==========

OBJECT-ORIENTED JAVASCRIPT (OOJS)

==========

🔹 Definition:

OOJS (Object-Oriented JavaScript) is a programming style
where code is organized around objects (data + behavior).

Objects represent real-world things.
Example: A Car object has brand, color (properties) and start() (method).

1️⃣ Ways to create "classes" in JS

A) Pre-ES6: Constructor functions + prototypes
B) ES6 and later: class keyword (syntactic sugar)

2️⃣ Implementing Class (Prototype method)

```
function Car(brand, color) {
  this.brand = brand;
  this.color = color;
}

// Prototype methods (shared by all objects)
Car.prototype.start = function() {
  console.log(`${this.brand} started!`);
};

Car.prototype.showDetails = function() {
  console.log(`Brand: ${this.brand}, Color: ${this.color}`);
};

// Creating objects
const car1 = new Car("Tesla", "Red");
car1.start();
```

3️⃣ Static methods and properties (pre-ES6)

- Static members belong to the class, not to its instances.
- Define them directly on the constructor function.

```
Car.category = "Vehicle";
```

```
Car.compare = function(a, b) {
  return a.brand === b.brand ? "Same brand!" : "Different brands.";
};

console.log(Car.category);
console.log(Car.compare(car1, car2));

----------------------------------------
4️⃣ Property declaration
----------------------------------------
- Instance properties are declared inside the constructor using `this`.
- Methods are declared on the prototype so they are shared.

function Person(name, age) {
  this.name = name;
  this.age = age;
}

Person.prototype.greet = function() {
  console.log(`Hi, I'm ${this.name}, ${this.age} years old.`);
};

----------------------------------------
5️⃣ ES6 equivalent (for understanding)
----------------------------------------
class Car {
  constructor(brand, color) {
    this.brand = brand;
    this.color = color;
  }

  start() { ... }

  static category = "Vehicle";
  static compare(a, b) { ... }
}
```

==========

ES6 CONCEPTS – IMPORT/EXPORT, ASYNC/AWAIT, CLASSES

==========

🔹 1️⃣ Import and Export (Modules)

Purpose: Split JS into multiple files for organization and reusability.

A) Named export

File: math.js

```
export const PI = 3.14;
export function add(a, b) { return a + b; }
```

Import:

```
import { PI, add } from "./math.js";
```

B) Default export

File: greet.js

```
export default function greet(name) {
  console.log(`Hello, ${name}!`);
}
```

Import:

```
import greet from "./greet.js";
```

You can mix:

```
import greet, { PI } from "./math.js";
```

🔹 2️⃣ Async / Await

- async marks a function as asynchronous (it returns a Promise).
- await pauses execution until the Promise is resolved.

Example:

```
async function fetchData() {
  try {
    const res = await fetch("https://api.example.com");
    const data = await res.json();
    console.log(data);
  } catch (err) {
    console.error(err);
  }
}
```

🔹 3️⃣ Classes

- Classes are blueprints for creating objects.

Syntax:

```
class Car {
  constructor(brand, color) {
    this.brand = brand;
    this.color = color;
  }

  start() {
    console.log(`${this.brand} started!`);
  }

  static info() {
    console.log("Static method on class.");
  }
}
```

Usage:

```
const car1 = new Car("Tesla", "Red");
car1.start();
Car.info(); // static method
```

==========
STRING, MATH & DATE METHODS – JAVASCRIPT
==========

1️⃣ STRING METHODS

---

length → number of characters
toUpperCase(), toLowerCase()
trim(), trimStart(), trimEnd()
includes(substring)
indexOf(), lastIndexOf()
slice(start, end)
replace(), replaceAll()
split(delimiter)
startsWith(), endsWith()

Use cases:
- Input validation
- Search

- Text formatting
- Parsing data

2️⃣ MATH OBJECT

Math.round(x)
Math.floor(x)
Math.ceil(x)
Math.random()
Math.max(a, b, c)
Math.min(a, b, c)
Math.pow(a, b) or a ** b
Math.sqrt(x)
Math.abs(x)

Math is static → no new keyword

3️⃣ DATE & TIME
---
new Date() → current date/time
new Date("YYYY-MM-DD")

Get values:
getFullYear(), getMonth(), getDate()
getDay(), getHours(), getMinutes(), getSeconds()

Set values:
setFullYear(), setMonth(), setDate()

Timestamp:
Date.now()

Difference:
date2 - date1 → milliseconds

Formatting:
toDateString()
toTimeString()
toLocaleDateString()
toLocaleTimeString()

==========