Animesh Parab 88 T2 – T21

IP WRITTEN ASSIGNMENT 1

1. **Compare XML and JSON.** **Co 1**

ANS- Parameter	**JSON**	**XML**
1.Full form	It stands for JavaScript Object Notation.	It stands for Extensible Markup Language.
2.Format	It is used for representing objects. It has a key-value pair format.	It is a markup language. It uses tags to represent data items.
3.Array support	It supports arrays.	It does not support arrays.
4.Encoding	It supports only UTF-8 encoding.	It supports various other encodings.
5.Comments	It does not support comments.	It supports comments.
6.Readability	It is more compact and is easier for humans to read and write.	It is comparatively more difficult to read and write.
7.Namespace support	It does not support namespaces.	It supports namespaces.

2. **Explain different types of arrow functions.** **Co 2**

ANS-

1) Arrow functions, also called lambda functions are anonymous functions used in programming.

2) There are 3 parts in an arrow function: i. Parameters
ii. Fat arrow notation (=>)
iii. Statements

3) Arrow functions remove the need to type out the 'function' keyword every time we create a function.

4) There are three main types of arrow functions: i. Arrow function with no argument-

ii. Arrow function with parameters-

If a function that does not take any argument, then we should use empty parentheses.
For example-
let greet = () => console.log("Hello");
greet();

iii. Arrow Function with Multiple Statements-

If a function takes only one argument, then we can ignore the parentheses.
For example-
let cube = x => x*x*x;
greet(10);
However, if there are no parameters or more than one parameter, you must enclose the parameter list in parentheses.
If your arrow function needs to include multiple statements, you'll need to use curly braces and an explicit `return` statement.
For example-

```
const largerNumber = (a, b) => {
if (a > b) {
return a;
} else {
return b;
}
};
```

3. What is DNS? Explain working of DNS. Co 1
ANS-
1) A Domain Name System (DNS) turns domain names into IP addresses, which allows browsers to access websites and other internet resources. DNS thus allows us to access an online resource without needing to know its IP address.

2) Web browsing and most other internet activities rely on DNS to quickly provide the information necessary to connect users to remote hosts. DNS mapping is distributed throughout the internet in a hierarchy of authority. They also typically run DNS servers to manage the mapping of those names to those addresses. Most Uniform Resource Locators (URLs) are built around the domain name of the web server that takes client requests.

3) The steps involved in the working of DNS are: i. The user enters a web address or domain name into a browser.

ii. The browser sends a message, called a recursive DNS query, to the network to find out which IP or network address the domain corresponds to.

iii. The query goes to a recursive DNS server, which is also called a recursive resolver, and is usually managed by the internet service provider (ISP). If the recursive resolver has the address, it will return the address to the user, and the webpage will load.

iv. If the recursive DNS server does not have an answer, it will query a series of other servers in the following order: DNS root name servers, top-level domain (TLD) name servers and authoritative name servers.

v. The three server types work together and continue redirecting until they retrieve a DNS record that contains the queried IP address. It sends this information to the recursive DNS server, and the webpage the user is looking for loads. DNS root name servers and TLD servers primarily redirect queries and rarely provide the resolution themselves.

vi. The recursive server stores, or caches, the A record for the domain name, which contains the IP address. The next time it receives a request for that domain name, it can respond directly to the user instead of querying other servers.

vii. If the query reaches the authoritative server and it cannot find the information, it returns an error message.

4. Explain promises in ES6. Co 2
ANS-
1) In JavaScript, Promises are a programming construct introduced in ECMAScript 6 (ES6) to help manage asynchronous operations in a more organized and readable manner. Asynchronous operations are tasks that may take some time to complete, such as fetching data from a server or reading a file, and they don't block the execution of other code while waiting for their completion. Prior to Promises, managing asynchronous operations often involved deeply nested callback functions, leading to a pattern known as "callback hell".

2) Promises provide a cleaner way to handle asynchronous code by abstracting the process of handling success and failure outcomes. Promises represent a value that may not be available yet, but will be resolved or rejected at some point in the future. They have three states: i. Pending: The initial state. The Promise is neither fulfilled nor rejected; it's still in progress.

ii. Fulfilled: The asynchronous operation has completed successfully, and the Promise is fulfilled. It transitions to this state with a result value.

iii. Rejected: The asynchronous operation has encountered an error, and the Promise is rejected. It transitions to this state with a reason for the rejection.

3) Here's how you can use Promises in JavaScript, specifically ES6:

```
// Creating a new Promise
const myPromise = new Promise((resolve, reject) => {
// Simulate an asynchronous operation
setTimeout(() => {
const randomNumber = Math.random();

if (randomNumber > 0.5) {
resolve(randomNumber); // Resolve the Promise with a value
} else {
reject("Value is too low"); // Reject the Promise with a reason
}
}, 1000); // Simulating a delay of 1 second
});
// Using the Promise
myPromise
then(result => {
console.log("Fulfilled with result:", result);
```

```
})
.catch(error => {
console.error("Rejected with error:", error);
});
```
In this example, `myPromise` represents an asynchronous operation that simulates generating a random number. Depending on the value of the random number, the Promise either resolves or rejects. The `.then()` method is used to handle the fulfilment case, and the `.catch()` method is used to handle the rejection case.
4) Promises also support chaining, allowing you to sequence asynchronous operations more effectively:

```
function fetchUserData() {
return fetch('https://api.example.com/user')
then(response => response.json())
then(userData => {
console.log('User data:', userData);
})
.catch(error => {
console.error('Error fetching user data:', error);
});
}
```
In this example, the `fetchUserData` function returns a Promise that fetches user data from an API. The `.then()` methods are chained to process the response and handle potential errors.