JavaScript Execution Context Notes

// Jab koi JS likhte hai to ek Global Execution Context banta hai

// JS single-threaded hai

// Two types of Execution Contexts:

// 1. Global Execution Context

// 2. Functional Execution Context

// In dono phase ke completion ke baad niche wale phase run hote hai

// Code do phase me chalta hai:

// 1. Memory Creation Phase

// 2. Execution Phase

// Browser ke andar 'this' ka value window object hota hai

// Iss code se sikhenge kaise code run hota JS me

```
let num1 = 10;
let num2 = 5;
const sum = function(num1, num2) {
return num1 + num2;
}
let result1 = sum(num1, num2);
let result2 = sum(19, 20);
```

// Step-by-step Explanation:

1. Global Execution Context create hota hai, jisme 'this' ka value window object hota hai.
2. Memory Allocation Phase:

- num1, num2 => undefined
- sum => function definition stored
- result1, result2 => undefined

3. Execution Phase:

- num1 = 10, num2 = 5 assign hota hai
- result1 execute hota hai => sum function call hota hai

4. Function Execution Context create hota hai:

- Memory Phase: num1, num2 (parameters) => undefined
- Execution Phase: arguments pass hota hai (num1 = 10, num2 = 5)
- return value => result1 me assign

5. Same process for result2 with new Function Execution Context

Execution Context Diagram (Illustrative)

```
[ Global Execution Context ]

-----------------------------------------

| this -> window                  |

| num1 -> undefined               |

| num2 -> undefined               |

| sum -> function definition      |

| result1 -> undefined            |

| result2 -> undefined            |

-----------------------------------------


-> Execution Phase Starts


| num1 = 10

| num2 = 5

| result1 = sum(10, 5)


=> [Function Execution Context - sum()]

-----------------------------------------

| Arguments: num1 = 10, num2 = 5      |

| Return: 15                      |

-----------------------------------------
```

Same for result2 with values 19 and 20.