntaxError	An error due to syntax
TypeError	An error due to a type error
URIError	An error due to encodeURI()

26 WHAT ARE THE VARIOUS STATEMENTS IN ERROR HANDLING?

Below are the list of statements used in an error handling,

- try: This statement is used to test a block of code for errors
- catch: This statement is used to handle the error
- throw: This statement is used to create custom errors.
- finally: This statement is used to execute code after try and catch regardless of the result.

27 WHAT DO YOU MEAN BY STRICT MODE IN JAVASCRIPT AND CHARACTERISTICS OF JAVASCRIPT STRICT-MODE?

In ECMAScript 5, a new feature called JavaScript Strict Mode allows you to write a code or a function in a "strict" operational environment. When it comes to throwing errors, javascript is often 'not extremely severe'. However, in "Strict mode," all errors, even silent faults, will result in a throw. Debugging hence becomes more easier. Thus, the chance of a coder committing a mistake is decreased.

Characteristics of strict mode in javascript:

- Duplicate arguments are not allowed by developers.
- Use of javascript's keyword as parameter or function name is not allowed.
- The 'use strict' keyword is used to define strict mode at the start of the script. Strict mode is supported by all browsers
- Creating of global variables is not allowed.

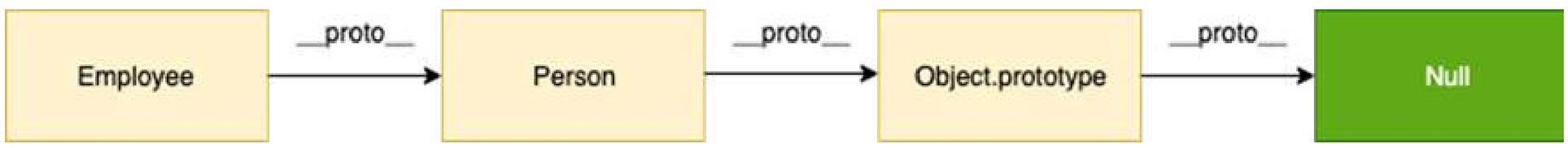
28 WHAT ARE THE DIFFERENCES BETWEEN COOKIE, LOCAL STORAGE AND SESSION STORAGE?

Cookie	Local storage	Session
Can be accessed on both server- side & client side	Can be accessed on client- side only	Can be accessed on client- side only
As configured using expires option	Lifetime is until deleted	Lifetime is until tab is closed
SSL is supported	SSL is not supported	SSL is not supported
Maximum size is 4 KB	Maximum size is 5 MB	Maximum size is 5 MB

29 EXPLAIN PROTOTYPE CHAINING

Prototype chaining is used to build new types of objects based on existing ones. It is similar to inheritance in a class based language.

The prototype on object instance is available through Object.getPrototypeOf(object) or __proto__ property whereas prototype on constructors function is available through Object.prototype.



30 WHAT ARE GENERATORS AND WHAT ARE ITS DIFFERENT KINDS?

Introduced in the ES6 version, generator functions are a special class of functions.

They can be stopped midway and then continue from where they had stopped.

Generator functions are declared with the function* keyword instead of the normal function keyword.

There are five kinds of generators,

- Generator function declaration
- Generator function expressions
- Generator method definitions in object literals
- Generator method definitions in class
- Generator as a computed property

31 DIFFERENCE BETWEEN DEBOUNCING AND THROTTLING.

Debouncing	Throttling
Debouncing waits for a certain time before invoking the function again.	An error has occurred in the eval() function
Ensures that the function is called only once, even if the event is triggered multiple times.	An error has occurred with a number "out of range"
Useful when you want to delay the invocation of a function until a certain period of inactivity has passed.	An error due to an illegal reference
Eg. You can debounce an async API request function that is called every time the user types in an input field.	An error due to syntax

Syntax:
function debounce(func, delay) {
let timerId;
return function () {
const context = this;
const args = arguments;
clearTimeout(timerId);
timerId = setTimeout(function () {
func.apply(context, args);
}, delay);
};
}
	Syntax:
	function throttle(callback, delay = 1000) {
	let shouldWait = false;
	return (...args) => {
	if (shouldWait) return;
	callback(...args);
	shouldWait = true;
	setTimeout(() => {
	shouldWait = false;
	}, delay);
	};
	}