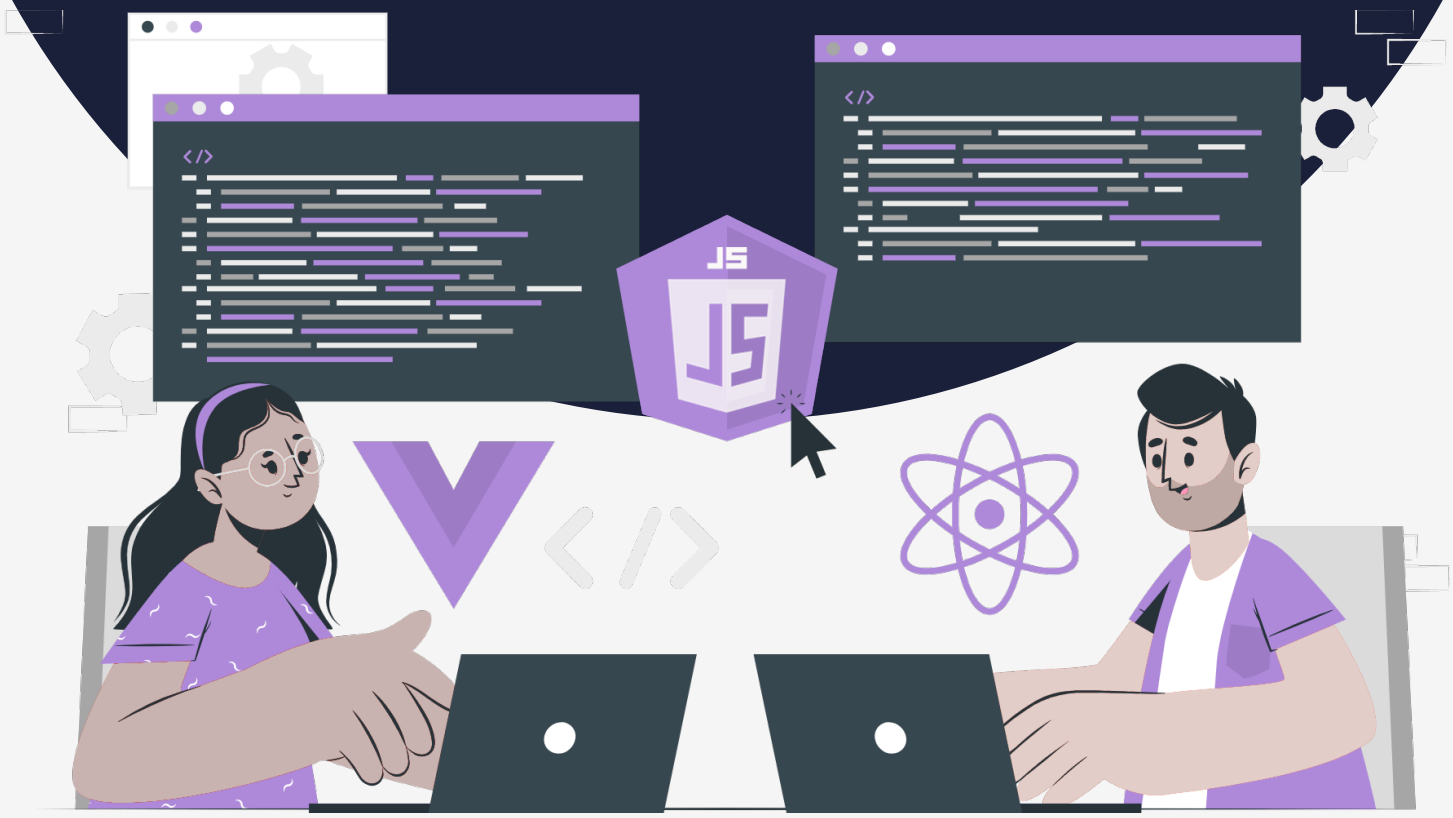
Lesson:

Defining a class, Class Instantiation



List of concepts:

1. What is a class
2. Examples
3. Class instantiation
4. Example

What is a class

Classes serve as a blueprint for creating objects, providing a way to organize and structure code. They allow you to define objects with similar properties and methods in a reusable manner, reducing the amount of code you need to write and making your code easier to maintain.

Here are some of the reasons why you might want to use classes in JavaScript:

Object-Oriented Programming: Classes provide a way to do object-oriented programming in JavaScript, which is a popular programming paradigm for organizing code into objects and classes.

Reusability: Classes provide a way to create multiple objects with similar properties and methods, reducing the amount of code you need to write.

Abstraction: Classes provide a way to abstract away the implementation details of objects and hide the complexity of your code. This makes your code more readable and easier to understand.

Encapsulation: Classes provide a way to encapsulate data and behavior within objects, making it easier to manage state and behavior in a structured way.

Modularity: Classes provide a way to organize code into distinct, reusable units, making it easier to manage and maintain large codebases.

Overall, classes in JavaScript are useful tools for organizing and structuring code, making it easier to write, understand, and maintain.

In JavaScript, classes were introduced in ECMAScript 6 (ES6). A class is defined using the class keyword, followed by the class name, and a block of code that defines the properties and methods of the class.

Here is an example of a class definition in JavaScript:

```
class Animal {
  makeSound() {
    console.log("Animal sound");
  }
}

const myAnimal = new Animal();
myAnimal.makeSound();
```

In this example, a Car class is defined that takes make, model, and year parameters. The class also has a honk method that logs a message to the console. An object of the Car class is created with the new operator and assigned to the myCar variable, and its properties and method can be accessed using dot notation.

Example: Demonstrate an example of class definition

```
class Car {
  honk() {
    console.log("Beep Beep!");
  }
}

const myCar = new Car();
myCar.honk();
```

In this example, the Car class is defined. It has a single method honk that logs a message to the console. An object of the Car class is created using the new operator and assigned to the myCar variable, and its method can be accessed using dot notation.

Example : Let us try adding some data in the same car class.

```
class Car {
  setMake(make) {
    this.make = make;
  }
  setModel(model) {
    this.model = model;
  }
  setYear(year) {
    this.year = year;
  }

  honk() {
    console.log("Beep Beep!");
  }
}

const myCar = new Car();
myCar.setMake("Toyota");
myCar.setModel("Camry");
myCar.setYear(2020);
console.log(myCar.make);
console.log(myCar.model);
console.log(myCar.year);
myCar.honk();
```

In this example, the Car class is defined, it has three methods setMake, setModel, and setYear for setting the make, model, and year properties, and a method honk for honking the horn. An object of the Car class is created using the new operator and assigned to the myCar variable, and its methods can be accessed using dot notation.

Class Instantiation

Class instantiation in JavaScript refers to creating an object from a class. An object for a class is a kind of primitive data type that shows the behavior and the characteristics of a class. This is done using the 'new' operator, which creates an instance of the class and returns the object.

When the new keyword is used, it creates an instance of an object. It does so by following these steps:

- It creates a new, empty object.
- It sets the prototype property of the newly created object to be the prototype property of the constructor function being invoked with the new keyword.
- It binds the 'this' keyword within the constructor function to the newly created object.

The 'this' keyword refers to the object that is currently executing the code. Its value depends on how a function is called, and it can be used to access properties and methods on the current object.

If the constructor function doesn't explicitly return an object, then the new keyword returns the newly created object.

Let us see the example below for better understanding:

Example :

```
class Person {
  setName(name) {
    this.name = name;
  }
  setAge(age) {
    this.age = age;
  }
  introduce() {
    console.log(`Hi, my name is ${this.name} and I am ${this.age} years old.`);
  }
}

const myPerson = new Person();
myPerson.setName("Rohan");
myPerson.setAge(30);
myPerson.introduce();
```

In this example, the Person class has two methods setName and setAge for setting the name and age properties, and a method introduce for introducing the person. An object of the Person class is created using the new operator and assigned to the myPerson variable, and its methods can be accessed using dot notation.

A class may also be instantiated as expression. Let us look at the example below to know more :

```
const MyClass = class {
  constructor(expression) {
    this.result = expression;
  }
};

const myInstance = new MyClass(10 + 2);
console.log(myInstance.result);
```

Here, the class expression is assigned to the constant MyClass. The expression 10 + 2 is then evaluated and used as the argument to the new keyword, effectively instantiating an instance of the MyClass class. The value of the expression is stored as a property on the instance, myInstance, which can be accessed using the .result property.