

# Async and Sync

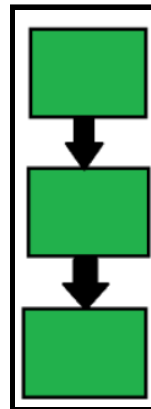
## Topics Covered:

- Synchronous JavaScript
- Asynchronous JavaScript
- Memory Allocation
- Function call stack
- Event Loop
- Callback Hell

## Topics in Detail:

### Synchronous JavaScript

- Every statement in a **code** is **executed** in a **sequence**, one after the other.
- Every statement will **wait** for one statement to **complete execution**.
- JavaScript is a **single-threaded synchronous programming language**.
- The JavaScript code **does not** run in **parallel**, but it can only **run one at a time**.



```
console.log("Before delay");

function delayBySeconds(sec) {
  let start = now = Date.now()
  while(now-start < (sec*1000)) {
    now = Date.now();
  }
}

delayBySeconds(5);

// Executes after delay of 5 seconds
console.log("After delay");
```

## Output

```
Before delay
(... waits for 5 seconds)
After delay
```

## Asynchronous JavaScript

- The program will be executed **immediately** in **asynchronous code**.
- Many operations can be performed simultaneously in **AJAX**.

```
<script>
  document.write("Hi");
  document.write("<br>");

  setTimeout(() => {
    document.write("Let us see what happens");
  }, 2000);

  document.write("<br>");
  document.write("End");
  document.write("<br>");
</script>
```

## Output

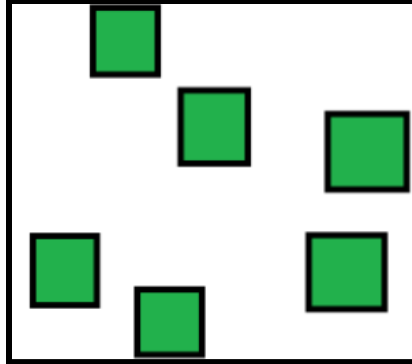
```
Hi
End
Let us see what happens
```

- At first, **Hi statement** will get logged.
- Then, JavaScript passes **setTimeout** function to **web API** and **rest of the code** will be **executed**.
- **After executing** all the code, the **setTimeout** function is pushed to the **call stack** and finally gets **executed**.

## Memory Allocation

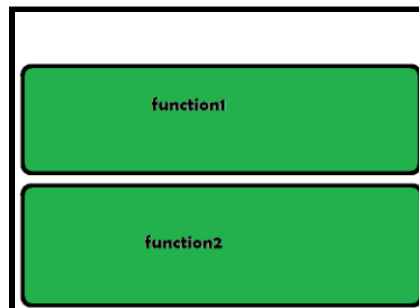
### Heap Memory

- The **data** will be **stored randomly** and **memory** is also allocated in the same manner.



### Stack Memory

- The **memory** will be allocated in the **form of a stack**. In case of **functions**, **stack memory** is used.



## Function call stack

- The **function stack** is a **function** which **keeps track of all the functions** that are executed during the **run time**.
- When an **error** is occurred, we can see a **function stack** being printed at that time.

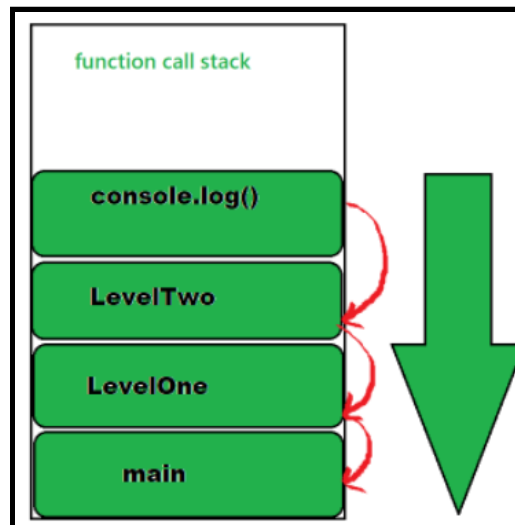
```
function LevelTwo() {
  console.log("Inside Level Two!")
}

function LevelOne() {
  LevelTwo()
}

function main() {
  LevelOne()
}

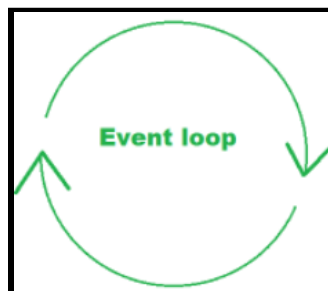
main()
```

- The function gets **popped out of stack** after the function's purpose gets over.

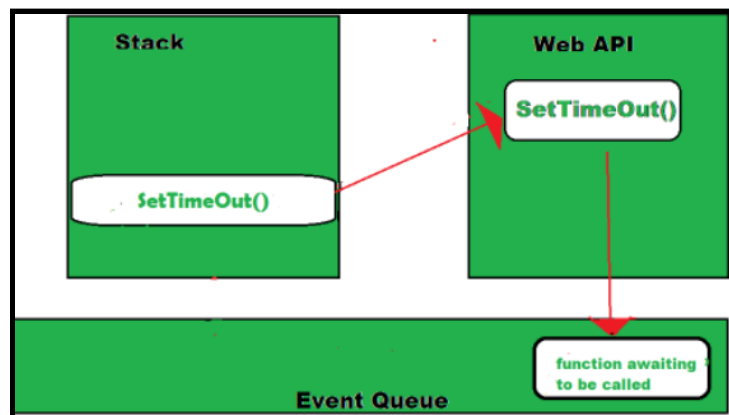


## Event Loop

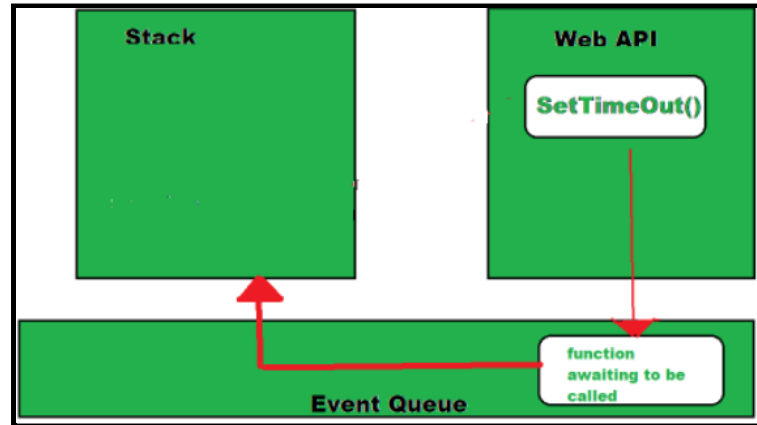
- Whenever a **function stack** is **empty**, the **event loop** pulls the stuff **out of queue** and places it over the **function stack**.



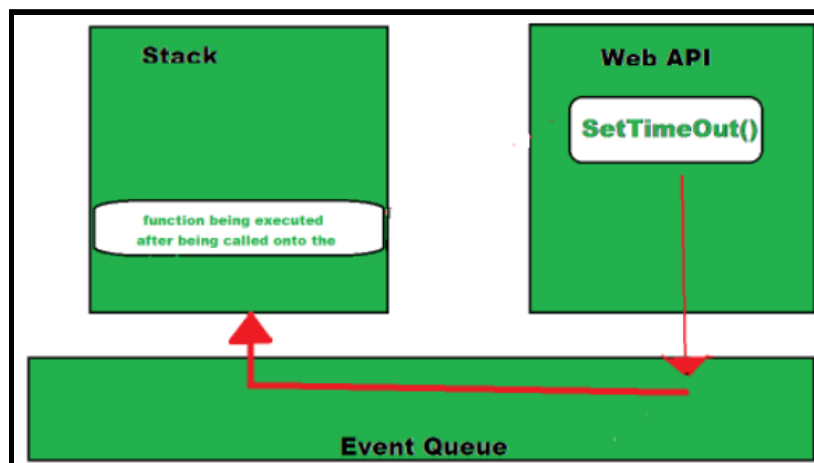
- The event loop gives the illusion of **multithreaded**.



- The **callback** is in the **event queue** is **waiting** for its turn in the stack to run when **setTimeout()** is being **executed**. When the **function stack** becomes **empty**, it is **loaded** to the **stack**.



- The **first event** from the event queue is now being placed on the **stack**. This cycle is called **event loop** and this is how JavaScript handles **events**.



## Callback Hell

- The code with **complex nested callbacks** will cause a **big issue** called **Callback Hell**.
- The **result** of the **previous callback** is taken up by the **upcoming callbacks**.
- The **code structure** will look like a **pyramid**.
- It is **difficult** to **read** and **maintain**.
- If anyone **function** has an **error**, it will **affect** all the **other function**.

## How to avoid callback hell?

- In JavaScript, **event queue** and **promises** help to **escape** from a **callback hell**.
- Any **asynchronous function** will **return an object** called **promise**. A **callback method** can be added to a **promise**.

- **.then()** method is used by **promises** to **call async callbacks**. As many callbacks can be chained together. The **order** of the callbacks is also **strictly maintained**.
- Promise uses
  - **.fetch()** method to **fetch an object** from the network.
  - **.catch()** method to **catch any exception** when any block fails.
- The subsequent JS code **doesn't block** if these **promises** are put in **event queue**. The event queue finishes its operations once the results are returned.
- The keywords and methods like **async**, **wait**, **setTimeout()** are used to **simplify** and make **callbacks used better**.