|  |  |
| --- | --- |
| Practical: 5 | *// Define a class representing a vehicle with properties like make, model, and*  *// year. Implement methods to display the vehicle details and calculate the*  *// mileage.Create child classes like Car and Motorcycle that inherit from the Vehicle class*  *// and add specific properties and methods.*  class Vehicle {      constructor(make, model, year) {  *this*.make = make;  *this*.model = model;  *this*.year = year;      }        displayDetails() {        console.log(`Make: ${*this*.make}`);        console.log(`Model: ${*this*.model}`);        console.log(`Year: ${*this*.year}`);      }        calculateMileage() {        console.log('Mileage calculation is not available for this vehicle.');      }    }      class Car extends Vehicle {      constructor(make, model, year, fuelType) {  *super*(make, model, year);  *this*.fuelType = fuelType;      }        displayCarDetails() {  *super*.displayDetails();        console.log(`Fuel Type: ${*this*.fuelType}`);      }        calculateMileage() {        console.log('Calculating car mileage...');      }    }      class Motorcycle extends Vehicle {      constructor(make, model, year, engineType) {  *super*(make, model, year);  *this*.engineType = engineType;      }        displayMotorcycleDetails() {  *super*.displayDetails();        console.log(`Engine Type: ${*this*.engineType}`);      }        calculateMileage() {        console.log('Calculating motorcycle mileage...');      }    }      const myCar = new Car('Hyundai', 'Grant-i10', 2017, 'Petrol');    const myMotorcycle = new Motorcycle('Honda', 'Splender', 2015, 'ninfawfw');      console.log('Car Details:');    myCar.displayCarDetails();    myCar.calculateMileage();      console.log('\nMotorcycle Details:');    myMotorcycle.displayMotorcycleDetails();    myMotorcycle.calculateMileage(); |
| Practical: 6 | *// Use the prototype property to add a new method to an existing object*  *// constructor, such as Array or String.*  Array.*prototype*.customMethod = function () {      for (let i = 0; i < *this*.length; i++) {  *this*[i] \*= 2;      }    };      const myArray = [1, 2, 3, 4, 5];    myArray.customMethod();    console.log(myArray); |
| Practical: 7 | calculator.js  export class Calculator {      add(a, b) {        return a + b;      }      subtract(a, b) {        return a - b;      }        multiply(a, b) {        return a \* b;      }      divide(a, b) {        if (b === 0) {          throw new Error("Division by zero is not allowed.");        }        return a / b;      }    }    Practical\_7.js  *// Create a JavaScript module that exports a class representing a calculator with*  *// methods to perform basic arithmetic operations. Import the module in another*  *// JavaScript file and use the calculator class to perform calculations.*  import { Calculator } from './calculator.js';  const calculator = new Calculator();  const result1 = calculator.add(5, 3);  console.log(`5 + 3 = ${result1}`);  const result2 = calculator.subtract(10, 4);  console.log(`10 - 4 = ${result2}`);  const result3 = calculator.multiply(6, 7);  console.log(`6 \* 7 = ${result3}`);  try {    const result4 = calculator.divide(8, 0);    console.log(`8 / 0 = ${result4}`);  } catch (error) {    console.error(error.message);  } |
| Practical: 8 | fetchdata.js  async function fetchData(username) {      try {        const response = await fetch(`https://api.github.com/users/${username}`);        if (response.*ok*) {          const userData = await response.json();          return userData;        } else {          return null;        }      } catch (error) {        return null;      }    }    export default fetchData;  Practical\_8.js  *// Create a JavaScript module that fetches data from an API using the fetch()*  *// function and exports the retrieved data.*  *// Create an async function getUsers(names), that gets an array of GitHub logins,*  *// fetches the users from GitHub and returns an array of GitHub users.*  *// The GitHub url with user information for the given USERNAME is:*  *// https://api.github.com/users/USERNAME.*  *// There’s a test example in the sandbox.*  *// Important details:*  *// • There should be one fetch request per user.*  *// • Requests shouldn’t wait for each other. So that the data arrives as soon*  *// as possible.*  *// • If any request fails, or if there’s no such user, the function should return*  *// null in the resulting array.*  import fetchData from "./fetchdata.js";  async function main() {    const username = "21ce114";    const userData = await fetchData(username);    if (userData) {      const { login, id, node\_id, url } = userData;      console.log("User Data:");      console.log(`        id : ${id}        login : ${login}        node\_id : ${node\_id}        url : ${url}      `);    } else {      console.log("User not found or request failed.");    }  }  main(); |
| Practical: 9 | moduleA.js  export function greetA() {      console.log("Hello from Module A!");    }    moduleB.js  export function greetB() {      console.log("Hello from Module B!");    }    Practical\_9.js  *// Implement dynamic imports using the import() function to load modules*  *// asynchronously based on certain conditions.*  async function loadModule(condition) {      if (condition) {  *// Dynamically import moduleA.js when the condition is true*        const moduleA = await import('./moduleA.js');        moduleA.greetA();      } else {  *// Dynamically import moduleB.js when the condition is false*        const moduleB = await import('./moduleB.js');        moduleB.greetB();      }    }    const condition = false;    *// Load the appropriate module based on the condition*    loadModule(condition); |
| Practical: 10 | *// Create an iterator that generates an infinite sequence of numbers and a generator*  *// that yields a sequence of even numbers. Use the iterator and generator in*  *// different scenarios.*  *// Infinite sequence iterator*  class InfiniteSequence {      [Symbol.*iterator*]() {        let num = 0;        return {          next: () => ({ value: num++, done: false })        };      }    }    const iterator = new InfiniteSequence()[Symbol.*iterator*]();    for (let i = 0; i < 5; i++) {      console.log(iterator.next().value);    }    *// Generator for even numbers*  function\* evenNumberGenerator() {      let num = 0;      while (true) {        yield num;        num += 2;      }    }      const evenGen = evenNumberGenerator();    for (let i = 0; i < 5; i++) {      console.log(evenGen.next().value);    } |