Skill Training
Advanced JavaScript

Narasimha

Sr. Corporate Trainer, Mentor

tnrao.trainer@gmail.com

Schedule for Advanced CSS & JS

Day1 : Advanced JS : ES6, Arrow Functions,…

Day2 : Advanced JS : OOPs, Modules, Closures

Day3 : Advanced JS : Asynchronous, Promises,…

Day4 : Advanced CSS : CSS3 Layouts, Media Queries

Day5 : Advanced CSS: UI Frameworks

Advanced JavaScript
Asynchronous Programming

Narasimha
Sr. Corporate Trainer, Mentor
tnrao.trainer@gmail.com

Index – Day3

1. **Asynchronous JavaScript**
2. **Promises**
3. **Server Calls using fetch() API**
4. **Event Loop**

Narasimha

Sr. IT Trainer/Consultant

Asynchronous Programming

What is Callback?

Narasimha

Sr. IT Trainer/Consultant

What is Callback?

- A callback is a function passed as an argument to another function.
- This technique allows a function to call another function.
- A callback function can run after another function has finished.

Promises

Narasimha

Sr. IT Trainer/Consultant

The **Promise** object represents the eventual completion (or failure) of an asynchronous operation and its resulting value.

What is Promise?

- Promises are used to handle asynchronous operations in JavaScript.
- The Promise object represents the eventual completion (or failure) of an asynchronous operation and its resulting value.

States in Promise

A Promise is in one of these states:

a. **pending**: initial state, neither fulfilled nor rejected.
b. **fulfilled**: meaning that the operation was completed successfully.
c. **rejected**: meaning that the operation failed.

How to create promise?

```
const myPromise = new Promise((resolve, reject) => {
        resolve(“success");
        (or)
        reject(“error”);
});
```

How to subscribe promise?

Syntax: myPromise.then(callback);

Eg:

```
        myPromise.then( (response) =>
        {

        } );
```

How to subscribe promise?

```
myPromise.then( (response) =>
{


}
. catch( (error) =>
{


});
```

Server Calls using fetch() API

Narasimha

Sr. IT Trainer/Consultant

Fetch()

```
fetch(url).then( (response)  =>
{
          response.json().then(  (resData)  =>
          {
                    displayData(  resData.records );

          });
});
```

Async/Await

Narasimha

Sr. IT Trainer/Consultant

Async & Await

- There’s a special syntax to work with promises in a more comfortable fashion, called “async/await”.
- It’s surprisingly easy to understand and use.
- We can simplify promise creation using async functions.

Async & Await

- An async function is a function declared with the **async** keyword, and the **await** keyword is permitted within it.
- The async and await keywords enable asynchronous, promise-based behavior to be written in a cleaner style, avoiding the need to explicitly configure promise.

The **Promise** object represents the eventual completion (or failure) of an asynchronous operation and its resulting value.

Async & Await

- **async** makes a function return a Promise
- **await** makes a function wait for a Promise

Async & Await

```
async function   getServerData()
{
  let url =  "https://www.w3schools.com/angular/customers.php";
  let response =  await fetch(url);
  let finalResult = await response.json();
  return finalResult;
}
```

Async & Await

```
async function   getServerData()
{
       try {
        let url =  "https://www.w3schools.com/angular/customers.php";
        let response =  await fetch(url);
        let finalResult = await response.json();
        return finalResult;
      }
      catch(error)  {
             // code
      }
}
```

Practice Hands-Ons

Narasimha

Sr. IT Trainer/Consultant

Event Loop

What is Event loop?

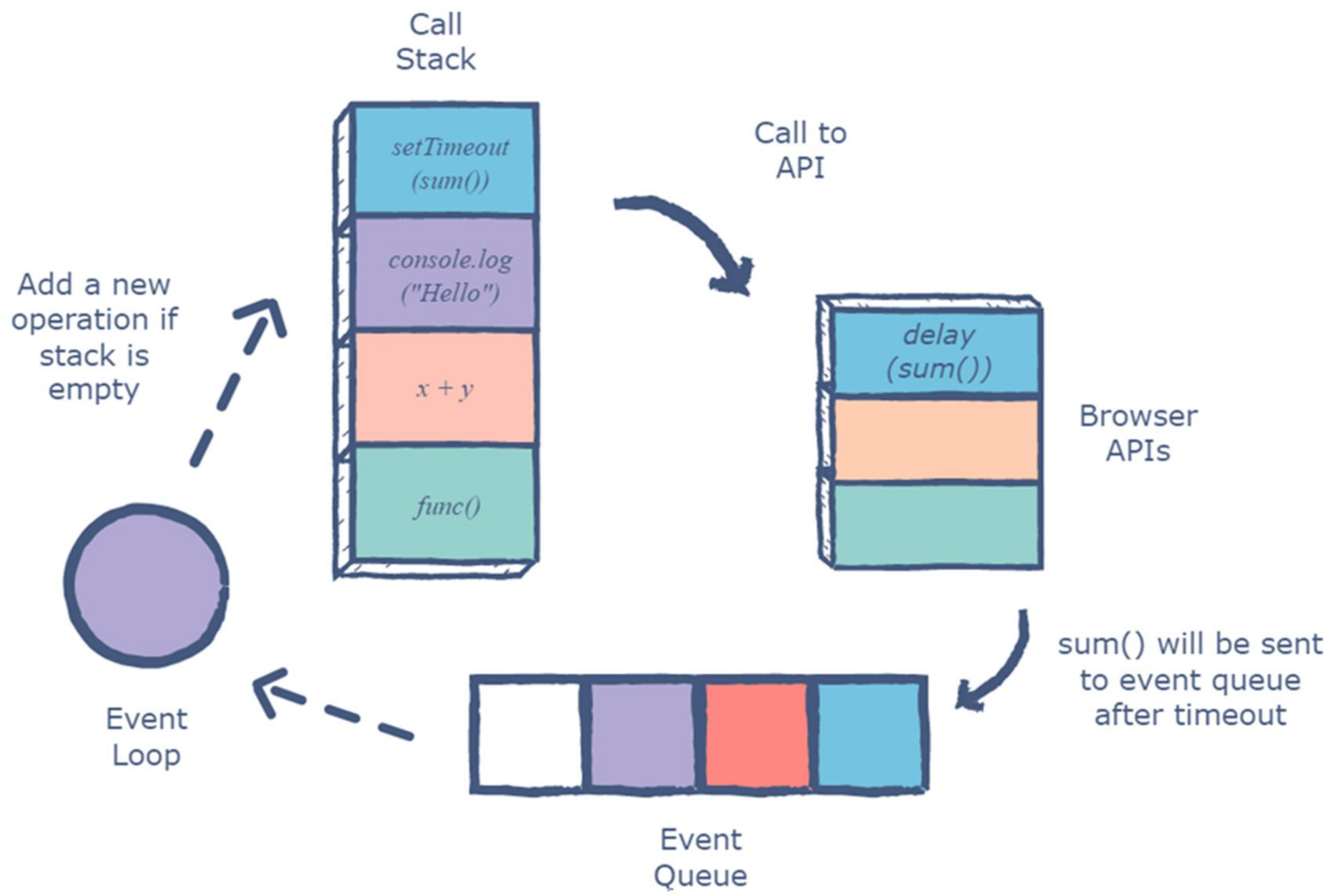
- In computer science, the event loop is a programming construct or design pattern that waits for and dispatches events or messages in a program.

Event loop in JavaScript

- The **event loop** is the secret behind JavaScript’s asynchronous programming.
- JavaScript has a runtime model based on an event loop.
- JS executes all operations on a single thread, but using a few smart data structures, it gives us the illusion of multi-threading.

Event loop in JavaScript

- Event loop is responsible for
  - executing the code
  - collecting and processing events,
  - executing queued sub-tasks.


Call Stack
setTimeout (sum())
console.log ("Hello")
x + y
func()
Call to API
delay (sum())
Browser APIs
sum() will be sent to event queue after timeout
Event Queue
Event Loop
Add a new operation if stack is empty

Practice Hands-Ons

Narasimha

Sr. Corporate Trainer, Mentor

tnrao.trainer@gmail.com

Narasimha

Sr. Corporate Trainer, Mentor

9030005961, tnrao.trainer@gmail.com