

# **INP-2024-December-PYQ Answers**

## **Q1. [5 Marks]**

- a. Differentiate between HTML and XML.
- b. How to declare variable in ES6.
- c. What is React JSX.

## **Q2. [10 Marks]**

- a. Write a ReactJS code for creating 2 components and refer those components on webpage.
- b. What is NodeJS? Explain different types of NodeJS modules.

## **Q3. [10 Marks]**

- a. Differentiate between MVC, FLUX and REDUX.

## **Q4. [10 Marks]**

- a. What is DNS? Explain working of DNS.

## **Q5. [10 Marks]**

- a. Write a program in Express JS to display Hello World on browser?

## **Q6. [5 Marks]**

- a. Write a short note on any 4
  - NPM

## Q1. [5 Marks] - Answers

### a. Differentiate between HTML and XML.

Feature	HTML (HyperText Markup Language)	XML (eXtensible Markup Language)
Purpose	Designed to display data and create web pages	Designed to store and transport data
Focus	Presentation-focused	Data-focused
Tag Predefined	Uses predefined tags like <code>&lt;p&gt;</code> , <code>&lt;h1&gt;</code> , <code>&lt;div&gt;</code>	User-defined tags (e.g., <code>&lt;student&gt;</code> , <code>&lt;price&gt;</code> )
Case Sensitivity	Not case-sensitive ( <code>&lt;BODY&gt;</code> = <code>&lt;body&gt;</code> )	Case-sensitive ( <code>&lt;Data&gt;</code> ≠ <code>&lt;data&gt;</code> )
Error Handling	Tolerant to minor errors	Errors can break parsing
Use Case	Web page layout and formatting	Data exchange between systems (e.g., APIs)

### b. How to declare variable in ES6.

#### Declaring Variables in ES6

In ES6 (ECMAScript 2015), variables are declared using two modern keywords: `let` and `const`. These provide better control over scope and behavior compared to the older `var` keyword.

- `let` is used to declare variables that can be updated later. It is **block-scoped**, meaning it is only accessible within the block `{}` where it is defined.
- `const` is used to declare **constants**—variables whose value cannot be reassigned. It is also block-scoped and must be initialized at the time of declaration.
- `var`, from earlier versions of JavaScript, is **function-scoped** and allows redeclaration. It is generally avoided in modern ES6 code due to scope-related issues.

## Example:

```
let name = "Sameer"; // Can be reassigned
const age = 21;     // Cannot be reassigned
var city = "Mumbai"; // Function-scoped (legacy)
```

## c. What is React JSX?

### React JSX (JavaScript XML)

#### Definition

JSX stands for **JavaScript XML**.

It is a **syntax extension** of JavaScript used in **ReactJS** to describe the structure of the **User Interface (UI)**.

JSX allows developers to **write HTML-like code** directly inside JavaScript, making code more **readable and expressive**.

#### Key Features

- Combines **HTML and JavaScript** in the same file.
- **Transpiled by Babel** into `React.createElement()` calls.
- Must return a **single parent element**.
- Allows embedding **JavaScript expressions** inside `{}`.
- Increases **readability** and **reduces code complexity**.

## Example:

```
function Welcome() {
  const name = "Sameer";
  return <h2>Hello, {name}! Welcome to React.</h2>;
}
```

## Q2. [10 Marks] - Answers

a. Write a ReactJS code for creating 2 components and refer those components on webpage.

```
// index.js
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App';

// Rendering the main App component to the webpage
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<App />);
```

```
// App.js
import React from 'react';
import Header from './Header';
import Footer from './Footer';

// Main App component that refers to two child components
function App() {
  return (
    <div>
      <Header />
      <h2>Welcome to My React Webpage</h2>
      <p>This page demonstrates multiple React components.</p>
      <Footer />
    </div>
  );
}
```

```
export default App;
```

```
// Header.js
import React from 'react';

function Header() {
  return (
    <header style={{ background: '#4CAF50', color: 'white', padding: '10px' }}>
      <h1>React Components Example</h1>
    </header>
  );
}

export default Header;
```

```
// Footer.js
import React from 'react';

function Footer() {
  return (
    <footer style={{ background: '#333', color: 'white', padding: '10px', marginT
op: '20px' }}>
      <p>© 2025 My React Website. All rights reserved.</p>
    </footer>
  );
}

export default Footer;
```

## Explanation

- **Header Component:** Displays the page title using inline styles.
- **Footer Component:** Displays copyright information.
- **App Component:** Acts as the **parent** and refers to both child components (`Header` and `Footer`).
- **index.js:** Renders the `App` component into the root `<div>` of the webpage.

## b. What is NodeJS? Explain different types of NodeJS modules.

### What is Node.js?

Node.js is an **open-source, cross-platform** runtime environment that allows developers to run JavaScript code **outside the browser**, typically on the server side. It is built on the **V8 JavaScript engine** (used by Chrome) and is known for its **event-driven, non-blocking I/O model**, which makes it ideal for building scalable and high-performance applications like web servers, APIs, and real-time apps.

### Types of Node.js Modules

Node.js uses **modules** to organize code into reusable units. There are three main types:

#### 1. Core Modules

- Built into Node.js and do not require installation.
- Provide essential functionalities like file system access, networking, and streams.
- Examples:
  - `fs` – File system operations
  - `http` – Create web servers
  - `path` – Handle file paths

- o `os` – Get system information

```
const fs = require('fs');
```

## 2. Local Modules

- Created by developers to structure their own application logic.
- Can be reused across files and projects.
- Typically stored in `.js` files and imported using `require()` or `import`.

```
// greet.js
module.exports = function () {
  console.log("Hello, Sameer!");
};

// app.js
const greet = require('./greet');
greet();
```

## 3. Third-Party Modules

- Installed via **npm (Node Package Manager)**.
- Provide additional features like routing, database access, authentication, etc.
- Examples:
  - o `express` – Web framework
  - o `mongoose` – MongoDB ODM
  - o `lodash` – Utility functions

```
npm install express
```

```
const express = require('express');
```

## Q3. [10 Marks] - Answers

### a. Differentiate between MVC, FLUX and REDUX.

Feature	MVC (Model-View-Controller)	Flux	Redux
Architecture Type	Triangular (Model ↔ View ↔ Controller)	Unidirectional data flow	Unidirectional data flow
Data Flow	Bidirectional	One-way (Action → Dispatcher → Store → View)	One-way (Action → Reducer → Store → View)
Components	Model, View, Controller	Action, Dispatcher, Store, View	Action, Reducer, Store, View
State Management	Spread across Model and Controller	Centralized in Store	Centralized in a single immutable Store
Complexity	Can become complex with large apps	More structured than MVC	Simplified and predictable
Use Case	Traditional web apps (e.g., Java, .NET)	React applications	React + modern JS apps
Middleware Support	Not built-in	Limited	Strong (e.g., Redux Thunk, Redux Saga)
Debugging	Harder due to bidirectional flow	Easier with dev tools	Very easy with time-travel debugging
Example Frameworks	AngularJS (early), ASP.NET MVC	Facebook's React with Flux	React with Redux

## Q4. [10 Marks] - Answers

### a. What is DNS? Explain working of DNS.

#### What is DNS?

DNS stands for **Domain Name System**. It is like the **phonebook of the internet**, translating **human-readable domain names** (e.g., [www.google.com](http://www.google.com)) into **IP addresses** (e.g., [142.250.182.4](http://142.250.182.4)) that computers use to locate each other on the network.

Without DNS, users would need to remember complex numerical IP addresses to access websites, which is impractical.

#### Working of DNS (Step-by-Step)

##### 1. User Request

- You type a domain name (e.g., [www.example.com](http://www.example.com)) into your browser.

##### 2. Browser Cache Check

- The browser first checks its local cache to see if it already knows the IP address.

##### 3. Operating System Cache

- If not found, the OS checks its own DNS cache.

##### 4. DNS Resolver (ISP)

- If still unresolved, the request goes to your ISP's **DNS resolver**, which begins the lookup process.

##### 5. Root DNS Server

- The resolver contacts a **root server**, which directs it to the correct **Top-Level Domain (TLD)** server (e.g., `.com`, `.org`).

## 6. TLD Server

- The TLD server points to the **authoritative name server** for the domain.

## 7. Authoritative Name Server

- This server provides the final IP address for `www.example.com`.

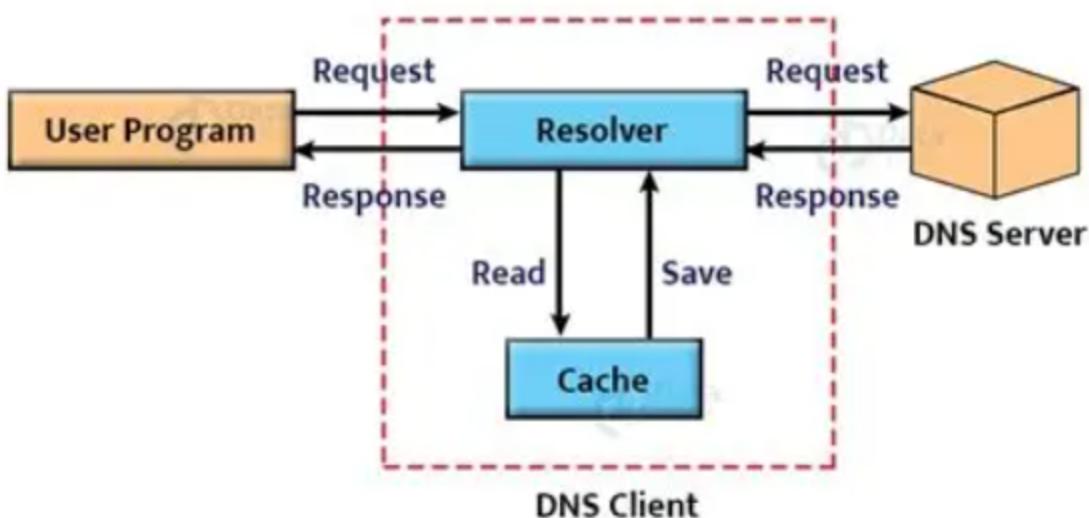
## 8. Response to Browser

- The resolver sends the IP address back to your browser, which then makes a request to the web server.

## 9. Website Loads

- The server responds with the website content, and the page is displayed.

# Working of DNS



## Q5. [10 Marks] - Answers

### a. Write a program in Express JS to display Hello World on browser?

#### Step 1: Install Node.js

- Make sure **Node.js** is installed.
- Verify by running in terminal:

```
node -v  
npm -v
```

#### Step 2: Initialize a Project

- Create a new folder and navigate into it:

```
mkdir myExpressApp  
cd myExpressApp
```

- Initialize npm to create **package.json**:

```
npm init -y
```

#### Step 3: Install Express

```
npm install express
```

#### Step 4: Create the App File

- Create **app.js** (or **index.js**) in the project folder.

#### Step 5: Write the Hello World Program

```
// Step 5.1: Import express
const express = require('express');

// Step 5.2: Create an Express application
const app = express();

// Step 5.3: Define a route
app.get('/', (req, res) => {
  res.send('Hello World');
});

// Step 5.4: Start the server
app.listen(3000, () => {
  console.log('Server is running on http://localhost:3000');
});
```

### Explanation of Steps:

1. **Import Express** → Load the Express library.
2. **Create App** → `express()` initializes the app.
3. **Define Route** → `app.get()` handles GET requests to `/` and sends a response.
4. **Start Server** → `app.listen()` makes the app listen on port 3000.

---

### Step 6: Run the App

```
node app.js
```

- Open browser and go to:

```
http://localhost:3000
```

- You should see:

```
Hello World
```

## Q6. [5 Marks] - Answers

### a. Write a short note on NPM

#### NPM (Node Package Manager)

##### Definition

NPM stands for **Node Package Manager**.

It is the **default package manager for Node.js**, used to **install, manage, and share** reusable JavaScript packages (libraries and tools) for both **frontend and backend** development.

##### Key Features

- **Package Installation:** Installs open-source libraries from the npm registry using simple commands.
- **Dependency Management:** Automatically handles all dependencies required by a project.
- **Version Control:** Allows specifying package versions to maintain compatibility.
- **Custom Scripts:** Enables running build, test, and deploy scripts.
- **Global & Local Packages:** Supports both project-specific (local) and system-wide (global) installations.

##### Common Commands

```
npm init      # Initialize a new project (creates package.json)
npm install react # Install React library
npm uninstall xyz # Remove a package
npm update      # Update installed packages
```