FULL STACK DEVELOPMENT

Unit-2

Imp Q & A

1) Explain OOPS concepts of JavaScript engine.

Ans) Object-Oriented Programming (OOP) in JavaScript allows developers to create reusable and modular code by modeling real-world entities as objects. Let's delve into the core OOP concepts in the JavaScript engine:

1. Objects

Objects are collections of properties, and a property is an association between a name (or key) and a value. The value can be a function, which makes the property a method.

Example:

javascript

```
const person = {
   name: "John",
   age: 30,
   greet: function() {
      console.log("Hello, my name is " + this.name);
   }
};


person.greet();  // Output: Hello, my name is John
```

2. Classes

Classes are templates for creating objects. They encapsulate data with code to work on that data. JavaScript introduced the class syntax in ECMAScript 6 (ES6).

Example:

javascript

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

```
  }

  greet() {
    console.log(`Hello, my name is ${this.name}`);
  }
}

const john = new Person('John', 30);
john.greet();  // Output: Hello, my name is John
```

3. Inheritance

Inheritance is a mechanism where one class (subclass or child class) inherits properties and methods from another class (superclass or parent class). This promotes code reuse.

Example:

javascript

```
class Employee extends Person {
  constructor(name, age, jobTitle) {
    super(name, age);  // Call the parent class constructor
    this.jobTitle = jobTitle;
  }

  describe() {
    console.log(`I am ${this.name}, a ${this.jobTitle}`);
  }
}

const alice = new Employee('Alice', 25, 'Developer');
alice.greet();     // Output: Hello, my name is Alice
alice.describe();  // Output: I am Alice, a Developer
```

4. Encapsulation

Encapsulation is the practice of keeping fields (data) within a class private and providing controlled access via public methods. This protects the data from unintended modifications.

Example:

javascript

```
class BankAccount {
  constructor(balance) {
    this._balance = balance;
  }

  deposit(amount) {
    this._balance += amount;
  }

  withdraw(amount) {
    if (amount <= this._balance) {
      this._balance -= amount;
    } else {
      console.log('Insufficient funds');
    }
  }

  getBalance() {
    return this._balance;
  }
}

const account = new BankAccount(1000);
account.deposit(500);
account.withdraw(300);
console.log(account.getBalance());  // Output: 1200
```

5. Polymorphism

Polymorphism allows methods to do different things based on the object it is acting upon. This can be achieved through method overriding and method overloading (though method overloading is not directly supported in JavaScript).

Method Overriding Example:

javascript

```
class Animal {
   makeSound() {
      console.log('Some generic sound');
   }
}

class Dog extends Animal {
   makeSound() {
      console.log('Bark');
   }
}

const generic = new Animal();
const dog = new Dog();

generic.makeSound();  // Output: Some generic sound
dog.makeSound();      // Output: Bark
```

Summary

- **Objects**: Collections of properties and methods.
- **Classes**: Blueprints for creating objects.
- **Inheritance**: Classes can inherit properties and methods from other classes.
- **Encapsulation**: Restrict direct access to some of an object's components.
- **Polymorphism**: Methods can behave differently based on the object instance.

2) Java script functions and functions with parameters?

Ans) Function parameters are variables defined in the function declaration that receive values (arguments) when the function is called. They play a key role in making functions reusable and dynamic.

- Values are assigned to parameters in the order they are passed.
- You can assign default values to parameters if no arguments are provided.
- Allows capturing an indefinite number of arguments into an array.
- Primitive types are passed by value, whereas objects are passed by reference.

```
function greet(name) {
  return `Hello, ${name}!`;
}
console.log(greet("Meeta"));
```

Output

Hello, Meeta!

- **Parameter:** name in the function definition.
- **Argument:** “Meeta” passed when calling the function.

Types of Parameters in JavaScript

1. Required Parameters

These are the basic parameters expected by the function. If not provided, they will be undefined.

```
function add(a, b) {
  return a + b;
}
console.log(add(5, 3));
console.log(add(5));
```

Output

8

NaN

2. Default Parameters

Introduced in ES6, default parameters allow you to assign a default value to a parameter if no argument is passed or if the argument is undefined.

```
function mul(a, b = 1) {
   return a * b;
}
console.log(mul(5));
console.log(mul(5, 2));
```

Output

5

10

3. Rest Parameters

Rest parameters allow a function to accept an indefinite number of arguments as an array. Use the … syntax to capture all additional arguments.

```
function sum(...numbers) {
   return numbers.reduce((total, num) => total + num, 0);
}
console.log(sum(1, 2, 3, 4));
```

Output

10

4. Destructured Parameters

You can destructure arrays or objects passed as arguments into individual variables.

```
function displayUser({ name, age }) {
   return `${name} is ${age} years old.`;
}
const user = { name: "Meeta", age: 25 };
console.log(displayUser(user));
```

Output

Meeta is 25 years old.

5. Passing Functions as Parameters (Higher-Order Functions)

Functions in JavaScript can accept other functions as parameters, making it easy to create reusable code.

```
function executeTask(task, callback) {
```

```
    console.log(`Task: ${task}`);
    callback();
}
executeTask("Clean the room", () => {
    console.log("Task Completed!");
});
```

Output

Task: Clean the room

Task Completed!

Functions: Encapsulate reusable code blocks.

Parameters: Placeholders for input values.

Default Parameters: Set default values for parameters.

Anonymous Functions: Functions without names.

Arrow Functions: Concise syntax for writing functions.

Higher-Order Functions: Functions that take or return other functions.

3) What are Java script arrow functions?

Ans) An arrow function is a shorter syntax for writing functions in JavaScript. Introduced in ES6, arrow functions allow for a more concise and readable code, especially in cases of small functions. Unlike regular functions, arrow functions don't have their own this, but instead, inherit it from the surrounding context.

- Arrow functions are written with the => symbol, which makes them compact.
- They don't have their own this. They inherit this from the surrounding context.
- For functions with a single expression, the return is implicit, making the code more concise.
- Arrow functions do not have access to the arguments object, which is available in regular functions.

```
const add = (a, b) => a + b;
console.log(add(5, 3));
```

Output

8

- **‘add’** is an arrow function that takes two parameters a and b, and returns their sum.
- The arrow function’s concise syntax eliminates the need for the function keyword and curly braces for single-line expressions.

1. Arrow Function without Parameters

An arrow function without parameters is defined using empty parentheses (). This is useful when you need a function that doesn’t require any arguments.

```
const gfg = () => {
   console.log( "Hi from GeekforGeeks!" );
}


gfg();
```

Output

Hi from GeekforGeeks!

2. Arrow Function with Single Parameters

If your arrow function has a single parameter, you can omit the parentheses around it.

```
const square = x => x*x;

console.log(square(4));
```

Output

16

3. Arrow Function with Multiple Parameters

Arrow functions with multiple parameters, like **(param1, param2) => { }**, simplify writing concise function expressions in JavaScript, useful for functions requiring more than one argument.

```
const gfg = ( x, y, z ) => {
   console.log( x + y + z )
}


gfg( 10, 20, 30 );
```

Output

60

4. Arrow Function with Default Parameters

Arrow functions support default parameters, allowing predefined values if no argument is passed, making JavaScript function definitions more flexible and concise.

```
const gfg = ( x, y, z = 30 ) => {
   console.log( x + " " + y + " " + z);
}


gfg( 10, 20 );
```

Output

10 20 30

5. Return Object Literals

In JavaScript, returning object literals within functions is concise: **() => ({ key: value })** returns an object { key: value }, useful for immediate object creation and returning.

```
const makePerson = (firstName, lastName) =>
({first: firstName, last: lastName});
console.log(makePerson("Pankaj", "Bind"));
```

Output

{ first: 'Pankaj', last: 'Bind' }

Advantages of Arrow Functions

- **Concise Syntax:** Arrow functions reduce the amount of code needed for function expressions.
- **Lexical *this* Binding:** Arrow functions automatically bind this to the surrounding context, eliminating common issues when dealing with callbacks.
- **Improved Readability:** For shorter functions, arrow syntax can make your code more readable.

Limitations of Arrow Functions

- **No prototype Property:** Arrow functions do not have the prototype property, so they cannot be used as constructors.
- **Cannot be Used with *new*:** Since they lack a prototype, they cannot be used with the new keyword to create instances.

- **Cannot be Generators:** Arrow functions cannot be used as generator functions (function*) because they do not support the yield keyword.
- **Anonymous Nature:** Debugging can be harder because arrow functions are anonymous by default.
- **No Own this, arguments, super, or new.target:** Arrow functions do not have their own bindings for these properties, which can limit their use in some cases.

4) How browser memory we can use in JavaScript write one program?

Ans) Memory management in JavaScript is handled automatically by the runtime environment, typically the JavaScript engine in web browsers or Node.js. JavaScript uses a garbage collector to manage memory and ensure that developers do not need to manually allocate or deallocate memory.

Memory Life Cycle

Irrespective of the programming language, the memory life cycle follows the following stages:

1. **Allocates the memory we need:** JavaScript allocates memory to the object created.
2. **Use the allocated memory.**
3. **Release the memory when not in use:** Once the allocated memory is released, it is used for other purposes. It is handled by a JavaScript engine.

The second stage is the same for all the languages. However, the first and last stages are implicit in high-level languages like JavaScript.

Note: “Objects” in this context not only mean objects in JavaScript but also functions and function scopes.

JavaScript engines have two places to store data

- **Stack:** It is a data structure used to store static data. Static data refers to data whose size is known by the engine during compile time. In JavaScript, static data includes primitive values like strings, numbers, boolean, null, and undefined. References that point to objects and functions are also included. A fixed amount of memory is allocated for static data. This process is known as **static memory allocation.**
- **Heap:** It is used to store objects and functions in JavaScript. The engine doesn’t allocate a fixed amount of memory. Instead, it allocates more space as required.

Overview:

Stack	Heap
Primitive data types and references	Objects and functions
Size is known at compile time	Size is known at run time
Fixed memory allocated	No limit for object memory

Example: In the below example object 'employee' is created in the heap and a reference to it is in the stack.

- Javascript

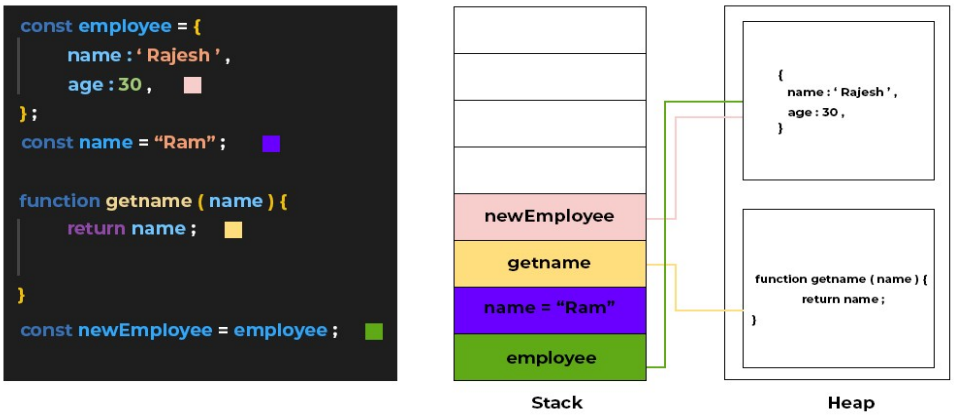
```
const employee = {
  name: 'Rajesh',
  age: 30,
};

const name="Ram"
// Allocates memory for object in heap.Values
// in object are primitive,which is why they
// are stored in stack.

function getname(name) {
   return name;
}
// The function return value is given to stack after
// being evaluated in the heap

const newEmployee = employee;
// The newEmployee object will be stored in the stack and
```

```
// it will refer to the employee object in heap
```



Memory Management in JavaScript

Garbage Collection

Garbage collectors are used in releasing memory. Once the engine recognizes that a variable, object, or function is not needed anymore, it releases the memory it occupied. The main issue here is that it is very difficult to predict accurately whether a particular variable, object, or function is needed anymore or not. Some algorithms help to find the moment when they become obsolete with great precision.

Reference-counting garbage collection

It frees the memory allocated to the objects that have no references pointing to them. However, the problem with this algorithm is that it doesn't understand cyclic references.

Example: In this example, the game and boy both reference each other. Thus, the algorithm won't release the allocated memory. Setting them to *null* won't make the algorithm realize that they can't be used anymore, leading to no release of allocated memory.

- Javascript

```
let game = {
  name: 'cricket',
};

let boy = {
  name: 'Ram',
```

```
}


game.boy = boy;
boy.game = game;


boy = null;
game = null;
```

5) How AJAX PROGRAM WORKS while exchanging the data in the background write a script for AJAX SAMPLE CODE

Ans) AJAX (Asynchronous JavaScript and XML) allows web pages to update asynchronously by exchanging small amounts of data with the server in the background. This avoids reloading the entire page, making web applications more interactive and faster.

How AJAX Works

1. **User Action**: A user triggers an event (e.g., clicking a button).
2. **JavaScript Creates an XMLHttpRequest Object**: This object sends a request to the server.
3. **Server Processes Request**: The server processes the request and sends back a response.
4. **JavaScript Receives Response**: The response is processed, and the webpage updates dynamically without a full reload.

AJAX Sample Code (Using JavaScript and PHP)

Here's a simple AJAX example where a button click fetches data from a PHP file and displays it without refreshing the page.

1. HTML & JavaScript (Frontend)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
    <title>AJAX Example</title>
    <script>
        function loadData() {
            // Create an XMLHttpRequest object
            var xhr = new XMLHttpRequest();

            // Define the request method, URL, and set it to asynchronous
            xhr.open("GET", "data.php", true);

            // Handle response
            xhr.onreadystatechange = function () {
                if (xhr.readyState == 4 && xhr.status == 200) {
                    // Update the content of the div with the response text
                    document.getElementById("result").innerHTML = xhr.responseText;
                }
            };

            // Send the request to the server
            xhr.send();
        }
    </script>
</head>
<body>

    <h2>AJAX Example</h2>
    <button onclick="loadData()">Fetch Data</button>
    <div id="result">Data will be displayed here...</div>

</body>
```

```
</html>
```

2. PHP Script (Backend - data.php)

```
<?php
// Simulating fetching data from the database or an API
echo "Hello, this is data fetched using AJAX!";
?>
```

How It Works

1. The user clicks the "Fetch Data" button.
2. The loadData() JavaScript function is called.
3. An AJAX request (XMLHttpRequest) is sent to data.php.
4. When the request is successful, the response is displayed inside the div without reloading the page.

6) JQUERY events hide Show Toggle fadeOut fadeIn fadeToggle slideUp slideDown slideToggle Write Program on above.

Ans)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>jQuery Events Example</title>
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
  <style>
    .box {
      width: 100px;
      height: 100px;
      background-color: #007BFF;
      margin: 10px;
```

```
        }
    </style>
</head>
<body>
    <h1>jQuery Events Example</h1>
    <div class="buttons">
        <button id="btn-hide">Hide</button>
        <button id="btn-show">Show</button>
        <button id="btn-toggle">Toggle</button>
        <button id="btn-fadeout">Fade Out</button>
        <button id="btn-fadein">Fade In</button>
        <button id="btn-fadetoggle">Fade Toggle</button>
        <button id="btn-slideup">Slide Up</button>
        <button id="btn-slidedown">Slide Down</button>
        <button id="btn-slidetoggle">Slide Toggle</button>
    </div>
    <div class="box"></div>

    <script>
        $(document).ready(function() {
            $("#btn-hide").click(function() {
                $(".box").hide();
            });

            $("#btn-show").click(function() {
                $(".box").show();
            });

            $("#btn-toggle").click(function() {
```

```
            $(".box").toggle();
        });

        $("#btn-fadeout").click(function() {
            $(".box").fadeOut();
        });

        $("#btn-fadein").click(function() {
            $(".box").fadeIn();
        });

        $("#btn-fadetoggle").click(function() {
            $(".box").fadeToggle();
        });

        $("#btn-slideup").click(function() {
            $(".box").slideUp();
        });

        $("#btn-slidedown").click(function() {
            $(".box").slideDown();
        });

        $("#btn-slidetoggle").click(function() {
            $(".box").slideToggle();
        });
    });
  </script>
</body>
```

</html>

Explanation of jQuery Events Used

1. **hide()** → Hides the #box element.
2. **show()** → Shows the #box element.
3. **toggle()** → Toggles between show and hide.
4. **fadeOut()** → Fades the element out.
5. **fadeIn()** → Fades the element in.
6. **fadeToggle()** → Toggles between fade in and fade out.
7. **slideUp()** → Slides the element up (hides it).
8. **slideDown()** → Slides the element down (shows it).
9. **slideToggle()** → Toggles between slide up and slide down.

7) jQuery UI components Date picker Accordion Auto complete Tabs Draggable write a small program using these

Ans) <!DOCTYPE html>

```
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>jQuery UI Components Example</title>
  <link rel="stylesheet" href="https://code.jquery.com/ui/1.12.1/themes/base/jquery-ui.css">
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
  <script src="https://code.jquery.com/ui/1.12.1/jquery-ui.min.js"></script>
  <style>
    .draggable { width: 100px; height: 100px; background-color: #FF5733; text-align: center; line-height: 100px; color: white; margin: 20px; cursor: move; }
  </style>
</head>
<body>
```

```
<h1>jQuery UI Components Example</h1>

<!-- Date Picker -->
<p>Date: <input type="text" id="datepicker"></p>

<!-- Accordion -->
<div id="accordion">
  <h3>Section 1</h3>
  <div><p>Content for section 1.</p></div>
  <h3>Section 2</h3>
  <div><p>Content for section 2.</p></div>
  <h3>Section 3</h3>
  <div><p>Content for section 3.</p></div>
</div>

<!-- Autocomplete -->
<label for="autocomplete">Tags: </label>
<input id="autocomplete">

<!-- Tabs -->
<div id="tabs">
  <ul>
    <li><a href="#tabs-1">Tab 1</a></li>
    <li><a href="#tabs-2">Tab 2</a></li>
    <li><a href="#tabs-3">Tab 3</a></li>
  </ul>
  <div id="tabs-1"><p>Content for Tab 1.</p></div>
  <div id="tabs-2"><p>Content for Tab 2.</p></div>
  <div id="tabs-3"><p>Content for Tab 3.</p></div>
```

```
</div>

<!-- Draggable -->
<div class="draggable">Drag me!</div>

<script>
    $(function() {
        // Date Picker
        $("#datepicker").datepicker();

        // Accordion
        $("#accordion").accordion();

        // Autocomplete
        var availableTags = ["ActionScript", "AppleScript", "Asp", "BASIC", "C", "C++",
"Clojure", "COBOL", "ColdFusion",
                        "Erlang", "Fortran", "Groovy", "Haskell", "Java", "JavaScript", "Lisp",
"Perl", "PHP", "Python",
                        "Ruby", "Scala", "Scheme"];
        $("#autocomplete").autocomplete({
            source: availableTags
        });

        // Tabs
        $("#tabs").tabs();

        // Draggable
        $(".draggable").draggable();
    });
</script>
```

</body>

</html>

Features Included

1. **Datepicker** → Click on the input field to select a date.
2. **Autocomplete** → Start typing a country name, and suggestions appear.
3. **Tabs** → Click on tabs to switch content.
4. **Accordion** → Expand/collapse sections.
5. **Draggable** → Move the #draggable-box anywhere on the screen.