| // 1. Type Conversion.  // Write a function called convertToNumber that takes a string as an argument and returns the equivalent number. If the string cannot be converted to a number, the function should return the string "Invalid number". Use error handling in javascript to achieve this output.  function convertToNumber(str) {  try {  const num = Number(str);  if (Number.isNaN(num)) {  throw new Error("Invalid number");  }  return num;  } catch (error) {  return error.message;  } }  console.log(convertToNumber("123")); console.log(convertToNumber("abc")); |
| --- |

| // 2. Building Robust Functions in JavaScript  // Create a function called getPerson that takes an object as a parameter representing a person's name and age. The function should return the person's name and age as a string in the format "Name: <name>, Age: <age>". However, if the parameter is not a valid object with the properties "name" and "age", the function should throw an error with the message "Invalid parameter type". Use try-catch to handle this error and return the error message if it occurs.  function getPerson(person) {  try {  if (typeof person !== "object" || !person.name || !person.age) {  throw new Error("Invalid parameter type");  }  return `Name: ${person.name}, Age: ${person.age}`;  } catch (error) {  return error.message;  } }  // Expected Output console.log(getPerson({ name: "Mithun", age: 20 })); // Name: Mithun, Age: 20 console.log(getPerson({ name: "Mithun" })); // Invalid parameter type console.log(getPerson(["name", "Mithun"])); // Invalid parameter type |
| --- |

| // 3. Car Description Class.  // Create a class called Car with three properties: company, model, and year. The class should have a method called getDescription that returns a string in the format "This is a <year> <company> <model>". Instantiate an instance of the Car class and call the getDescription method.  // Expected Output  // console.log(myCar.getDescription()); // Output: This is a 2022 Skoda Rapid.  class Car {  constructor(company, model, year) {  this.company = company;  this.model = model;  this.year = year;  }   getDescription() {  return `This is a ${this.year} ${this.company} ${this.model}.`;  } }  const myCar = new Car("Skoda", "Rapid", 2022);  console.log(myCar.getDescription()); // Output: This is a 2022 Skoda Rapid. |
| --- |

| /\* 4. Employee Class Challenge.  Create a class called Employee with three properties: name, position, and salary. The class should have a method called getSalary that returns the employee's salary. Instantiate an instance of the Employee class and call the getSalary method.  Expected Output: console.log(employee1.getSalary()); // Output: 80000 \*/  class Employee {  constructor(name, position, salary) {  this.name = name;  this.position = position;  this.salary = salary;  }   getSalary() {  return this.salary;  } }  const employee1 = new Employee("Prabir Kumar", "Software Engineer", 80000);  // Expected Output console.log(employee1.getSalary()); // Output: 80000 |
| --- |

| /\*  5. Implementing a Person Class with Default Values  Create a class called Person with two properties: name and age. The class should have a method called getDetails that returns a string in the format "Name: <name>, Age: <age>". Use default parameters in the constructor to set the values of name and age to "Unknown" and 0 if they are not provided.  Expected Output  const person1 = new Person("Mithun", 20); console.log(person1.getDetails()); // Output: "Name: Mithun, Age: 20"  const person2 = new Person(); console.log(person2.getDetails()); // Output: "Name: Unknown, Age: 0"  \*/  class Person {  constructor(name = "Unknown", age = 0) {  this.name = name;  this.age = age;  }   getDetails() {  return `Name: ${this.name}, Age: ${this.age}`;  } }  // Expected Output  const person1 = new Person("Mithun", 20); console.log(person1.getDetails()); // Output: "Name: Mithun, Age: 20"  const person2 = new Person(); console.log(person2.getDetails()); // Output: "Name: Unknown, Age: 0" |
| --- |

| /\*  6. Using Static Method to Add Two Numbers with Calculator Class  Create a class called Calculator with a static method called add. The add method should take two numbers as arguments and return their sum. Instantiate the Calculator class and call the add method.  Expected Output:  console.log(result); // Output: 15  \*/  class Calculator {  static add(num1, num2) {  return num1 + num2;  } }  const result = Calculator.add(5, 10);  // Expected Output: console.log(result); // Output: 15 |
| --- |

| /\*  7. Password Checker.  Create a class called User with properties username and password. Implement a getter method for password that returns the password with all characters replaced by asterisks. Implement a setter method for password that checks if the new password is at least 8 characters long and contains at least one number and one uppercase letter. If the password is valid, set the new password. If not, log an error message.  Expected output: const user = new User("johndoe", "Password123"); console.log(user.getPassword()); // \*\*\*\*\*\*\*\*\*\*\*  user.setPassword("myPassword"); // Error: Password must be at least 8 characters long and contain at least one number and one uppercase letter.  user.setPassword("MyPassword"); // Error: Password must be at least 8 characters long and contain at least one number and one uppercase letter.  user.setPassword("Mypassword123"); console.log(user.getPassword()); // \*\*\*\*\*\*\*\*\*\*\*\*\*    \*/  class User {  constructor(username, password) {  this.username = username;  this.password = password;  }   getPassword() {  return this.password.replace(/./g, "\*");  }   setPassword(newPassword) {  let containsNumber = false;  let containsUppercase = false;  for (let i = 0; i < newPassword.length; i++) {  const char = newPassword.charAt(i);  if (!isNaN(char)) {  containsNumber = true;  } else if (char === char.toUpperCase()) {  containsUppercase = true;  }  }  if (newPassword.length >= 8 && containsNumber && containsUppercase) {  this.password = newPassword;  } else {  console.log(  "Error: Password must be at least 8 characters long and contain at least one number and one uppercase letter."  );  }  } }  // Expected output: const user = new User("johndoe", "Password123"); console.log(user.getPassword()); // \*\*\*\*\*\*\*\*\*\*\*  user.setPassword("myPassword"); // Error: Password must be at least 8 characters long and contain at least one number and one uppercase letter. user.setPassword("MyPassword"); // Error: Password must be at least 8 characters long and contain at least one number and one uppercase letter. user.setPassword("Mypassword123"); console.log(user.getPassword()); // \*\*\*\*\*\*\*\*\*\*\*\*\* |
| --- |

| // 8. Adding a Method to a Prototype.  // Create a prototype object called Student with a property name. Add a method called printDetails to the prototype that logs the string "Hello, my name is {name}" to the console. Instantiate a Student object with the name "Mithun" and call the printDetails method.  // Expected Output  // const student = new Student("Mithun"); // student.printDetails(); // "Hello, the student is Mithun"  function Student(name) {  this.name = name; }  Student.prototype.printDetails = function () {  console.log(`Hello, my name is ${this.name}`); };  // Expected Output  const student = new Student("Mithun"); student.printDetails(); // "Hello, the student is Mithun" |
| --- |

| /\*  9. Check the presence using closures.  Create a numberChecker function that takes an array of numbers as an argument and returns a function. The returned function should take another number as an argument and return true if the number is in the array, and false otherwise.   \*/  function numberChecker(numbers) {  return function (num) {  return numbers.includes(num);  }; }  // Expected Result: const arr = [1, 2, 3, 4, 5]; const checkNum = numberChecker(arr);  console.log(checkNum(3)); // true console.log(checkNum(6)); // false |
| --- |

| /\*  10. Filter by Category.  Write a function that takes an array of products and returns a function that filters the array by a given product category. The function must filter an eCommerce products array by a specific category. The closure filters products using the filter() method. Finally, it returns a new array containing only the products with the same category as the input.   \*/  function filterByCategory(products) {  return function (category) {  return products.filter(function (product) {  return product.category === category;  });  }; }  // Expected Output  var products = [  { name: "Shirt", category: "Clothing" },  { name: "Pants", category: "Clothing" },  { name: "Hat", category: "Accessories" },  { name: "Sunglasses", category: "Accessories" }, ];  var clothingProducts = filterByCategory(products)("Clothing");  console.log(clothingProducts); // Output: [{name: "Shirt", category: "Clothing"}, {name: "Pants", category: "Clothing"}] |
| --- |