DAY – 2

Functions • Parameters • Spread/Rest • Return

Collision

Two values clash.

Syntactic Sugar

Shortcut writing style that makes code simpler, but does not add new functionality.

Implicit

JavaScript does something automatically without you writing it clearly.

Implicit Type Conversion (Type Coercion)

```
console.log("5" - 2);   // 3 → "5" becomes 5
console.log("5" + 2);   // "52" → number becomes string
```

Implicit Return in Arrow Functions

```
const add = (a, b) => a + b;
```

Implicit Boolean Conversion

Falsy: 0, "", null, undefined, NaN, false

Truthy & Falsy Values

Truthy Values

- All non-zero numbers
- All non-zero BigInt values
- All objects
- All functions
- Boolean true

Falsy Values

- false
- 0
- -0
- 0n
- ""
- null
- undefined
- NaN

Notation

Types of Notation

Dot Notation

```
person.name
```

Bracket Notation

```
person["name"]
```

Functions

Function Declaration

- Uses `function` keyword
- Fully hoisted (can call before definition)

Example:

```
console.log(add()); // Works

function add() {
  return 10 + 20;
}
```

Function Expression / Anonymous Function

- Function stored in a variable
- Not hoisted

Example:

```
console.log(add()); // Error
```

```
const add = function() {
  return 10 + 20;
};

console.log(add()); // Works
```

Arrow Functions

- Short syntax
- Uses =>
- Implicit return in single line
- Not hoisted
- No own this/arguments

Example:

```
const add = (a, b) => a + b;
```

Spread Operator (…)

Use Cases

1. Combine Arrays

```
const arr1=[1,2,3];
const arr2=[4,5];
const combined=[...arr1, ...arr2];
```

2. Copy Arrays

```
const a=[10,20,30];
const copy=[...a];
```

3. Convert String → Array

```
console.log([..."Jagan"]);
```

4. Expand Values Inside Functions

```
function add(a,b,c){ return a+b+c;}
console.log(add(...[10,20,30]));
```

Parameters & Arguments

Parameters

Variables inside function definition.

```
function add(a,b){}
```

Arguments

Actual values passed.

```
add(10,20);
```

Rest Operator (…)

Why Use Rest?

- Collect unlimited arguments
- Remaining array items
- Remaining object properties

1. Unlimited Arguments

```
function sum(...nums){ return nums.reduce((a,b)=>a+b); }
```

2. Remaining Array Items

```
const [a, ...rest] = [1,2,3,4];
```

3. Remaining Object Props

```
const {name, ...others} = {name:"Jagan", age:22, city:"Chennai"};
```

Tasks:-

What is a Callback?

A **callback** is a **function passed as an argument to another function**, so that the other function can **call it later**.

1. To handle asynchronous operations

Examples:

- Fetching data from an API
- Reading files
- Timers (setTimeout)
- Database queries

To control the order of execution (avoid blocking the program)

Example:
If you read a file from disk, instead of stopping the entire program, JavaScript continues the next lines and uses a callback to notify when file reading is complete.

To reuse logic (flexibility)

You can pass different callback functions to customize the behavior.

```
function greet(name, callback) {

   console.log("Hello " + name);
```

```
    callback();  // calling the callback function
}
function sayBye() {
    console.log("Bye!");
}
greet("Jagath", sayBye);
```