UID TUTORIAL – I

SET - A

2 Marks

1. *ES6 Classes vs. Constructor Functions:*

ES6 classes are a syntactical abstraction over constructor functions and prototypes in JavaScript, providing a more organized and intuitive way to create objects and define their behavior. They offer a more structured and familiar syntax for defining classes and their methods, promoting better code organization and readability.

2. *Creating an ES6 Class and its Components:*

To create an ES6 class, use the `class` keyword, followed by the class name. Components include:

- Constructor method: Initialize properties when an object is created.
- Methods: Define class-specific functions.
- Getter and setter methods: Access and modify class properties.
- Static methods: Functions that belong to the class itself, not instances.

3. *Inheritance with the `extends` Keyword:*

In ES6 classes, inheritance is achieved using the `extends` keyword. It allows a subclass (child) to inherit properties and methods from a superclass (parent). Subclasses can also have their own properties and methods.

4. *Creating an Instance of a Class:*

Use the `new` keyword followed by the class name to create an instance of a class:

```
javascript
class MyClass {
    // class definition
}
const myInstance = new MyClass();
```

5. *Benefits of ES6 Classes:*

- More readable and structured syntax.
- Easier inheritance and subclassing.
- Improved encapsulation and privacy with `constructor`, `get`, and `set` keywords.

6. *Defining Getter and Setter Methods:*

Use the `get` and `set` keywords within a class to define getter and setter methods:

```
javascript
class MyClass {
    get myProperty() {
        return this._myProperty;
    }
    set myProperty(value) {
        this._myProperty = value;
    }
}
```

7. *Role of the Constructor Method:*

The constructor method initializes class properties when an object is created. It's automatically called when using the `new` keyword. It sets up the initial state of the object.

SET – B

2 Marks

8. *Creating Object with Computed Property Names:*

Use square brackets `[]` to create an object with computed property names:

```
javascript
const dynamicKey = "propertyName";
const obj = {
    [dynamicKey]: "propertyValue"
};
```

9. *Difference between Object.keys(), Object.values(), and Object.entries():*

- `Object.keys()` returns an array of object keys.
- `Object.values()` returns an array of object values.
- `Object.entries()` returns an array of [key, value] pairs.

10. *ES6 Classes and Objects with Asynchronous Programming:*

```
javascript
```

```
class AsyncExample {
    async fetchData() {
        const response = await fetch('https://api.example.com/data');
        const data = await response.json();
        return data;
    }
}
const asyncInstance = new AsyncExample();
asyncInstance.fetchData().then(data => {
    console.log(data);
});
```

11. *Arrow Functions vs. Regular Functions:*

Arrow functions are a concise way to write functions in ES6. They have a shorter syntax, do not have their own `this` context, and cannot be used as constructors.

12. *Declaring a Basic Arrow Function:*

```
javascript
const add = (a, b) => a + b
```

13. *Lexical Scoping in Arrow Functions:*

Arrow functions capture the surrounding `this` value lexically, i.e., based on where they are defined. They do not have their own `this`, making them suitable for maintaining the parent's `this` context.

14. *Arrow Function Multiple Lines and this:*

For multiple lines, wrap the function body in curly braces:

```
javascript
const multiply = (a, b) => {
    const result = a * b;
    return result;
};
```

SET - C

2 Marks

15. *Advantages of Arrow Functions:*

- Concise syntax.

- Avoids issues with `this` scoping.

- Often used in callback functions to maintain context.

16. *Using the `arguments` Object:*

Arrow functions do not have their own `arguments` object. They inherit `arguments` from their containing function, which can lead to unexpected behavior.

17. *Arrow Function Returning an Object:*

javascript

```
const createPerson = (name, age) => ({ name, age });
```

18. *Arrow Function with Multiple Lines of Code:*

javascript

```
const complexFunction = (a, b) => {
    const resultA = a * 2;
    const resultB = b * 3;
    return resultA + resultB;
};
```

19. *Arrow Functions vs. Regular Functions:*

Arrow functions have different behavior with `this`, cannot be constructors, and offer a shorter syntax compared to regular functions.

20. *Arrow Functions in Event Handlers:*

Arrow functions are often used in event handlers to maintain the lexical `this` context, avoiding the need for `.bind(this)`.

21. *Difference between var, let, and const:*

- `var` has function scope and can be hoisted.

- `let` and `const` have block scope and do not hoist.
- `const` is used for variables that should not be reassigned after initialization.

Example:

javascript

```
var x = 10;       // Hoisting
let y = 20;       // Block-scoped
const z = 30;     // Block-scoped constant
```

SET-A

10 Marks

21.i)Create a class called Person with properties name and age. Implement a method called introduce() that returns a string introducing the person with their name and age. Then, create an object of the class and call the introduce() method to display the introduction.

```
class Person {
   constructor(name, age) {
      this.name = name;
      this.age = age;
   }

   introduce() {
      return `Hi, I'm ${this.name} and I'm ${this.age} years old.`;
   }
}
// Creating an object of the Person class
const person = new Person("John", 25);
const intro = person.introduce();
console.log(intro);
```

ii)Given an array of numbers, use arrow functions with array methods to solve the following tasks:

a. Find the sum of all numbers in the array. b. Filter out even numbers from the array.

```
// a. Find the sum of all numbers in the array
const numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
const sum = numbers.reduce((acc, num) => acc + num, 0);
```

```
console.log("Sum of numbers:", sum);


// b. Filter out even numbers from the array
const evenNumbers = numbers.filter(num => num % 2 === 0);
console.log("Even numbers:", evenNumbers);
```

SET-B

10 Marks

22. i)Create a class called Car with properties brand, model, and year. Display all the info in a web page.

```
<!DOCTYPE html>
<html>
<head>
  <title>Car Information</title>
</head>
<body>
  <div id="carInfo"></div>

  <script>
    class Car {
      constructor(brand, model, year) {
        this.brand = brand;
        this.model = model;
        this.year = year;
      }
    }

    // Create an instance of the Car class
    const myCar = new Car("Toyota", "Camry", 2022);

    // Display car information on the web page
    const carInfoDiv = document.getElementById("carInfo");
    carInfoDiv.innerHTML = `
      <h2>Car Information</h2>
```

```
        <p><strong>Brand:</strong> ${myCar.brand}</p>
        <p><strong>Model:</strong> ${myCar.model}</p>
        <p><strong>Year:</strong> ${myCar.year}</p>
      `;
  </script>
</body>
</html>
```

ii)Given an array of numbers, use arrow functions with array methods to solve the following tasks:a. Square each number in the array and store the results in a new array.b. Filter out numbers which are divisible by from the array.

```
<!DOCTYPE html>
<html>
<head>
  <title>Number Operations</title>
</head>
<body>
  <div id="output"></div>

  <script>
    const numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

    // a. Square each number in the array and store the results in a new array
    const squaredNumbers = numbers.map(num => num * num);

    // b. Filter out numbers which are divisible by 3 from the array
    const nonDivisibleByThree = numbers.filter(num => num % 3 !== 0);

    const outputDiv = document.getElementById("output");
    outputDiv.innerHTML = `
      <h2>Squared Numbers</h2>
      <p>${squaredNumbers.join(", ")}</p>
```

```
        <h2>Non-Divisible by Three</h2>
        <p>${nonDivisibleByThree.join(", ")}</p>
      `;
  </script>
</body>
</html>
```

SET-C

10 Marks

23. i)Create a class called Event with properties name, venue, and date. Display all the info in a web page.

```
<!DOCTYPE html>
<html>
<head>
  <title>Event Information</title>
</head>
<body>
  <div id="eventInfo"></div>

  <script>
    class Event {
      constructor(name, venue, date) {
        this.name = name;
        this.venue = venue;
        this.date = date;
      }
    }

    // Create an instance of the Event class
    const myEvent = new Event("Music Festival", "Central Park", "2023-09-15");

    // Display event information on the web page
    const eventInfoDiv = document.getElementById("eventInfo");
```

```
        eventInfoDiv.innerHTML = `
            <h2>Event Information</h2>
            <p><strong>Name:</strong> ${myEvent.name}</p>
            <p><strong>Venue:</strong> ${myEvent.venue}</p>
            <p><strong>Date:</strong> ${myEvent.date}</p>
        `;
    </script>
</body>
</html>
```

ii)Create an array of numbers and use the some() method with an arrow function to check if there is any number greater than 10 in the array.

```
<!DOCTYPE html>
<html>
<head>
    <title>Check Numbers</title>
</head>
<body>
    <div id="output"></div>

    <script>
        const numbers = [5, 8, 12, 4, 7, 3, 15, 9];

        // Check if there is any number greater than 10 in the array
        const hasNumberGreaterThan10 = numbers.some(num => num > 10);

        const outputDiv = document.getElementById("output");
        outputDiv.innerHTML = `
            <h2>Result</h2>
            <p>Does the array contain a number greater than 10? ${hasNumberGreaterThan10 ? 'Yes' : 'No'}</p>
        `;
    </script>
</body>
```

```
</html>
```