Defining-a-Class.js

#JavaScript-with-JC

```
// 💡 Two ways to define a class in javascript

// 👉 1) Let's take an example Class Declaration
class Person {
  // special method for creating and initializing an object
  constructor(name, age) {
    // instance members ( created for each object separately )
    this.name = name;
    this.age = age;
  }

  // prototype members ( created in Person.prototype common sharable with all object )
  getInfo() {
    console.log("name", this.name, "age", this.age);
  }
}

const jayesh = new Person("jayesh", 24);
jayesh.getInfo(); // name jayesh age 24

// 👉 2) Let's take an example Class Expression
// unnamed class expression
const Person1 = class {
  constructor(name, age) {
    // instance members ( created for each object separately )
    this.name = name;
    this.age = age;
  }
};

const sam = new Person("sam", 24);
sam.getInfo(); // name sam age 24

// named class expression
const Person2 = class PersonClass {
  constructor(name, age) {
    // instance members ( created for each object separately )
    this.name = name;
    this.age = age;
  }
};

const john = new Person("john", 24);
john.getInfo(); // name john age 24
```

(Jayesh Choudhary)

Class-Hoisting.js

#JavaScript-with-JC

```
// 💡 Class Hoisting

// Class Declaration hoisting ? => NO

const Ford = new Car("black");
// Uncaught ReferenceError: Cannot access 'Car' before initialization

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

// Class Expression hoisting ? => NO

const Ferrari = new Car("red");
// Uncaught ReferenceError: Cannot access 'Car' before initialization

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

(Jayesh Choudhary)

Class-constructors.js

#JavaScript-with-JC

```
// 💡 Two types of Constructor

// 👉 1) Let's take an example of default constructor in Class Declaration
class Animal1 {
  // If we do not specify any constructor default constructor will be used

  // prototype member
  display() {
    console.log("prototype member");
  }
}

const cat = new Animal1();
cat.display();


// 👉 2) Let's take an example of parameterized constructor in Class Declaration
class Animal2 {
  //  parameterized constructor overriding default constructor
  constructor(color) {
    // instance member
    this.color = color;
  }

  // prototype members
  display() {
    console.log("color", this.color);
  }
}

const dog = new Animal2("black");
dog.display(); // color black
```

(Jayesh Choudhary)

Class-inheritance.js

#JavaScript-with-JC

```
// 💡 Class Inheritance

class Parent {
  constructor(name, age) {
    // instance members
    this.name = name;
    this.age = age;
  }

  // prototype member
  getInfo() {
    console.log("name", this.name, "age", this.age);
  }
}

const sachin = new Parent("sachin", "60");
sachin.getInfo(); // name sachin age 60

class Child extends Parent {
  // case 1 :- if we don't specify constructor then default constrouctor
  //           calls super constructor automatically

  // case 2 :- if we specify parameterized constructor then need
  //           to call super constructor at the starting
  constructor(name, age) {
    super(name, age); // similar to Parent.call(this, name, age);
  }
}

const sachinBaby = new Child("sachinBaby", "28");
sachinBaby.getInfo(); // name sachinBaby age 28 ( Child inheriting the properties of Parent )
```

(Jayesh Choudhary)

Method-Overriding.js

#JavaScript-with-JC

```
// 💡 Class Inheritance Method Overriding

class Father {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }
  getInfo() {
    console.log("father Info", this.name, this.age);
  }
}

const daddy = new Father("daddy", 70);
daddy.getInfo(); // father Info daddy 70

class Son extends Father {
  constructor(name, age, sport) {
    super(name, age);
    this.sport = sport;
  }
  // overriding the method getInfo() of Parent class
  getInfo() {
    console.log("Son Info", this.name, this.age, "plays", this.sport);
  }
}

const babby = new Son("babby", 10, "cricket");
babby.getInfo(); // Son Info babby 10 plays cricket
```

(Jayesh Choudhary)

Static-Methods-and-Properties.js

#JavaScript-with-JC

```
// 💡 Class Static Methods and Properties ( created once and only access by class )

class Student {
  static school = "little flower";
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }

  // can be used as utility functions
  static ageDifference(a, b) {
    return a - b;
  }

  // object or intance method
  displaySchool() {
    // can't access static members
    console.log(this.school);
  }

  // class method
  static displaySchoolName() {
    console.log(this.school);
  }
}

const stud1 = new Student("jonh", 15);
const stud2 = new Student("sam", 13);

// static method or class method
const ageDiff = Student.ageDifference(stud1.age, stud2.age);
console.log(ageDiff); // 2

Student.displaySchoolName(); // little flower
stud1.displaySchool(); // undefined
stud2.displaySchool(); // undefined
```

(Jayesh Choudhary)

Private-Properties.js

#JavaScript-with-JC

```
// 💡 Private Properties in Class using "#"

class BankAccount {
  // private member
  #amount;
  constructor(name, amount) {
    this.name = name; // this is public
    this.#amount = amount; // this is private
  }

  // private method
  #withdrawAmount(amount) {
    this.#amount -= amount;
    console.log(this.#amount);
  }

  displayAmount() {
    // here you can access private members
    console.log(this.#amount);
  }

  enterPinAndWithdraw(amount) {
     // here you can access private members
    this.#withdrawAmount(amount);
  }
}

const account = new BankAccount("rakesh", 5000);
console.log(account.name); // rakesh

// console.log(account.#amount); // SyntaxError
account.displayAmount(); // 5000

// account.#withdrawAmount(); // SyntaxError
account.enterPinAndWithdraw(2000); // 3000
```

(Jayesh Choudhary)

getters-and-setters.js

#JavaScript-with-JC

```
// 💡 Classes getters and setters

class User {
  #name;
  #age;
  constructor(name, age) {
    this.#name = name;
    this.#age = age;
  }

  get name() {
    return this.#name;
  }

  set name(name) {
    this.#name = name;
  }

  get age() {
    return this.#age;
  }

  set age(age) {
    this.#age = age;
  }
}

const mukesh = new User("mukesh", 44);
console.log(mukesh.name); // mukesh
console.log(mukesh.age); // 44

mukesh.name = "new mukesh";
console.log(mukesh.name); // new mukesh
mukesh.age = 45;
console.log(mukesh.age); // 45
```

(Jayesh Choudhary)

8-Ways-to-create-Object.js

#JavaScript-with-JC

```
// 💡 8 Ways to create an object in javascript

// 👉 1) using new keyword ( object constructor )
const obj1 = new Object();
obj1.name = "Jc";
obj1.age = 24;
console.log(obj1); // { name: 'Jc', age: 24  }


// 👉 2) using object literals
const obj2 = {
  name: "Jc",
  age: 24,
};
console.log(obj2); // { name: 'Jc', age: 24  }


// 👉 3) using function constructor
function Person3(name, age) {
  this.name = name;
  this.age = age;
}
const obj3 = new Person3("Jc", 24);
console.log(obj3); // Person3 { name: 'Jc', age: 24 }


// 👉 4) using class and constructor
class Person4 {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }
}
const obj4 = new Person4("Jc", 24);
console.log(obj4); // Person4 { name: 'Jc', age: 24 }


// 👉 5) using Object.create()
const obj5 = Object.create({ name: "Jc", age: 24 });
console.log(obj5.name); // Jc
console.log(obj5.age); // 24


// 👉 6) using Object.assign()
const obj6 = Object.assign({}, { name: "Jc", age: 24 });
console.log(obj6); // { name: 'Jc', age: 24 }


// 👉 7) using singleton pattern
const obj7 = new (function () {
  this.name = "Jc";
  this.age = 24;
})();
console.log(obj7); // { name: 'Jc', age: 24 }


// 👉 8) using factory function
function Person8() {
  return {
    name: "Jc",
    age: 24,
  };
}
const obj8 = Person8();
console.log(obj8); // { name: 'Jc', age: 24 }
```

(Jayesh Choudhary)