OOPs - objects

A programming style that is centered around objects rather than functions.

```
const student1 = {
    name: "John",
    class: "5",
    info: () => console.log("I'm John from class 5"),
}

const student2 = {
    name: "Cathy",
    class: "5",
    info: () => console.log("I'm Cathy from class 7"),
}
console.log(student1.name)
console.log(student1.info())
```

OOPs - class

```
class Student {
    constructor (name, class) {
        this.name = name
        this.class = class
    }
    info = () => console.log(this.name,this.class)
}
const student1 = new Student("John", 5)
```

OOPs - Inheritance

```
class Company {
      constructor(dept) {
        this.dept = dept;
      }
      getCompany() {
        return "ABC Inc";
      }
    }
    class Employee extends Company {
      constructor(dept, name) {
        super(dept);
        this.name = name;
      }
      show() {
        return "I am " + this.name + " from " + this.dept + " department," +
this.getCompany()
      }
    }
    let emp1 = new Employee("HR", "John");
    console.log(emp1.show())
    let emp2 = new Employee("IT", "Cathy");
    console.log(emp2.show())
```

OOPs - Encapsulation

```
class Employee {
      #name = "";
      get name() {
        return this.#name;
      }
      set name(value) {
          this.#name = value;
      }
    }
    const emp1 = new Employee();
    emp1.name = "John";
    console.log(emp1.name);
    //console.log(#name)
```

OOPs - Polymorphism

```
class Shape {
        draw() {
            console.log("Draw Shape")
        }
    }
    class Square extends Shape{
        draw() {
            console.log("Draw Square")
        }
    }
    let sqr = new Square()
    sqr.draw()
    let shp = new Shape()
    shp.draw()
```

OOPs - Abstraction

```
class Circle {
      constructor(length, width) {
        this.length = length;
        this.width = width;
      }

      area() {
        console.log(this.length * this.width);
      }
    }
    let circle = new Circle(4, 5);
    circle.area();
```