1. How JavaScript Works 🔥& Execution Context

📘 Index (Table of Contents)

❓ Questions and Answers

1. What is JavaScript Execution?

Q: How does JavaScript execute code?

A: JavaScript executes code in a single-threaded, synchronous manner. It processes code line by line, creating execution contexts for each function call.

Analogy: Think of JavaScript as a chef in a kitchen. The chef can only handle one task at a time, such as chopping vegetables or stirring a pot. Similarly, JavaScript processes one task at a time in a single thread.

---

2. What is an Execution Context?

Q: What is an execution context in JavaScript?

A: An execution context is a conceptual environment where the JavaScript code is evaluated and executed. It contains information about the variable environment, scope chain, and `this` value.

Example: When a function is called, a new execution context is created, which includes the function's parameters, local variables, and the `this` value.

---

3. How does the Call Stack work?

Q: What is the call stack in JavaScript?

A: The call stack is a data structure that keeps track of function calls. When a function is called, its execution context is pushed onto the stack. Once the function finishes executing, its context is popped off the stack.

Analogy: Imagine a stack of plates in a cafeteria. When a new plate is added, it's placed on top. When a plate is used, it's removed from the top. Similarly, functions are added and removed from the call stack in a last-in, first-out (LIFO) order.

---

4. What are the Phases of Execution?

Q: What are the two phases of JavaScript execution?

A: JavaScript execution occurs in two phases:

- **Memory Creation Phase:** During this phase, the JavaScript engine allocates memory for variables and functions. Variables are initialized with `undefined`, and functions are stored in memory.
- **Code Execution Phase:** In this phase, the JavaScript engine executes the code line by line, assigning values to variables and executing functions.

Example: Consider the following code:

```
console.log(a);
var a = 5;
```

In the memory creation phase, `a` is declared and initialized with `undefined`. In the code execution phase, `a` is assigned the value `5`.

---

5. What is Hoisting?

Q: What is hoisting in JavaScript?

A: Hoisting is JavaScript's behavior of moving declarations to the top of their containing scope during the compile phase. This means variables and functions can be used before they are declared.

Example: In the following code, `a` is hoisted:

```
console.log(a);
var a = 5;
```

Even though `a` is declared after the `console.log`, JavaScript hoists the declaration, so the code doesn't throw an error. However, `a` is `undefined` until it is assigned the value `5`.

6. What is the Global Execution Context?

Q: What is the global execution context?

A: The global execution context is the default context in which any JavaScript code runs. It is created when the JavaScript environment is initialized and remains active as long as the script is running.

Analogy: Think of the global execution context as the main stage in a theater. All other performances (function calls) happen on this stage, and the stage remains set up throughout the play.

7. What is the Function Execution Context?

Q: What happens when a function is called?

A: When a function is called, a new execution context is created for that function. This context contains information about the function's parameters, local variables, and the `this` value.

Example: Consider the following code:

```
function greet(name) {
  console.log("Hello, " + name);
}
greet("Alice");
```

When `greet("Alice")` is called, a new execution context is created for the `greet` function, with `name` set to `"Alice"`.

8. How does the Event Loop work?

Q: What is the event loop in JavaScript?

A: The event loop is a mechanism that allows JavaScript to handle asynchronous operations. It continuously checks the call stack and the message queue. If the call stack is empty, it pushes the first message from the queue to the stack for execution.

Analogy: Imagine a single-lane road where cars (tasks) are waiting in a queue. The traffic light (call stack) can only let one car pass at a time. Once the road is clear, the next car in the queue is allowed to pass.

🔑 Summary and Key Takeaways

- **Single-Threaded Execution:** JavaScript executes code in a single thread, processing one task at a time.
- **Execution Contexts:** Each function call creates a new execution context, which contains information about the function's parameters, local variables, and the `this` value.
- **Call Stack:** The call stack keeps track of function calls, adding new contexts when functions are called and removing them when they finish executing.
- **Phases of Execution:** JavaScript code is executed in two phases: memory creation and code execution.
- **Hoisting:** Declarations are hoisted to the top of their scope, allowing variables and functions to be used before they are declared.
- **Global Execution Context:** The global execution context is the default context in which any JavaScript code runs.
- **Function Execution Context:** Each function call creates a new execution context with its own scope.
- **Event Loop:** The event loop allows JavaScript to handle asynchronous operations by checking the call stack and message queue.