**All about JavaScript from Harkirat's Cohort**

JS is an interpreted language

JS is a Dynamic Language

JS is a Single Threaded Language

**Synchronous Functions:** Functions that basically runs line by line (hence “synchronous”). Below are some examples of this particular function.

**<str>.length()**

**<str>.indexOf(<target>)**

**<str>.lastIndexOf(<target>)**

**<str>.slice(<start index>, <end index>)**: the last index is not included

**<str>.replace(<target word>, <replacement word>)**

**<str>.trim()**

**parseInt(<str>)**: other data types included

**<arr>.push(<desired element)**: inserts the desired element at the end of the array

**<arr>.unshift(<desired element)**: inserts the desired element at the beginning of the array

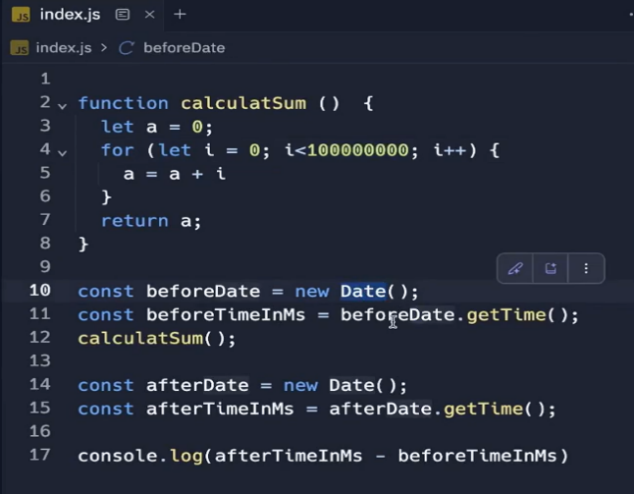
**<arr>.pop()**: deletes the element of the last index

**<arr>.shift()**: deletes the element of the 1st index

**<arr>.concat(<arr>):** Difference between push and concat, in concat we are adding a whole array and in push we’re adding an element

**<obj>.getTime()**: you should create an object of Date type which is a predefined class in JS containing many functions getDate(), getFullYear(), etc.

One cool thing we can do with getTime() is that we can **measure the execution time** of a function in ms. Let’s take an example by which we can measure the time taken by calculatSum() to execute.



**JSON.parse(<string>):** converts string to an object (NOTE: The keys of the object should also be written in double quotes specifically and additionally

**JSON.stringify(<obj>)**: converts object to string

**Asynchronous Functions:** Those functions that allow a program to start a potentially long running operation and continue executing other tasks without waiting for that operation to complete. This is particularly important in environments like web browsers or Node.js where waiting for an operation to finish (like fetching a data from a server or reading a large file) could make the application unresponsive. Two important such functions are:

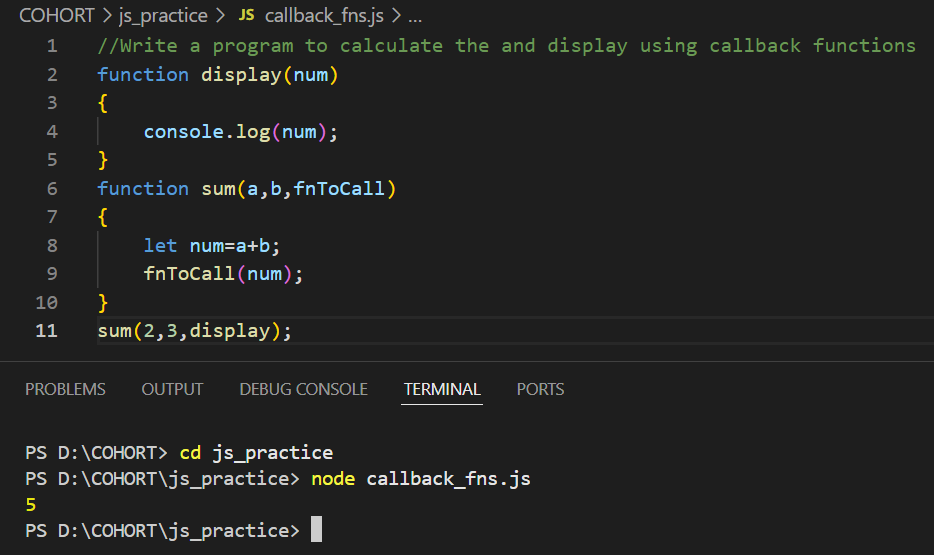
**setInterval(<function reference>, <time delay in ms>)**: schedules the provided function to run repeatedly after every delay, and will continue running the function until explicitly stopped using clearInterval().

**setTimeout(<function refernce>, <time delay in ms>):** runs a function once after a delay

**Callback Functions?**

In JavaScript, a callback function is a function passed as an argument to another function, intended to be executed at a later time.

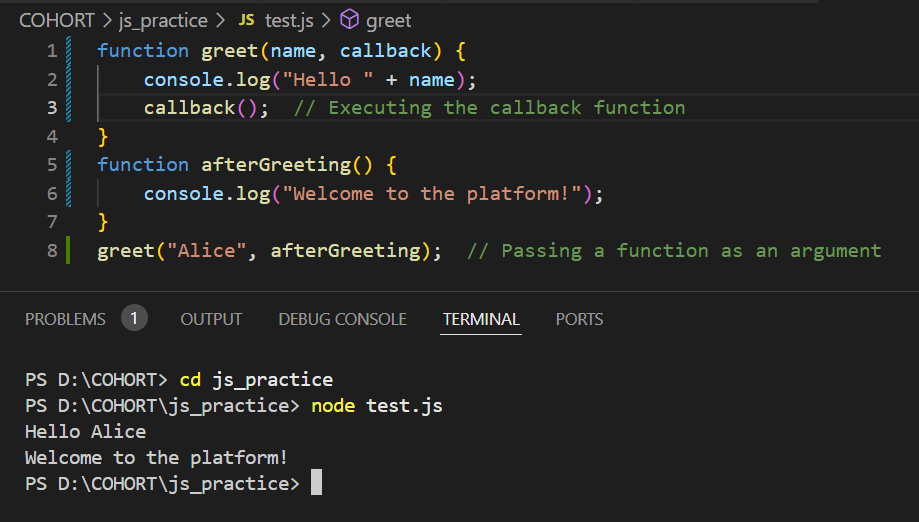
**A simple example demonstrated is:**

**

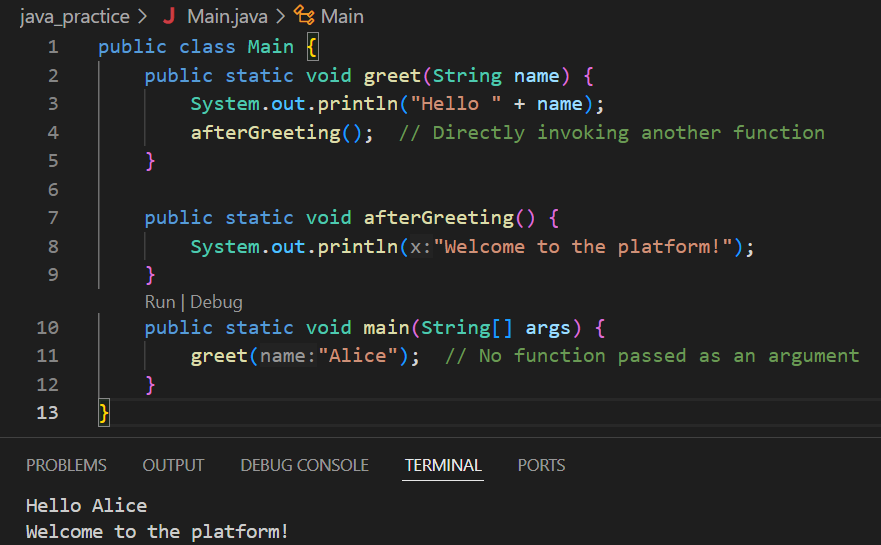
This differs from typical function calls in languages like Java, where functions are usually invoked directly within another function's block rather than passed as arguments.

**For example:**

**JavaScript (using a callback):**

**

**Java (direct function call):**



If we connect the dots we see, setTimeout() and setInterval() are also a showcase of the use of callback functions. Since when we use either of the functions we call the function refernce basically the function which we are calling is being passed as an argument inside the setTimeout() or setInterval() functions.

This mechanism supports asynchronous programming by enabling JavaScript to handle tasks that might take time, like fetching data from an API or processing large files.

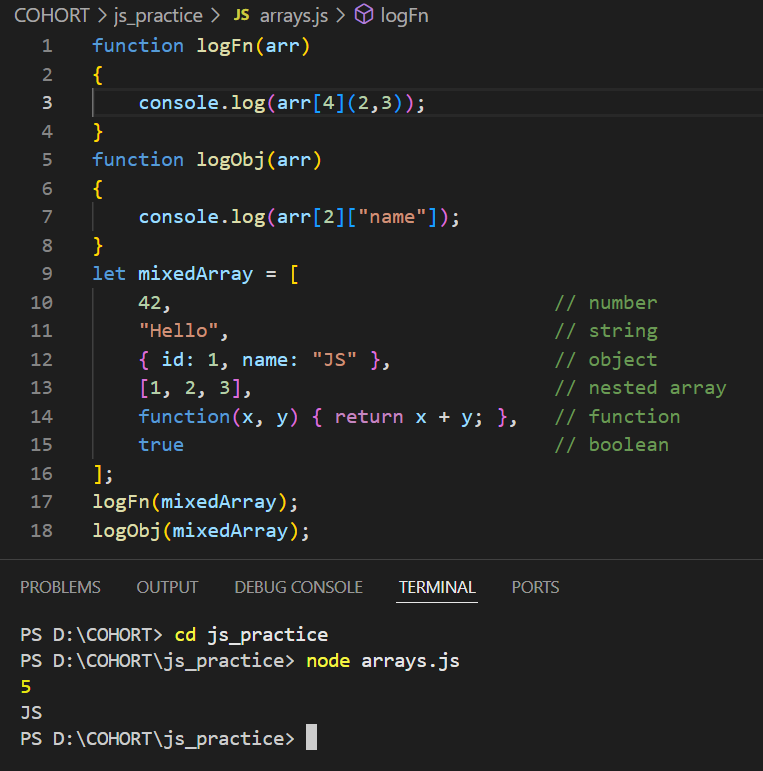
When a function is passed as an argument, it doesn't run immediately but is instead stored for execution at the appropriate time. This approach makes JavaScript more flexible and allows developers to create more dynamic and responsive applications.

**Arrays in JS**

Unlike statically-typed languages where arrays must contain elements of a predetermined type, JavaScript arrays are dynamic containers. Arrays in JS have

1. Type flexibility: JavaScript arrays can hold mixed types (numbers, strings, objects, functions etc.) in the same array
2. Dynamic sizing: JavaScript arrays can grow or shrink dynamically
3. Array declaration: let arr = [1,2,3,4]; (In Java we use curly brackets {} while initialising the elements of the array but in JS we use the square brackets [];

The feature that arrays in JS can also include functions and objects is so cool. Let’s check it out:



**Objects in JS**

In JavaScript, objects are a key-value-based structure, and they are one of the foundational ways to store and organize data.

They Store key-value pairs in an unordered way.

Example:

const dog = { name: "doggie", speaks: "bhow bhow" }

The variable dog here is an object literal. Arrays and objects may look like arrays but they serve different purpose. Objects are:

* Key-value pairs: Each property (key) is tied to a specific value.
* Unordered: The properties don't maintain a specific order.
* Used for structured data: Best for representing entities or objects with attributes.

One cool thing in JS is that we can also pass objects as arguments in a function.