Name - Ritesh maurya

VASANTDADA PATIL PRATISHTHAN'S
COLLEGE OF ENGINEERING & VISUAL ARTS

Page No.

Assignment No. 2

(1) Explain arrow function with example.

Ans An Arrow function in Javascript introduce in ES6 offers a concise Syntax for defining function Expression using arrow (=>)

Arrow function are anonymous function that is function without a name but they are often asigned to any variable. They are also called Lambda function.

ex
```
const sum = (x,y,z) => {
    console.log(x+y+z)
}
sum(10,20,30).
```

Advantages →

→ It reduce the size of the code
→ Return statement & functions brackets are optional for single line function.

→ It increases the read-ability of code.

(2) Explain promises in ES6 with an example.

Ans In ES6 promises provide a way to handle asynchronous operations more easily, especially when dealing with tasks like fetching data from API, reading file or interacting with a database promises help to avoid "callback hell" and make asynchronous code more readable and manageable

A promise is an object representing the eventual completion (or failure) of an asynchronous operation it can be in one of three steps.

→ pending: The initial state, meaning the operation is still ongoing.

→ Folfilled: The operation completed successfully, and the promise has a value (resolved).

→ Rejected: The operation failed, and the promise has a reason for the failure (error).

Syntax: let promise = new promise(function(resolve, reject){

```
if(success)
    resolve(result);
} else{
    reject(error);
});
```

→ resolve.(value): Called when the asynchronous operation completes successfully

→ reject(reason): Called when the asynchronous operation fails.

Handling promises: Two main methods for handling

→ then(): Used to handle a fulfilled promise and retrieve the result.

→ catch(): Used to handle errors(reject promises.

(3) Write a short note on
a) NPM
b) State and props
c) Component life cycle

a) NPM (Node package manager) → NPM is the default package manager for Node.js it allows developers to manage and share JavaScript libraries and packages efficiently. With NPM, you can

→ Install packages locally (to a project) or globally.
→ Publish your own packages of other to use.
→ Handle dependencies by specifying them in a package.json file.

Commands → npm install a package.
→ npm init Create a new package.json file.
→ npm run: Execute custom scripts defined in package.json.

(b) State & props:→ In React, state & props are essential concepts for managing data in components.

State:- Represents the local state of a component that can changes over time.

→ It is mutable, and changes to the state cause the component to re-render.

→ managed internally by the component using Usestate or this.setstate (in class components).

ex→ Const [count, setCount] = Usestate(0);

Props :→ short for "properties", props are used to pass data from a plant component to a child component.

→ They are immutable, meaning the child component cannot modify them.

→ props allow components to be dynamic & reusable.

ex→ function Greeting({name}) {
return <H1> Hello, {name}! </H1>
}

(c) component life cycle → In React, the component life cycle refers to the phases a component goes through from creation to destruction. There are three main phases.

(1) Mounting : When a component is being created and inserted into the dom.

- Key lifecycle methods : componentDidmount(), constructor(), render().

(2) updating :- When the component's state are props changes causing a re-render.

- key lifecycle methods : componentDidupdate(), shouldComponentupdate(), render().

(3) unmounting :- when a component is being removed from the DOM.

- Key lifecycle method : componentwillunmount()

(4) Discuss refs, Hooks with example.

Refs :- Refs in React are used to directly access and interact with DOM elements or React element created within a component. They allow you to bypass React's state-based re-rendering and interact with element directly, epecially for things like, focusing and input field, scrolling, or playing a video.

Creating a Ref :- React.createRef() method in class component or the useRef() hooks in functional components.

eg
```
import React, { useRef } from 'react';
function FocusInput(){
    const inputRef = useRef(null);
    const handleFocus = () => {
        inputRef.current.focus();
```

```
};
return(
  <div>
    <input type="text" ref={inputRef} />
    <button onClick={handleFocus}> Focus the input </button>
  </div>
);
}
export default FocusInput;
```

Hooks in React :→ Hooks are functions introduced in React 16.8 that allow you to use state & other React features in functional components, which were previously only available in class components.

) UseState: Allows you to manage local state in functional components.

UseEffect: Handles side effect such as data fetching, subscriptions, or DOM manipulations it replace lifecycle methods like componentDidMount, componentDidUpdate.

eg→
```
import React, {useState} from 'react';
function Counter() {
  const [count, setCount] = useState(0);
  return(
    <div>
      <p> current count : {count} </p>
      <button onClick={() => setCount(count+1)}>
        Increment </button>
    </div>
  );}
  export default Counter;
```

(5) Explain different types of Node.JS module with example.

Ans Node.js modules are a fundamental aspect of Node.js development, allowing you to organize and reuse code effectively. There are several types of modules in Node.js:

1) Core modules → Core modules are built into Node.js and don't require installation. They provide essential functionality for various tasks.

- fs (File system) provides file system operation.

eg→
```
Const fs = require('fs');
fs.readFile('example.txt', (err, data)=>{
    if (err) throw err;
    Console.log(data);
});
```

http: Allows creating HTTP servers and clients.

```
Const http = require ('http');
Const server = http.createServer ((req, res)=> {
    res.statusCode=200;
    res.end ("Hello world");
});
server.listen (3000, '127.0.0.1', () => {
    Console.log ('server running!');
});
```

(ii) ES6 modules (ECMAScript 2015) modules are the official standard for modules in Javascript. Node.js support ES6 modules through the .mjs extension or by setting.

eg:- math.mjs

```
export function add(a, b) {
    return a+b;
}
export function subtract(a,b){
    return a-b;
```

(iii) Third-party modules :- Third-party modules are packages created by others and can be install from the npm registry.

ex. npm install express.

```
const express = require('express')
const app = express();
app.get ('/', (req,res)=>{
    res.send ("Hello");
});
app.listen(3000, ()=>{
    console.log ('Server is running
        on http://localhost:3000');
});
```

(6) Explain file system in Node.JS with example.

Ans In Node.Js, the file system module (fs) is used to interact with the file System on your Computer. it allows you to read, write, update, delete files, & work with directories. The fs module is built-in, so you don't need to install any external dependencies.

Basic operation:

1) Reading a file : you can read data from a file.

2) writing to a file : you can write data to a file, and if the file doesn't exist, it will create a new file.

3) Appending to a file : you can append data to an existing file.

4) Deleting a file; you can delete a file.

5) working with directories :→ you can create, read & remove directories.

Reading a file →
```
fs.readFile('example.txt', 'utf8', (err, data) => {
    if (err) {
        console.error(err);
        return;
    }
    console.log(data);
});
```

2) writing a file :-

```
fs.writeFile ('example.txt',
    'Hello', (err) => {
        if (err) {
            console.error (err);
            return;
        }
        console.log ('File written');
    });
```

3) Deleting a file; →

```
fs.unlink ('example.txt', (err) => {
    if (err) {
        console.error (err);
        return;
    }
    console.log ('File deleted');
});
```

The fs module in Node.js provides a powerful way to intract with the file system, allowing you to perform various task like reading, writting, appending, deleting etc.