#DailyDevDrop

WHAT IS EXECUTION CONTEXT & CALL STACK?

(IN 2 MINUTES)

Manish Kumar

Swipe

#DailyDevDrop

- **Execution Context** is the environment in which JavaScript code runs.
- Each time JavaScript runs code, it creates an execution context with variables, functions & `this` value.
- There are 3 types:
  - **Global Execution Context** – Created when JavaScript code starts executing.
  - **Function Execution Context** – Created every time a function is called.
  - **Eval Execution Context** – Created when `eval()` is used (rare).

Manish Kumar

Swipe

#DailyDevDrop

- **Call Stack** is a stack-based mechanism that tracks function calls in JavaScript.
- When a function is called:
  - A new **Function Execution Context** is created.
  - It is **pushed**
- When the function finishes execution:
  - Its context is **popped** off the stack.
- This helps JavaScript keep track of **which function is currently running** and what to return control to after it's done.
- Call Stack Overflow occurs when the Call Stack exceeds its maximum size.

Manish Kumar

Swipe

#DailyDevDrop

Code:

```
function first() {
  console.log("Inside first");
  second();
  console.log("Back in first");
}

function second() {
  console.log("Inside second");
}

console.log("Start");
first();
console.log("End");
```

Output:

```
Start
Inside first
Inside second
Back in first
End
```

JS

Manish Kumar

Swipe

#DailyDevDrop 05/08

✴️ Explanation 💡

- GEC is created. **(Prints: "Start")**
- `first()` runs → added to Call Stack. **(Prints: "Inside first")**
- `second()` runs → added to Call Stack. **(Prints: "Inside second")**
- `second()` ends → removed from stack.
- Back in `first()`. **(prints: "Back in first")**
- `first()` ends → removed from stack. **(Prints: "End")**
- GEC removed when done.

Manish Kumar

Swipe ←

#DailyDevDrop 06/08

✷ Interview Questions on EC & CS

- What is an Execution Context in JavaScript?
- What are the types of Execution Context?
- What is the Global Execution Context (GEC)?
- How does the Call Stack work?
- What happens to the Call Stack when a function finishes?
- How are Execution Contexts pushed and popped from the Call Stack?
- What causes a Call Stack overflow?

Manish Kumar

Swipe ←

What will be the output?

Code:

```
function greet(name) {
  return `Hello, ${name}`;
}

function greetUser(user) {
  const message = greet(user);
  console.log(message);
}

function startApp() {
  greetUser("Manish");
}

startApp();
```

Output:

```
// 🔍 Challenge: Can you predict the output of the following code?

// Leave your guess in the comments 👇
```

JS

Manish Kumar

Swipe

WAS THIS HELPFUL?

Let Me Know Your thoughts in the comments Below

Manish Kumar

Save