1 what is the call Stack, and How Does it Operate

The call stack is a data Structure used by JavaScript to manage function execution it Operates on the principles of LIFO whenever a function is invoked, it is added to the top of the Call Stack, and once the function execution is complete, it is removed the stack.

Steps

1. Function Call
2. Execution
3. Completion

Example

```
function first () {
    console.log ("First function");
}
 function Second () {
     first ();
    console.log ("Second function");
}
 Second ();
```

2) what is the Event Loop, and what Role Does it play in Asynchronous Processing

The Event Loop is a mechanism that ensures smooth execution of both synchronous and asynchronous tasks

Role
The Event loop checks if the calls stack is Empty
if it's empty it de queues task from the callback Queue or the Micro task Queue and pushes them onto the call stack for Execution

3 How does Do setTimeout and promises Fit into the Event loop

setTimeout is managed by the Web API
The callback function for SetTimeOut is sent to the callback Queue once the timer expires

The Event loop picks this call back from the callback Queue and pushes it onto the call ~~to~~ Stack for Execution only when the call stack is empty

promises

promises are part of the Microtask Queue.

when a promise is resolved its .then() callback is added to the Microtask Queue

The Event loop processes all Micro tasks before moving on to the call back Queue.

4 what happens when JavaScript Encounters a setTimeOut or a Promise ?

setTimeout Execution Flow

1. setTimeout is called, and the timer starts in Web API

After the timer Expires the call back is

added to the Callback Queue
The Event loop waits Until the call Stack
is empty then Pushes the callback onto
the Call Stack

promise Execution Flow
A promise Execution Flow
A promise is created and begins execution
Synchronously
when resolved or rejected its .then()
or .catch() call backs are added to
the Micro task Queue
the Event Loop ensures that all Micro tasks
are processed before any task in the
Call back Queue

5 Example

```
console.log ("start");
setTimeout (() => {
        console.log ("Timeout callback");
}, 0);

promise.resolve().then (() => {
    console.log ("Promise, call back');
});
    console.log ("End");
```