IP Theory Assignment 1

Roll No – 71 Batch – T14

1. Compare XML and JSON. **CO1**

Ans:

	XML	JSON
1	It is Extensible markup language	It is JavaScript Object Notation
2	It supports namespaces.	It does not provides any support for namespaces.
3	It is derived from SGML.	It is based on JavaScript language.
4	It supports comments.	It doesn't supports comments.
5	Supports different data types, including text, numbers, dates, and more complex structures.	Supports basic data types like strings, numbers, Booleans, arrays, and objects.
6	Commonly used in configuration files, document storage, and data exchange between heterogeneous systems.	Widely used for web APIs, configuration files, and data interchange in modern web applications.
7	Uses a tag-based structure where data is enclosed in elements surrounded by angle brackets ("<tag>data</tag>"). It is more verbose and human-readable.	Utilizes a key-value pair format where data is represented as object properties ("key": "value"). It is more concise and often easier for machines to parse.

2. Explain different types of arrow functions. **CO2**

Ans :

Arrow functions, also known as arrow function expressions, were introduced in JavaScript as a concise way to write anonymous functions. They have a more compact syntax compared to traditional function expressions, and they also have some specific behavior and limitations. There are different types of arrow functions based on their use cases and variations:

A] Basic Arrow function

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

B] Arrow function with no parameter

```
const greet = () => "Hello!";
```

C] Arrow Function with Single Parameter:

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

D] Arrow Function with Multiple Statements:

```
const printMessage = message => {
  console.log("Starting...");
  console.log(message);
  console.log("Ending...");
};
```

E] Arrow Function with Object Return:

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

F] Arrow Function as Callback:

```
const numbers = [1, 2, 3, 4, 5];
const squared = numbers.map(num => num * num);
```

G] Arrow Function in Higher-Order Functions:

```
const getEvenNumbers = numbers.filter(num => num % 2 === 0);
```

H] Arrow Function and this Context:

```
function Counter() {
  this.count = 0;
  setInterval(() => {
    this.count++;
    console.log(this.count);
  }, 1000);
}
```

3. What is DNS? Explain working of DNS. **CO1**

Ans:

DNS stands for Domain Name System. It is a fundamental technology that enables the translation of domain names (like www.darshilmarathe.me) into IP addresses (like 192.0.23.12) that computers use to identify each other on a network. DNS plays a crucial role in the functioning of the internet by providing a distributed and hierarchical system for mapping domain names to IP addresses.

Working of DNS:

1] Domain Name Resolution Request:

When a domain name is entered in a web browser's address bar or any other application that needs to access a remote server, a DNS resolution request is initiated. This request is sent to a DNS resolver (also known as a DNS server or a DNS recursive resolver), which is typically provided by Internet Service Provider (ISP) or another DNS service.

2] Local Cache Check:

The DNS resolver first checks its local cache to see if it has a recent record of the domain name and its corresponding IP address. If the information is found in the cache and is still valid, the resolver returns the IP address directly to the requesting application, and the process is complete.

3] Recursive Query:

If the domain name is not found in the local cache or the cached information has expired, the DNS resolver needs to find the IP address through a process called recursion. It queries the root DNS servers for the top-level domain (TLD) of the domain name (e.g., ".com"), asking for the authoritative name servers responsible for the TLD.

4] TLD Name Server Lookup:

The root DNS servers respond with a referral to the TLD name servers for the requested domain (e.g., the ".com" TLD name servers). The resolver then queries the TLD name servers to find the authoritative name servers responsible for the specific domain (e.g., "example.com").

5] Authoritative Name Server Query:

The TLD name servers respond with the IP addresses of the authoritative name servers for the domain "darshilmarathe.me" The DNS resolver then queries one of these authoritative name servers to obtain the actual IP address associated with the requested domain name.

6] IP Address Response:

The authoritative name server provides the IP address for the requested domain name back to the DNS resolver.

7] Caching and Response:

The DNS resolver caches the IP address information locally for a certain period (time to live, or TTL). It then returns the IP address to the requesting application, which can now use it to establish a connection to the remote server.

8] Subsequent Queries:

If other devices or applications on the same network request the same domain name, the DNS resolver can provide the cached IP address, reducing the need to repeat the entire resolution process.

4. **Explain Promises in ES6.** **CO2**

Ans:

Promises are essential tools for managing asynchronous operations in JavaScript. They are special objects that represent the eventual outcome of an asynchronous task. A Promise can produce either a successful value upon completion or an error if the operation fails. Promises are created using the Promise constructor, which takes an executor function as an argument. This function must call either the resolve or reject callback to signal success or failure.

Promise objects have internal properties like state and result, which represent the current status and outcome of the promise. The state can be "pending," "fulfilled," or "rejected." Once a promise is settled (resolved or rejected), further calls to resolve or reject are ignored.

To handle successful outcomes, you can use the .then() method, passing a callback function that will execute when the promise is resolved. Similarly, you can use .catch() to handle errors that occur during the promise's execution.

To ensure cleanups or actions regardless of success or failure, you can use the .finally() method, which is executed irrespective of the promise's outcome. This method is useful for performing tasks like stopping loaders or closing connections.

Using null as the first argument in .then() to handle errors is not recommended; it's better to use .catch() for error handling. .catch() provides a cleaner and more standardized way to manage errors.

In summary, promises are important for managing asynchronous operations in JavaScript. They allow you to handle success and errors more elegantly, making your code more readable and maintainable.

The below code demonstrates promises in Javascript:

```
const myPromise = new Promise((resolve, reject) => {
    setTimeout(() => {
      const success = true; //success or failure
      if (success) {
        resolve("Operation successful!");
      } else {
        reject(new Error("Operation failed!"));
      }
    }, 2000);
  });

  // handle successful outcome
  myPromise.then(
    result => {
      console.log(result);
    }
).catch(
  error => {
    console.error(error);
  }
).finally(() => {
  console.log("Promise execution completed.");
});

// Using .catch() to handle errors
const anotherPromise = new Promise((resolve, reject) => {
  setTimeout(() => {
    reject(new Error("Another operation failed!"));
  }, 1500);
});

anotherPromise.catch(
  error => {
    console.error("Caught an error:", error);
  }
).finally(() => {
  console.log("Another promise execution completed.");
});
```