Date :- 21/01/2025

Module 01

Questions :-

1. What are the benefits of using css in web development ?

→ css (Cascading Style Sheets) is used to style and format web pages.

- Benefits include :-
  - Separating of content and design for easier maintenance.
  - Faster page loading by reducing HTML code.
  - Consistent design across multiple pages.
  - Responsive layouts for Different screen size.
  - Better accessibility and user experience.

2. Explain what responsive design means in the context of web development.

→

- Responsive design ensures that web pages adapt to different screen size and devices.
- It uses flexible grids, media queries, and scalable image to provide an optimal user experience on desktops, tablets and mobile phones.

3. What is Bootstrap and what role does it play in web development.

→
- Bootstrap is a css framework that helps in designing responsive and mobile-first web pages quickly.
- It provide predefined css classes, components and Javascript plugins to simplify web Development.
- Features include a responsive grid systems, reusable components and built-in styles.

4. Briefly explain the syntax of Javascript.

→
- Javascript syntax include variables, function loops and Conditions.
- It follows ECMAScripts standards and is used for dynamic web interactions.
- Examples:-

```
let x = 10;
function greet(){
    console.log("welcome! ur !");
}
if (x > 5){
    console.log("x is greater than 5");
}
```

5. Describe one JavaScript inbuilt object and its purpose.

→

- One important JavaScript inbuild object is the Date object.
- It is used to work with dates & times.
- Examples:-

```
let currentDate = new Date();
console.log(currentDate);
```

- The Date object provides methods like getDate(), getMonth() and getFullYear() for data manipulations.

6. What is error handling in JavaScript and why is it important?

→

- Error handling allows developers to catch and manage errors efficiently.
- Prevents program crashes and ensures smooth user experience.
- Uses try-catch blocks to handle exceptions.

- Example:-

```
try {
    let x = y + 10;
} catch (error) {
    console.log("An error occurred: " +
                error.message);
}
```

7. Explain what event handling is in Javascript

→

- Event handling is the process of responding to user interactions (clicks, key presses, form submissions).
- Uses event listeners like addEventListener()
- Helps create interactive and dynamic web applications.
- Example:-

```
document.getElementById("btn").
    addEventListener("click", function(){
        alert("Button clicked!");
});
```

8. What is the DOM and why is it important in web development?

→

- DOM (Document Object Model) represents an HTML document as a tree structure.
- Allows JavaScript to access, modify and manipulate web page elements dynamically.
- Example:-

```
document.getElementById("title").innerText
        ="New Heading";
```

- Important for creating dynamic content, handling user interactions and modifying elements without reloading the page.

9. Define asynchronous programming in Javascript.

→

- Asynchronous programming allows code execution without blocking other operations.
- Uses callbacks, promises & async/await for handling tasks like API calls.
- Essential for handling APIs, database queries and long-running operations efficiently.
- Examples:-

```
fetch("https://api.github.com/users/ustav205")
  .then(response => response.json())
  .then(ustav205 => console.log(ustav205))
  .catch(error => console.log("Error:",
                              error));
```

10. Discuss the key components of css syntax with examples.

→

- Selector :- Specifies the HTML element to style.
- Property :- Define the aspect to modify (color, font, size).
- Value :- The assigned setting for the property.
- Example:-

```
h1
    color: blue;
    font-size: 24px;
    text-align: center;
}
```

- Other key components:-

- classes & IDs :- .class{} & #id {}
- Pseudo-classes :- :hover, :focus
- Box Model :- margin, padding, border.
- Media Queries for responsiveness.

11. Compare and contrast CSS and inline stylesheets, highlighting their advantages and disadvantage.

→

Feature	CSS Stylesheets	Inline Stylesheets
Definition	Separate .css file or <style> tag in <head>	Style applied directly within the element using style=""
Readability	Improves readability, keeps HTML clean	Clutters HTML, making it harder to maintain.
Reusability	Can be used across multiple pages	Limited to a single element.
performance	Faster loading as styles are cached	Slower since styles are applied individually
Example	body{ background-color: lightblue; }	Hello

12. Differentiate between synchronous and asynchronous programming in JavaScript.

→

Feature	Synchronous Programming	Asynchronous Programming
Execution Order	Code runs line-by-line, blocking the next task.	Tasks run independently, without blocking.
Example	`console.log("task-1");` `console.log("task-2");`	`setTimeout(()⇒ console.log("task2"),1000);` `console.log("task-1");`
Performance	Slower for time-consuming tasks.	More efficient, especially for API calls.
Use Cases	Simple scripts, calculations.	Fetching data, user interactions.

13. Discuss the importance of event handling in javascript and provide examples of events.

→

- Event handling enables web applications to respond to user interactions dynamically.
- Common Events :-

- click Event :- onclick()
- Keyboard Event :- onkeypress, onkeydown.

- Mouse Events :- onmouseover, onmouseleave
- Form Events :- onchange, onsubmit
- Examples :-

```
document.getElementById("btn").addEventListener("click", function() {
    alert("Button clicked!");
});
```

- Importance :-
  - Enhances user experience and interactivity
  - Triggers dynamic changes like animations, form validation etc.

14. What is a Promise in Javascript?

→
- A Promise is an object that handles asynchronous operations and represents a value that will be available in the future.
- It has three states :
  1. Pending :- Initial state.
  2. Fulfilled :- Operation successful (resolve).
  3. Rejected :- Operation failed (reject).
- Example :-

```
let promise = new Promise((resolve, reject) => {
    let sucess = true;
    sucess ? resolve("Success!") : reject("Failed!");
});
```

15. Explain the three state of a Promise.

→

1. Pending :- the Promise is created but hasn't completed.
2. Fulfilled (Resolved) :- the operation was successful, and .then() executes.
3. Rejected :- the operation failed, and .catch() handles error.

- Example :-

```
let promise = new Promise((resolve, reject)
    => { setTimeout(() => resolve("Data loaded"),
        2000);
});
promise.then(data => console.log(data)).
    catch(error => console.log(error));
```

16. What are the two main methods associated with a Promise object?

→

- The two main methods associated with a Promise object are :

1. .then() :- Used to handle the resolved value of a Promise.
2. .catch() :- Used to handle any error that occurs during the Promise execution.

17. How do you handle asynchronous operations using Promises?

→
- Asynchronous operations are handled using Promises by chaining .then() for successful execution and .catch() for error handling.
- Example :-

```
fetch("https://api.example.com/data")
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(error => console.error("Error:",
                                error));
```

18. What is chaining in Promises? Provide an Example.

→
- chaining in Promises refers to linking multiple .then() methods to handle sequential asynchronous operations.
- Example :-

```
fetch("https://api.example.com/user")
  .then(response => response.json())
  .then(user => fetch(`https://api.example
        .com/orders/${user.id}`))
  .then(response => response.json())
  .then(orders => console.log(orders))
  .catch(error => console.error("Error:",
                                error));
```

19. How can you create a new Promise? Provide an example.

→

- A new Promise can be created using the Promise constructor, which takes a function with resolve and reject parameters.
- Example:-

```
Const myPromise = new Promise ((resolve,
    reject) => {

    let success = true;
    if(success){
        resolve("Promise resolved successfully.");
    }else{
        reject("Promise rejected.");
    }
});

myPromise
  .then(message => console.log(message))
  .catch(error => console.error(error));
```

20. What is async/await in JavaScript?

→

- async/await is a modern way to handle asynchronous operations in JavaScript, making code easier to read and write.
- the async keyword defines an asynchronous function, and await pauses execution

until the Promise resolves.

- Example:-

```
async function fetchData() {
    let response = await fetch("https://
        api.example.com/data");
    let data = await response.json();
    console.log(data);
}
fetchData();
```

21. How do you handle error in async/await functions?

→

- Error in async/await functions are handled using try...catch blocks.
- Example:-

```
async function fetchData() {
    try {
        let response = await fetch("https://
            api.example.com/data");
        let data = await response.json();
        console.log(data);
    } catch (error) {
        console.error("Error fetching data:",
            error);
    }
}
fetchData();
```

22. How do you use async/await with Promise.all()?

→

- Promise.all() is used with async/await to execute multiple promises in parallel and await for all to resolve before continuing.
- Example:

```
async function fetchMultipleData() {
  try {
    let [users, posts] = await Promise.all([
      fetch("https://api.example.com/users").
        then(res => res.json()),
      fetch("https://api.example.com/posts").
        then(res => res.json());
    ]);
    console.log("users: ", users);
    console.log("Posts: ", posts);
  } catch (error) {
    console.error("Error fetching data:",
      error);
  }
}
fetchMultipleData();
```