Classes in JS:

Primitive types:

1. number
2. string
3. boolean

Complex types:

1. Objects
2. Arrays

Classes:

In JavaScript, classes are a way to define blueprints for creating objects (these objects are different from the objects defined in the last section).
For example:

```
class Rectangle {
    constructor(width, height, color) {
         this.width = width;
         this.height = height;
         this.color = color;
 }
 area() {
    const area = this.width * this.height;
    return area;
 }
}

 const rect = new Rectangle(2,4)
 const area = rect.area()
 console.log(area);
```

Key Concepts:

1. Class Declaration:
    - You declare a class using the class keyword.
    - Inside a class, you define properties (variables) and methods (functions) that will

2. Constructor:
    - A special method inside the class that is called when you create an instance (an object) of the class.
    - It's used to initialize the properties of the object.

3. Methods:
    - Functions that are defined inside the class and can be used by all instances of the class.

4. Inheritance:
    - Classes can inherit properties and methods from other classes, allowing you to create a new class based on an existing one.

5. Static Methods:
    - Methods that belong to the class itself, not to instances of the class. You call them directly on the class.

6. Getters and Setters:
    - Special methods that allow you to define how properties are accessed and modified.

Some more classes:

The above Rectangle class is the class built by user. But JS provides us some inbuilt classes. Some of them are:

Date: In JavaScript, date objects are created with new Date().
new Date() returns a date object with the current date and time.

Date Get Methods

Method	Description
getFullYear()	Get **year** as a four digit number (yyyy)
getMonth()	Get **month** as a number (0-11)
getDate()	Get **day** as a number (1-31)
getDay()	Get **weekday** as a number (0-6)
getHours()	Get **hour** (0-23)
getMinutes()	Get **minute** (0-59)
getSeconds()	Get **second** (0-59)
getMilliseconds()	Get **millisecond** (0-999)
getTime()	Get **time** (milliseconds since January 1, 1970)

```
// Date class in Js

const now = new Date();         //current date and time
console.log(now.toISOString());    // Outputs the date
in ISO format.
```

Maps: A Map holds key-value pairs where the keys can be any datatype.
A Map remembers the original insertion order of the keys.
typeof returns object: ie. Maps are object.

```
//Maps class in JS

const map = new Map();
map.set('name', 'Raj');
map.set('age', 21);
console.log(map.get('name'));
```

Object	Map
Not directly iterable	Directly iterable
Do not have a size property	Have a size property
Keys must be Strings (or Symbols)	Keys can be any datatype
Keys are not well ordered	Keys are ordered by insertion
Have default keys	Do not have default keys

The Promise Class:

The Promise class gives u a promise, that I will return u something in the future.
"I Promise a Result!"
"Producing code" is code that can take some time
"Consuming code" is code that must wait for the result
A Promise is an Object that links Producing code and Consuming code

```
function logName(){
    console.log('Rajshree');
}
setTimeout(logName, 1000);

 // the setTimeOut function promises to
call the logName function after 1 sec
```

You can either use async approach using
Callback
Or Promises.

Promises are just cleaner way than using callback.

 Calling a promise is easy, defining your own promise is where things get hard

A promise in JavaScript is in an object that represents the eventual completion (or failure) of an asynchronous operation and its resulting value. Promises are used to handle asynchronous operations more effectively than traditional callback functions, providing a cleaner and more manageable way to deal with code that executes asynchronously, such as API calls, file I/O, or timers.

Using a function that returns a promise:

Ignore the function definition of setTimeoutPromisified for now

```
function setTimeoutPromisified(ms){
          return    new    Promise(resolve    =>
setTimeout(resolve,ms));
}
function callback(){
    console.log("3 seconds have passed...!");
}
setTimeoutPromisified(3000).then(callback);
```

Promisified Version

```
setTimeoutPromisified(3000).then(callback);
```

Callback version

```
settimeout(callback,3000)
```