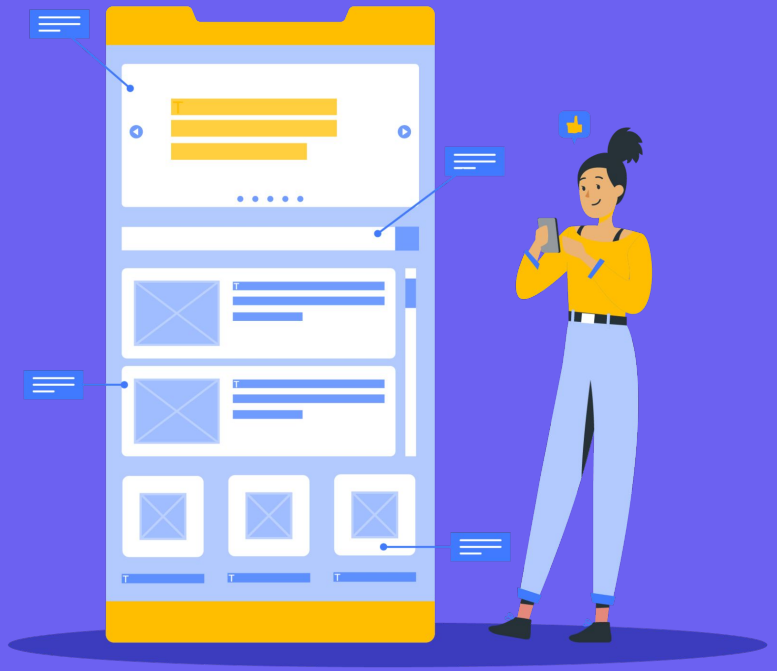
OOPs: Intro to OOPs in JS



Relevel
by Unacademy

Objects in Js

Object is an entity with properties and methods associated with it.

```
// Creating object using Object literal

const laptop = {
  make: 'Apple',
  model: 'MacBook Pro',
  memory: ['SSD', 'HDD'],
  cores: 8,
  memorySize: [256, 512],
};
```

```
// Creating object using new keyword
const laptop = new Object();;
laptop.make = 'Apple';
laptop.model = 'MacBook Pro';
laptop.cores = 8;
```

```
// Creating object using Object literal

const laptop = {
  make: 'Apple',
  model: 'MacBook Pro',
  memory: ['SSD', 'HDD'],
  cores: 8,
  memorySize: [256, 512],
};
```

Constructors

The **constructor** method/function is a special method of a class invoked for creating and initialising an object instance of that class.

```
// Default constructor
function Laptop() {
    this.make = 'Dell';
    this.model = 'Inspiron 1590';
    this.cores = '8';
}

const myLaptop = new Laptop();
```

this keyword

Generally, when we try to access variables/objects/functions through different scopes, we use the **"this"** keyword to access them.

```
const nakama = {
    name: 'Luffy',
    place: 'Foosha Village',
    getBio: function(){
        console.log(this.name + ' lives in ' + this.place)
    }
}
nakama.getBio(); // Luffy lives in Foosha Village
```

Accessing properties of objects in JS

- The most common way to access an object is using the '.' (dot) operator.
- Another way is similar to how to access array elements
- We can also retrieve the entire object laptop by logging in on the console.

```
const laptop = {
  make: 'Apple',
  model: 'MacBook Pro',
  memory: ['SSD', 'HDD'],
  cores: 8,
  memorySize: [256, 512],
};

console.log(laptop.make);
console.log(laptop.model);
console.log(laptop.memory);
console.log(laptop.memorySize);
console.log(laptop.cores);
```

```
const laptop = {
  make: 'Apple',
  model: 'MacBook Pro',
  memory: ['SSD', 'HDD'],
  cores: 8,
  memorySize: [256, 512],
};

console.log(laptop['make']);
console.log(laptop["model"]);
```

```
const laptop = {
  make: 'Apple',
  model: 'MacBook Pro',
  memory: ['SSD', 'HDD'],
  cores: 8,
  memorySize: [256, 512],
};

console.log('Laptop specifications:');
console.log(laptop);
```

Classes

A class can be defined as the template to create objects.
In JS classes are built on prototypes.
The syntax of classes has two components:

1. Class Expression
2. Class Declaration

```
class Circle{
    static shape;

    static {
        this.shape = "Circle";
    }

    //static Method
    static circumference(radius){
        return 2 * Math.PI * radius;
    }

    constructor(radius, ...generatorInput) {
        this.radius = radius;
    }

    //Getter
    get area(){
        return this.calArea();
    }

    //Method
    calArea(){
        return Math.PI * this.radius *this.radius;
    }

}

const circle = new Circle(5);
console.log(circle.shape); // undefined
console.log(Circle.shape); // Circle
console.log(circle.circumference); // undefined
console.log(Circle.circumference(5)); //31.41592653589793
console.log(circle.area); // 78.53981633974483

const circle1 = new Circle(10, ...[1,2,3]);
console.log(circle1.shape); // undefined
console.log(Circle.shape); // Circle
console.log(circle1.area); //314.1592653589793
```

Prototypes

It is the mechanism by which one object can inherit from another in Js. To provide inheritance, all objects in Js have a prototype object - that is, like a template object from which methods and properties are inherited.

```
function Vehicle(brand, model, price, speed, wheel){
    this.name = {
        'brand': brand,
        'model': model
    };
    this.price = price;
    this.speed = speed;
    this.wheel = wheel;
}


Vehicle.prototype.feature = function() {
    console.log(this.name.brand + " " + this.name.model + " has power steering")
}
```

Inheritance

Inheritance is defined as reuse of some methods or properties from some other class to a child class to increase code reusability, give a well defined structure to the code.

```
class Bike extends Vehicle{
   constructor(brand, model, price, speed, wheel, centerOfGravity){
       super(brand, model, price, speed, wheel);
       this._centerOfGravity = centerOfGravity;
   }

   get centerOfGravity() {
       return this._centerOfGravity;
   }

   set centerOfGravity(newCenterOfGravity){
       this._centerOfGravity = newCenterOfGravity;
   }
}
```

Practice/HW

1. Program to demonstrate default and parameterized constructor.
2. Program to demonstrate different ways of creating objects.

Thank you