**JavaScript Interview Series**

1. **JavaScript and its Environment:**

* JavaScript is one of the smartest programming languages which are exist till now, and to run Js we need an Environment called JavaScript Run Time Environment.
* We simply just install (i.e., Browser, NodeJS) it and then Js run on our system.
* Only logics are written in JavaScript.
* Different browser uses different JavaScript engines to run Js like Chrome uses V8, Firefox uses Monkey.
* All features are given in Js is by environment only like **setTimeOut()** function in Js is not a part of Js, it is the feature of Browser.
* All complex work in Js is done by environment (i.e., Thread, Queuing and all)
* Electron Js is used to make Desktop Apps.
* Js is developed within 10Days and created by **Brendan Eich of Netscape**.
* When it is formed basically it was created by combining 3 languages Schema (Way of describing Objects), Self (OOPs Concept), Java (Syntax) and Functional Programming and named as LiveScript and later known as JavaScript.

1. **ECMA Script:**

* It is the association of Europe who basically defines the standard of language.
* Changes and updating in **ECMA Script** is not written by any individual or company, actually a committee **TC-39** was held and where several Browsers vendor come and discuss about their opinion and then later on every Browser will implement their own changes through their browser engines.
* It has many versions like ES-1, ES-2, ES-3, ES-5, ES-6.

1. **Vanilla JavaScript:**

* Normal Js that we use is called as Vanilla Js.

1. **Angular Js/ Vue Js/ React Js:**

* Basically, the cost of changing the UI of any browser will increase in their cost so all we need to do is use these libraries of Js and then it will change the UI according to that and it will automatically the change the Algorithm as needed.

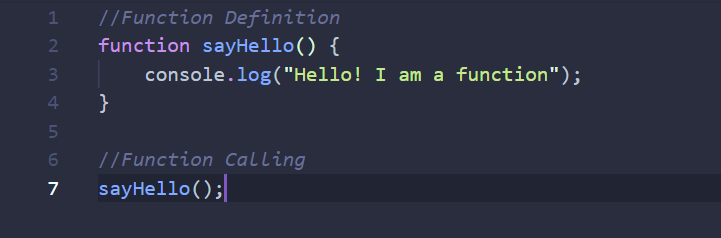
**Functions in JavaScript**

* In Js, functions play a very important role.
* It is also called as **FIRST-CLASS CITIZEN**.

Qns1: How to write a function in Js?

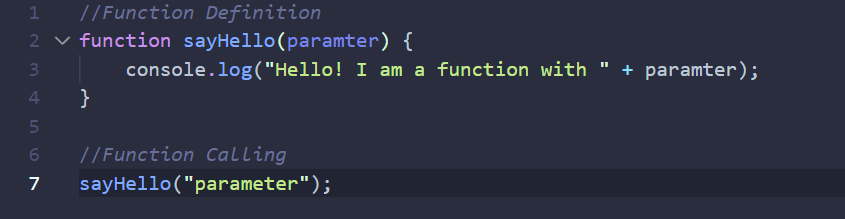
Ans1: First Way (without Parameters)

Output 🡪 Hello! I am a function



First Way (with Parameters)

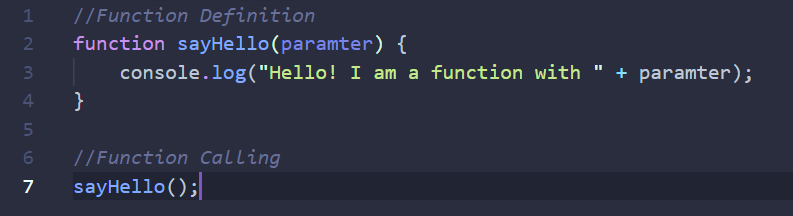
Output 🡪 Hello! I am a function with parameter



Qns2: What if we define the function with parameters but at the time of calling, we didn’t provide any parameter?

Ans2: Yes, the function will execute but where parameter is used to print, we saw **undefined**.

Output 🡪 Hello! I am a function with **undefined**

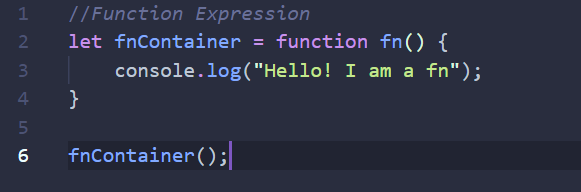


* Functions are treated as a variable in Js.
* Assignment is possible in Js as in variables.

1. **Function Expression**:

* In this we basically assign the function into a variable and then call the function.

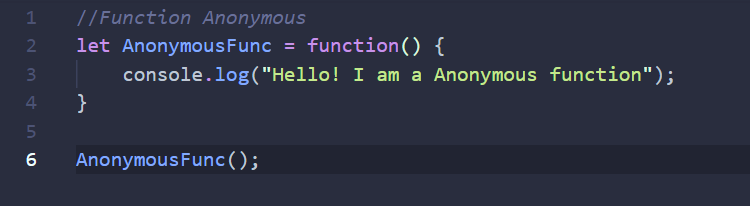
Output 🡪 Hello! I am a fn



1. **Function Anonymous:**

* When we didn’t give name to the function then it is called as Anonymous Function.
* In this case name of function is identify by the variable it is assigned to.

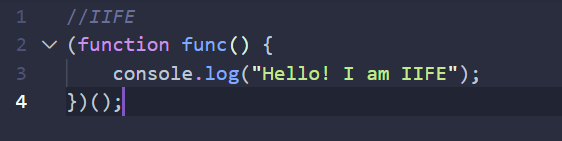
Output 🡪 Hello! I am a Anonymous function



1. **IIFE (Immediately Invoked Function Expression):**

* Initially, when **let, const** (Js variables) is not introduced then we use IIFE which basically runs automatically when user starts/ runs the program.
* So, when we write function then we just call it, this is called as IIFE.
* It helps to solve function like require and pollution.

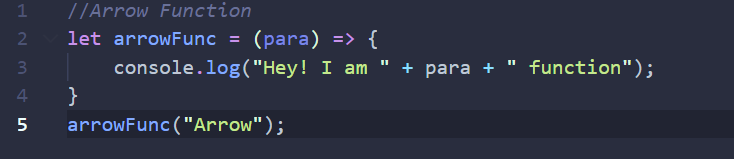
Output 🡪 Hello! I am IIFE



1. **Arrow Function:**

* It helps to solve the problems in react, this.

Output 🡪 Hey! I am Arrow function



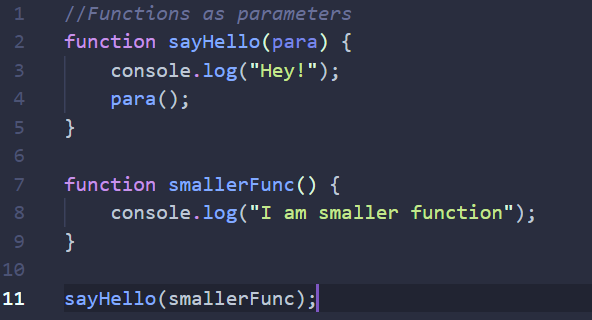
**First Class Citizens:**

* It simply means **“Being able to do what everyone else can do”!**

1. Functions are treated as variables.
2. We can assign function as in variables.
3. Functions can be passes as parameters (Functional Programming).

Output 🡪 Hey!

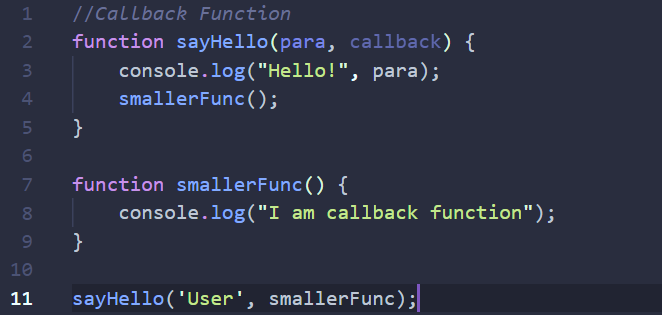
I am smaller function



1. Callback Functions

Output 🡪 Hello! User

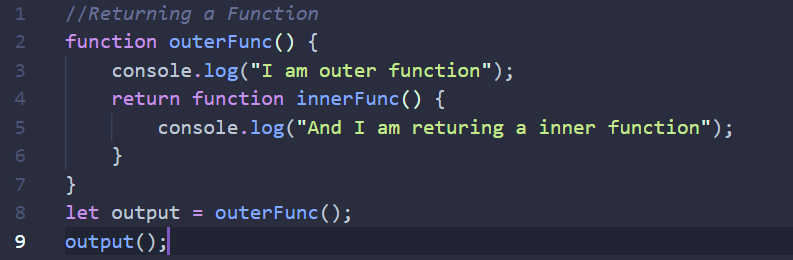
I am callback function



1. It also returns a function

Output 🡪 I am outer function

And I am returing a inner function



**Execution Context:**

* In JavaScript, execution context is an abstract concept that holds information about the environment within which the current code is being executed.
* The JavaScript engine creates the global execution context before it starts to execute any code.
* When execution is made it comes with ‘**global and this**’ object and both of them are made by Nodejs or environment engine which is being used not by Js. So, when we run ‘**global and this’** object in Nodejs ‘global returns an object’ ‘while this return empty object’ whereas browser provided ‘**window**’ and **‘this object’** and on running them both on browser refers to a same object.
* In Js there is no main function in which whole code is written and run, in Js Execution context is there which basically returns ‘**global and this**’ object.

**Global Execution Context:**

Creation

**Hoisting**

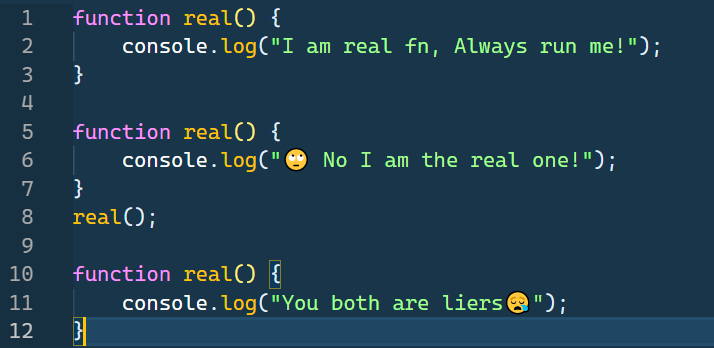
Variables 🡪 Memory Allocate (undefined)

Functions 🡪 Memory Allocate

**CODE**

* So, when a variable is called in a print statement without defining it, it may return undefined because in Global Execution Context it was never allocate, but talking about functions when a function is called without defining in top, it may still call it (if exist) as at the time of global execution context memory is already allocated to the function called **HOISTING**.

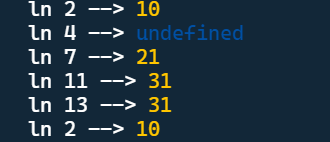
Output 🡪You both are liers😪

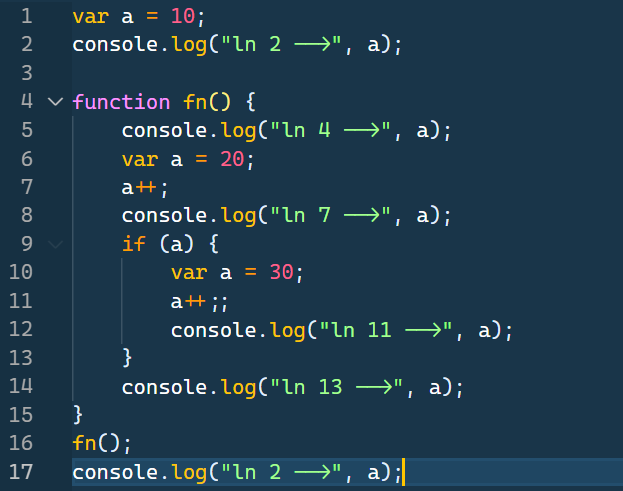


* Here in this problem last real function is print because as in hoisting, memory is allocated to all the functions present in the code so initially in heap first real function (line 1) gets its memory allocated and now on checking further it find another same function (line 5) is defined so now memory is allocated to now second real function and now on checking further it finds another real function is defined (line 10) so now memory is allocated to this one (line 10) and on calling the real function it will print the last real function in which memory is allocated (line 10).

**Var Keyword:**

Output 🡪





* Here at line 14 the value of a is 31 as if var, redefine defined once, then its value is same though out the code until and unless we can’t change its value.

**Lexical Scope and Scope Chain**

varName = undefined

Empty Object as we had run code in Nodejs

This – {}C



Call Stack

Function Execution Context

varName = undefined

ThisC

GlobalC

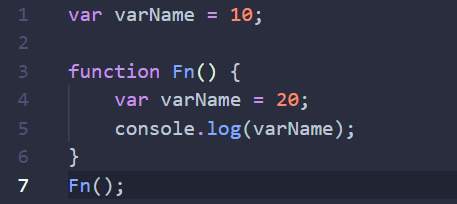
Global Execution Context

GlobalC

FnC

HeapC

Output 🡪 20

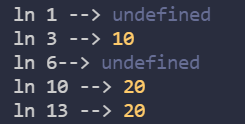


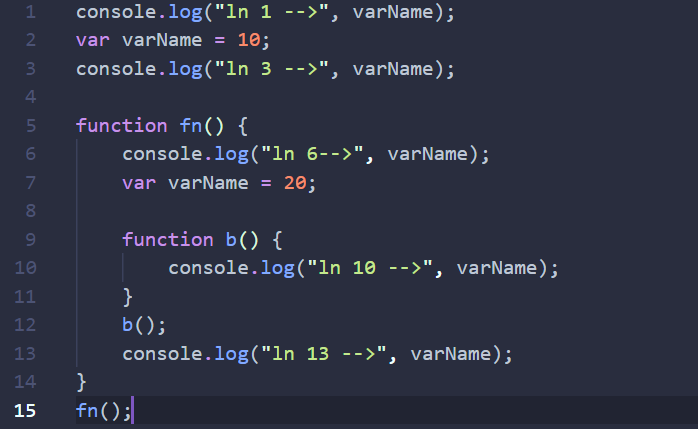
* Initially, we don’t have main function so instead of this whole code runs upon **Global execution context**. And when every function call and came on stack then one execution context is made of each function.
* **Execution Context** is made only when the function is called.
* Firstly, memory will allocate in Execution Context and then the code is run but only after the Execution Context is formed.
* So, before execution context if we try to print then it returns the undefined as execution context is not formed yet.

**Scope:**

* Area where a function or a variable can be found.

Output 🡪



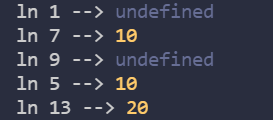


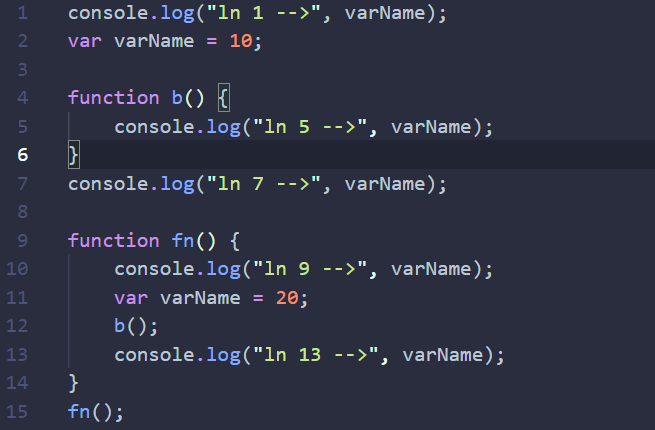
* Here in this problem when function b is executed it finds that it doesn’t have any varName inside it. So, to resolve this problem it simply borrow from its outer function or scope so the function will be executed as b has take the value from fn function which contain the value of varName as 20 but if in case fn function also don’t have any varName defined in it then the function b will check outer function or scope of fn i.e., anonymous and if it finds the varName then it will print it otherwise simply give the error (ReferenceError: varName is not defined).

1. **Lexical Scope:**

* A lexical scope in Js means that a variable defined outside a function can be accessible inside another function defined after the variable declaration.

Output 🡪





* Here when b function is called at line 12 then it finds that it doesn’t has any varName defined in it so now to run this function it need varName and then it simply takes from the anonymous function (line 2) where varName is defined as 10 but the main thing is that it can’t borrow or take the value of varName from the fn function (lime 9) because as mentioned in lexical scope when function didn’t find the variable assigned in it, it simply take the value from outside function definition so at line 12 function b is not defined it is called but at line 4 the function is defined so we take the outside value from there i.e., 10 (line 2)

1. **Scope Chain:**

* Within each execution context is a special object called a **scope chain** which is used to resolve variables.
* A **scope chain** is essentially a stack of currently accessible **scopes**, from the most immediate context to the global context.

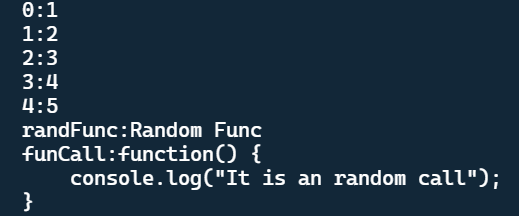
**Some popular Array Methods on JavaScript:**

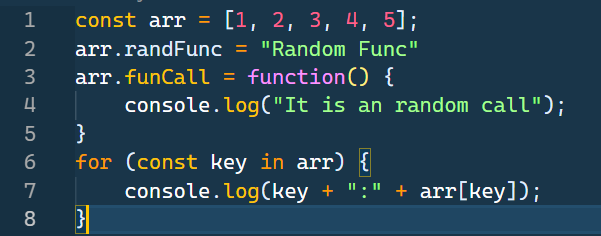
|  |  |
| --- | --- |
| **Array Methods in JavaScript** | **Their Function** |
| ***pop():*** | Remove an item from the end of an array. |
| ***push():*** | Add items to the end of an array. |
| ***shift():*** | Remove an item from the beginning of an array. |
| ***unshift():*** | Add items to the beginning of an array. |

**Fun Facts about JavaScript Arrays, Objects and Functions:**

* In Js basically ***Arrays acts as Objects*** as left-hand side parameters acts as key where right-hand side acts as their value.

Output 🡪





* We can pass any value, string, object, function in the array and later call them or print them.
* According to ECMA Script it is defined that Functions are the Objects, which means we can give key and value to a function and it simply return that on printing.
* In function a special property called Code Property is provided that can be executed when you invoke those functions i.e., inside a function some print statements are written that can be run we e call the function.

**JavaScript is Primitive or Object?**

|  |  |  |
| --- | --- | --- |
| Sr. No | Primitive | Objects |
| 1.  2.  3. | In JavaScript, a **primitive** (value, data type) is data that is not an object and has no methods.  There are **7 primitive data types-** string, number, big int, boolean, undefined, symbol, and null.  All **primitives** are immutable, i.e., they cannot be altered. | JavaScript is designed on a simple object-based paradigm.  An object is a collection of properties, and a property is an association between a name (or key) and a value.  In addition to objects that are predefined in the browser, you can define your own objects. |