1.2 WEB BROWSER

1.2.1 What is a Browser ?

GQ. What is web browser? Discuss working of web browser in detail. **(10 Marks)**

– A browser is a software program that is used to explore, retrieve, and display the information available on the World Wide Web. This information may be in the form of pictures,

web pages, videos, and other files that all are connected via hyperlinks and categorized with the help of URLs (Uniform Resource Identifiers). For example, you are viewing this page by using a browser.

- A browser is a client program as it runs on a user computer or mobile device and contacts the webserver for the information requested by the user. The web server sends the data back to the browser that displays the results on internet supported devices. On behalf of the users, the browser sends requests to web servers all over the internet by using HTTP (Hypertext Transfer Protocol). A browser requires a smartphone, computer, or tablet and internet to work.
- **Example of Web Browsers:** Chrome, Firefox, IE, Netscape, Safari, Edge, Opera, UC Browser, Vivaldi etc.

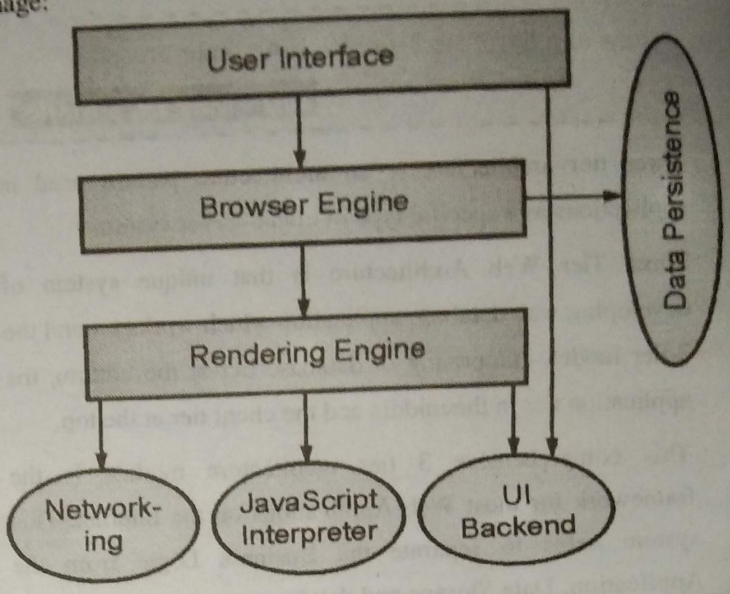
1.2.2 Features of Web Browser

Most Web browsers offer common features such as :

1. **Refresh button :** Refresh button allows the website to reload th ontents of the web pages. Most of the web browsers store local copies of visited pages to enhance the performance by using a caching mechanism. Sometimes, it stops you from seeing the updated information; in this case, by clicking on the refresh button, you can see the updated information.
2. **Stop button :** It is used to cancel the communication of the web browser with the server and stops loading the page content. For example, if any malicious site enters the browser accidentally, it helps to save from it by clicking on the stop button.
3. **Home button :** It provides users the option to bring up the predefined home page of the website.
4. **Web address bar :** It allows the users to enter a web address in the address bar and visit the website.
5. **Tabbed browsing :** It provides users the option to open multiple websites on a single window. It helps users to read different websites at the same time. For example, when you search for anything on the browser, it provides you a list of search results for your query. You can open all the results by right-clicking on each link, staying on the same page.
6. **Bookmarks :** It allows the users to select particular website to save it for the later retrieval of information, which is predefined by the users.

1.2.3 Component of a Web Browser

The primary components of a browser are shown in the below image:



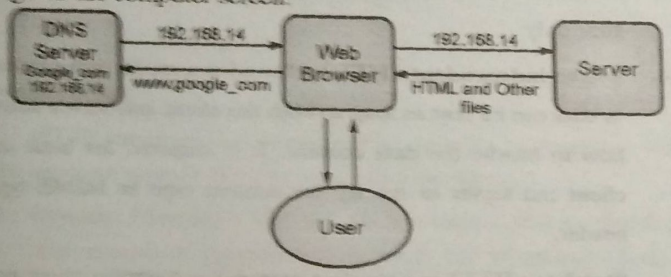
(1A2)**Fig.1.2.1: 3-tier Web Architecture**

1. **User Interface :** The user interface is an area where the user can use several options like address bar, back and forward button, menu, bookmarking, and many other options to interact with the browser.
2. **Browser Engine :** It connects the UI (User Interface) and the rendering engine as a bridge. It queries and manipulates the rendering engine based on inputs from several user interfaces.
3. **Rendering Engine :** It is responsible for displaying the requested content on the browser screen. It translates the HTML, XML files, and images, which are formatted by using the CSS. It generates the layout of the content and displays it on the browser screen. Although it can also display the other types of content by using different types of plugins or extensions. such as :
   - Internet Explorer uses Trident
   - Chrome & Opera 15+ use Blink
   - Chrome (iPhone) & Safari use Webkit
   - Firefox & other Mozilla browsers use Gecko
4. **Networking :** It retrieves the URLs by using internet protocols like HTTP or FTP. It is responsible for maintaining all aspects of Internet communication and security. Furthermore, it may be used to cache a retrieved document to reduce network traffic.

5. **JavaScript Interpreter :** As the name suggests, JavaScript Interpreter translates and executes the JavaScript code, which is included in a website. The translated results are sent to the rendering engine to display results on the device screen.
6. **UI Backend :** It is used to draw basic combo boxes and Windows (widgets). It specifies a generic interface, which is not platform-specific.
7. **Data Storage :** The data storage is a persistence layer that is used by the browser to store all sorts of information locally, like cookies. A browser also supports different storage mechanisms such as IndexedDB, WebSQL, localStorage, and FileSystem. It is a database stored on the local drive of your computer where the browser is installed. It handles user data like cache, bookmarks, cookies, and preferences.

1.2.4 Working of Web Browser

World Wide Web works on the client-server model. A user computer works as a client which can receive and send data to the server. When a web page is requested by a user, the browser contacts the requested server (where the website is stored) and by fetching and interpreting the requested files, it displays the web page on the computer screen.

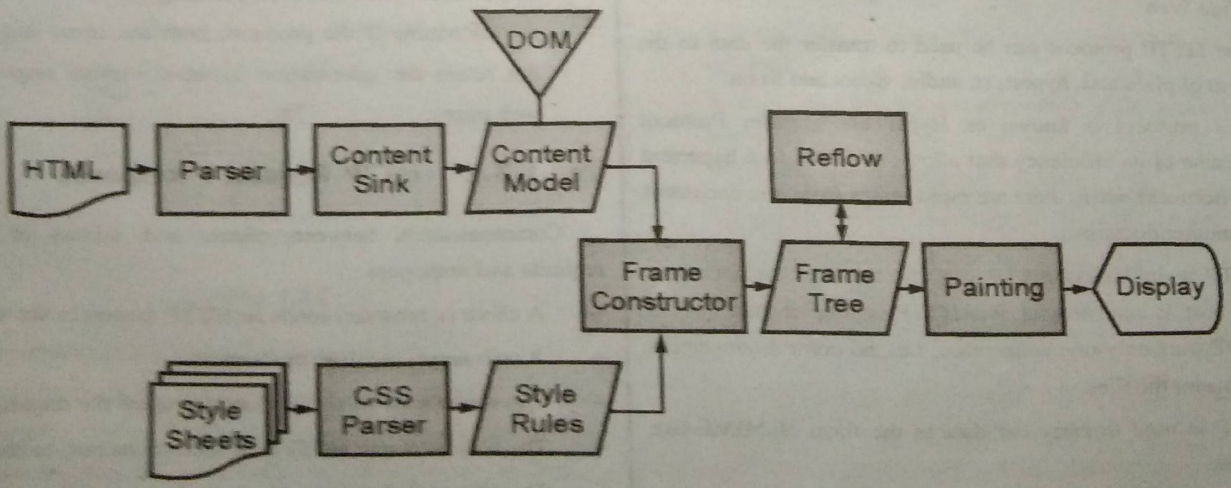


(TA8)Fig.1.2.2 : Working of Web browser

The whole process takes place in these three steps:

1. **Contact to DNS Server :** When a user enters a URL into the address bar and hits 'enter', at first browser contacts the DNS server. A DNS server stores the IP addresses of the server associated with the corresponding domain names. The DNS server takes the domain name from the browser and returns the corresponding IP address to the browser.
2. **Contact to Server :** After getting the IP address of the server for the requested webpage, browser sends a request to that server for the desired files. For example, consider the following URL:http://www.abc.com/articles

   This URL is divided into three parts. First one is HTTP – it is a protocol named Hyper Text Transfer Protocol which defines the way browser communicates with the server. The second part is www.abc.com which is translated by the DNS server with the IP address. It is the address of a computer (Web Server) where the requested web page is stored. The third part is 'articles' which tells the address of the file that is located in the root folder of the website. HTTP protocols are used to transfer the webpage named 'articles' to the browser. The protocol decides the format as well as the methods of communication between web client and server.
3. **Rendering :** The entire process followed by a browser from fetching the webpage to displaying it on the screen is called Rendering. It is executed in the following steps :



(TA6)Fig.1.2.3 : Rendering Process in Web Browsers

a) **Loading HTML :** As soon as the web page is found by the browser, it is loaded in to the browser using the http protocol. All the html tags as well as other resources (like images, flash files, pdf, etc.) referred in the html tags are loaded separately.

b) **Parsing :** An HTML Parser starts interpreting the html files and building subsequent content tree.

c) **Apply Styles :** Web pages may have style sheets attached and a web browser also has its default styles. Using a CSS parser, they are interpreted in order to define the styles (like height, width, spacing, border, color, margin, etc.) for the content specified in the html.

d) **Construct Frames :** Using the HTML and style sheets, the browser constructs the frames. A frame is a rectangular block with various properties like width, height, color, spacing, etc.

e) **Frames Layout :** This process includes arranging all the frames in order to make a layout of the web page.

f) **Frames Painting :** As soon as the layouts are constructed the next process is painting. Painting is the term used for the process to exactly render the objects on the screen and after that, the webpage is displayed on the computer screen. A browser executes all the processes in a flash of a second.

2.6.2 Anonymous Function

An anonymous function can be defined as a function without a name. The anonymous function does not bind with an identifier. It can be declared dynamically at runtime.

An anonymous function can be assigned within a variable. Such expression is referred to as function expression. The syntax for the anonymous function is as follows.

Syntax

```
var y = function( [arguments] )
{
   //code to be executed
}
```

Example

```
var hello = function() {
  console.log('Hello World');
  console.log('I am an anonymous function');
 }
 hello();
```

Parameterized Anonymous function

Example

```
var anon = function(a,b)
{
  return a+b
}
function sum() {
  var res;
  res = anon(100,200);
  console.log("Sum: "+res)
}
sum()
```

2.7 ES6 ARROW FUNCTION

GQ. Give an example of an Arrow function in ES6? List down its advantages. **(10 Marks)**

Arrow functions are anonymous functions (the functions without a name and not bound with an identifier). They don't return any value and can declare without the function keyword. Arrow functions cannot be used as the constructors. The context within the arrow functions is lexically or statically defined. They are also

called as Lambda Functions in different languages. Arrow functions do not include any prototype property, and they cannot be used with the new keyword.

Syntax for defining the arrow function

```
const functionName = (arg1, arg2, ?..) => {
   //body of the function
}
```

There are three parts to an Arrow Function or Lambda Function:

- Parameters: Any function may optionally have the parameters.
- Fat arrow notation/lambda notation: It is the notation for the arrow (=>).
- Statements: It represents the instruction set of the function.

Example

```
// function expression

var myfun1 = function show() {
 console.log("It is a Function Expression");
}
 // Anonymous function

var myfun2 = function () {
   console.log("It is an Anonymous Function");
 }
   //Arrow function

var myfun3 = () => {
   console.log("It is an Arrow Function");
 };

myfun1();
myfun2();
   myfun3();
```

Syntactic Variations

There are some syntactic variations for the arrow functions that are as follows:

- **Optional parentheses for the single parameter**

```
var num = x => {
   console.log(x);
}
   num(140);
```

- **Optional braces for single statement and the empty braces if there is not any parameter required.**

```
var show = () => console.log("Hello World");
   show();
```

2.7.1 Arrow Function with Parameters

If you require to pass more than one parameter by using an arrow function, then you have to pass them within the parenthesis.

Example

```
var show = (a,b,c) => {
   console.log(a + " " + b + " " + c );
}
    show(100,200,300);
```

Arrow function with default parameters

In ES6, the function allows the initialization of parameters with default values, if there is no value passed to it, or it is undefined.

Example

```
var show = (a, b=200) => {
   console.log(a + " " + b);
}
show(100);
```

In the above function, the value of **b** is set to **200** by default. The function will always consider **200** as the value of **b** if no value of **b** is explicitly passed.

Output

```
100 200
```

The default value of the parameter 'b' will get overwritten if the function passes its value explicitly. You can see it in the following example:

Example

```
var show = (a, b=200) => {
   console.log(a + " " + b);
}
show(100,500);
```

Output

```
100 500
```

2.7.2 Advantages of Arrow Functions

1. **It reduces the code size :** When we use the syntax of arrow function, the size of the code gets reduced. We can write less amount of code by using the arrow function.
2. **Return statement and Functional braces are optional for single line functions :** In ES5, we need to define the return statement in the functions, but in ES6, we do not require to define the return statement for single-line functions. Functional braces are also optional for the single-line functions.

   For example

```
In ES5
function show(value){
return value/2;
}
In ES6
var show = value => value/2;
console.log(show(100));
```

If you are not applying curly braces on the single statement, then you do not require to use return, but if you are using curly braces even on the single statement, then you must have to use the return keyword.

```
Without Return Keyword
var show = value => {
  value/2;
}
console.log(show(50));
With Return Keyword
var show = value => {
  return value/2;
}
console.log(show(50));
```

3. **Lexically bind the context :** Arrow function binds the context lexically or statically. The handling of this is different in arrow functions as compared to regular functions. In the arrow function, there is not any binding of this. In regular functions, this keyword is used to represent the objects that called the function, which could either be a window, a button, or a document or anything.

   But with arrow functions, this keyword always represents the object that defines the arrow function.

Example

Consider a class having an array of numbers, and if the number is less than 30, then we will push them into the child queue.

```
In ES5, it can be done as follows :
this.num.forEach(function(num) {
  if (num < 30)
    this.child.push(num);
}.bind(this));
In ES6, it can be done by using the arrow function
this.num.forEach((num) => {
  if (num < 30)
    this.child.push(num);
});
```

By using the arrow function, we do not require to bind it implicitly because it performs automatically by the arrow function. The arrow function makes the writing of function less complicated, and it also reduces the number of lines.

2.8 ES6 HTML DOM

Every web page resides inside a browser window, which can be considered as an object.

A document object represents the HTML document that is displayed in that window. The document object has various properties that refer to other objects which allow access to and modification of the document content. The way a document content is accessed and modified is called the Document Object Model, or DOM. The objects are organized in a hierarchy. This hierarchical structure applies to the organization of objects in a web document.

2.10 ES6 ITERATOR

GQ. Explain ES6 Iterator.

Introduction to Iterator

Iterator is an object which allows us to access a collection of objects one at a time.

The following built-in types are by default iterable :

- String
- Array
- Map
- Set

An object is considered iterable, if the object implements a function whose key is [Symbol.iterator] and returns an iterator. A for...of loop can be used to iterate a collection.

Example

The following example declares an array, marks, and iterates through it by using a for..of loop.

```
<script>
  let marks = [10,20,30]
```

```
//check iterable using for..of
for(let m of marks){
  console.log(m);
}
</script>
```

The output of the above code will be as given below :

```
10
20
30
```

Example

The following example declares an array, marks and retrieves an iterator object. The [Symbol.iterator]() can be used to retrieve an iterator object. The next() method of the iterator returns an object with 'value' and 'done' properties . 'done' is Boolean and returns true after reading all items in the collection.

```
<script>
let marks = [10,20,30]
let iter = marks[Symbol.iterator]();
console.log(iter.next())
console.log(iter.next())
console.log(iter.next())
console.log(iter.next())
</script>
```

The output of the above code will be as shown below –

```
{value: 10, done: false}
{value: 20, done: false}
{value: 30, done: false}
{value: undefined, done: true}
```

Custom Iterable

Certain types in JavaScript are iterable (E.g. Array, Map etc.) while others are not (E.g. Class). JavaScript types which are not iterable by default can be iterated by using the iterable protocol.

The following example defines a class named CustomerList which stores multiple customer objects as an array. Each customer object has firstName and lastName properties.

To make this class iterable, the class must implement [Symbol.iterator]() function. This function returns an iterator object. The iterator object has a function next which returns an object {value:'customer',done:true/false}.

```
<script>
//user defined iterable
class CustomerList {
  constructor(customers){
    //adding customer objects to an array
    this.customers = [].concat(customers)
  }
  //implement iterator function
  [Symbol.iterator](){
    let count=0;
    let customers = this.customers
    return {
      next:function(){
      //retrieving a customer object from the array
        let customerVal = customers[count];
        count+=1;
        if(count<=customers.length){
          return {
            value:customerVal,
            done:false
          }
        }
        //return true if all customer objects are iterated
        return {done:true}
      }
    }
  }
}
//create customer objects
let c1={
  firstName:'Sachin',
  lastName:'Tendulkar'
}
let c2={
  firstName:'Rahul',
  lastName:'Dravid'
}
//define a customer array and initialize it let customers=[c1,c2]
//pass customers to the class' constructor
let customersObj = new CustomerList(customers);
//iterating using for..of
for(let c of customersObj){
```

```
  console.log(c)
}
//iterating using the next() method
let iter = customersObj[Symbol.iterator]();
console.log(iter.next())
console.log(iter.next())
console.log(iter.next())
</script>
```

The output of the above code will be as follows :

```
{firstName: "Sachin", lastName: "Tendulkar"}
{firstName: "Rahul", lastName: "Dravid"}
{
  done: false
  value: {
    firstName: "Sachin",
    lastName: "Tendulkar"
  }
}
{
  done: false
  value: {
    firstName: "Rahul",
    lastName: "Dravid"
  }
}
{done: true}
```

2.11 ES6 GENERATORS

GQ. What do you understand by ES6 Generator function? **(10 Marks)**

ES6 generator is a different kind of function that may be paused in the middle either one or many times and can be resumed later. When the standard function is called, the control rests with the called function until it returns, but the generator in ES6 allows the caller function to control the execution of a called function.

Syntax

The syntax of the generator function is almost identical to regular function. The only real difference is that the generator function is denoted by suffixing the function keyword with an asterisk (*).

In the following syntax, we are showing you some of the valid ways of defining the generator function:

```
function* mygenfun()   // Valid
{
yield 1;
yield 2;
...
...
}

function *mygenfun()   // Valid
{
yield 1;
yield 2;
...
...
}

function*mygenfun() // Valid
{
yield 1;
yield 2;
...
...
}
```

Example

```
function* gen()
{
yield 100;
yield;
yield 200;
}
// Calling the Generator Function
var mygen = gen();
console.log(mygen.next().value);
console.log(mygen.next().value);
console.log(mygen.next().value);
```

Output

```
100
undefined
200
```

The yield statement

The yield statement suspends the function execution and sends a value back to the caller. It retains enough state to enable function to resume where it is left off. When it gets resumed, the function continues the execution immediately after the last yield run. It can produce a sequence of values.

next() method

In the above example, we have used the next() method, which is the main method of the generator. When you call the next() method along with an argument, it will resume the execution of the generator function, replacing the yielded expression where the execution was paused with the argument from the next() method.

The result of the next() method is always an object having two properties:

value: It is the yielded value.

done: It is a Boolean value which gives true if the function code has finished. Otherwise, it gives false.

Example

```
function* show() {
yield 100;
}

    var gen = show();   //here 'gen' is a generator object
    console.log(gen.next()); // { value: 100, done: false }
```

Output

```
{value: 100, done: false }
```

Generator object

The generator functions return generator objects. A generator object is an instance of the generator function, and it conforms to both the iterable and iterator interfaces.

The generator objects can be used either by calling the next() method or by using the generator object within a loop. The generator object is an iterator; that's why you can use it in for…of loops or in other functions accepting an iterable.

In the above next() method example, the variable gen is the generator object.

Generator function with for…of loop

Using for…of loop with generator function reduces the line of code. You can see the illustration for the same in the following example.

Example

```
"use strict"
function* vowels() {
  // here the asterisk marks this as a generator
  yield 'A';
  yield 'E';
  yield 'I';
  yield 'O';
  yield 'U';
}
for(let alpha of vowels()) {
  console.log(alpha);
}
```

Output

```
A
E
I
O
U
```

2.12 ES6 PROMISES

GQ. Explain Promises in ES6. **(10 Marks)**

Promises are a way to implement async programming in JavaScript(ES6). A Promise will become a container for future value. Like if you order any food on any site to deliver it to your place that order record will be the promise and the food will be the value of that promise. So the order details are the container of the food you ordered. Let's explain it with another example. You order an awesome camera online. After your order is placed you receive a receipt of the order. That receipt is a Promise that your order will be delivered to you. The receipt is a placeholder for the future value namely the camera.

Promises used in JavaScript for asynchronous programming. For asynchronous programming, JavaScript used callbacks but there is a problem using the callback which is callback hell or Pyramid of Doom. Using the ES6 Promise will simply avoid all the problems associated with the callback.

2.12.1 Need of Promises

The Callbacks are great when dealing with basic cases. But in while developing a web application where you have a lot of code. Callbacks can be great trouble. In complex cases, every callback adds a level of nesting which can make your code really messy and hard to understand. This excessive nesting of callbacks is often termed as Callback Hell.

Example :

```
Callback Hell
f1(function(x){
    f2(x, function(y){
        f3(y, function(z){
            ...
        });
    });
});
```

To deal with this problem, we use Promises instead of callbacks.

2.12.2 Making Promises

GQ. What are the states of promises in ES6? **(5 Marks)**

A Promise is basically created when we are unsure of whether or not the assigned task will be completed. The Promise object represents the eventual completion (or failure) of an async(asynchronous) operation and its resulting value. As the name suggests a Promise is either kept or broken.

A Promise is always in one of the following states :

- **fulfilled :** Action related to the promise succeeded.
- **rejected :** Action related to the promise failed.
- **pending :** Promise is still pending i.e not fulfilled or rejected yet.
- **settled :** Promise has fulfilled or rejected.

Syntax

```
const promise = new Promise((resolve,reject) => {....});
```

Example

```
const myPromise = new Promise((resolve, reject) => {
    if (Math.random() > 0) {
    resolve('Hello, I am positive number!');
    }
    reject(new Error('I failed some times'));
})
```

2.12.3 Callbacks to Promises

There are two types of callbacks which are used for handling promises .then() and .catch(). It can be used for handling promises in case of fulfillment (promise is kept) or rejection (promise is broken).

.then(): Invoked when a promise is kept or broken. It can be chained to handle the fulfilment or rejection of a promise. It takes in two functions as parameters. The first one is invoked if the promise is fulfilled and the second one(optional) is invoked if the promise is rejected.

Example

```
Handling Promise rejection using .then()
var promise = new Promise(function(resolve, reject) {
    resolve('Hello, I am a Promise!');
})

promise.then(function(promise_kept_message) {
        console.log(promise_kept_message);
    }, function(error) {

        // This function is invoked this time
        // as the Promise is rejected.
        console.log(error); })
```

.catch() can be used for handling the errors(if any). It takes only one function as a parameter which is used to handle the errors (if any).

Example

```
Handling Promise rejection(or errors) using .catch()
const myPromise = new Promise((resolve, reject) => {
    if (Math.random() > 0) {
        console.log('resolving the promise ...');
        resolve('Hello, Positive :)');
    }
    reject(new Error('No place for Negative here :('));
});
```

```
const Fulfilled = (fulfilledValue) =>
console.log(fulfilledValue);
const Rejected = (error) => console.log(error);
myPromise.then(Fulfilled, Rejected);


myPromise.then((fulfilledValue) => {
    console.log(fulfilledValue);
}}).catch(err => console.log(err));
```

2.14 FETCH

GQ. Explain fetch method with the help of example. **(5 Marks)**

The fetch() method in JavaScript is used to request to the server and load the information in the webpages. The request can be of any APIs that returns the data of the format JSON or XML. This method returns a promise.

Syntax

fetch(url, options)

Parameters

This method accept two parameters as mentioned above and described below:

- **URL :** It is the URL to which the request is to be made.
- **Options :** It is an array of properties. It is an optional parameter.
- Return Value : It returns a promises whether it is resolved or not. The return data can be of the format JSON or XML.
- It can be the array of objects or simply a single object.

Example 1 : Making Get Request using Fetch

GQ. How to make Get Request using fetch method. **(5 Marks)**

Without options Fetch will always act as a get request.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content=
    "width=device-width, initial-scale=1.0">
    <title>JavaScript | fetch() Method</title>
</head>
<body>
    <script>

        // API for get requests
        let fetchRes = fetch(
"https://jsonplaceholder.typicode.com/todos/1");

        // fetchRes is the promise to resolve
        // it by using.then() method
        fetchRes.then(res =>
            res.json()).then(d => {
                console.log(d)
            })
    </script>
</body>
</html>
```

Example 2 : Making Post Request using Fetch

GQ. How to make Post Request using fetch method. **(5 Marks)**

Post requests can be made using fetch by giving options as given below:

```
let options = {
 method: 'POST',
 headers: {
  'Content-Type': 'application/json;charset=utf-8'
 },
 body: JSON.stringify(data)
}
```

Example

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content=
      "width=device-width, initial-scale=1.0">

    <title>JavaScript | fetch() Method</title>
</head>
<body>
    <script>
      user = {
          "name": "Geeks for Geeks",
          "age": "23"
      }
      // Options to be given as parameter in fetch for making
requests other then GET
      let options = {
```

```
    method: 'POST',
    headers: {
        'Content-Type':
        'application/json;charset=utf-8'
    },
    body: JSON.stringify(user)
}
// Fake api for making post requests
let fetchRes =
fetch("http://dummy.restapiexample.com/api/v1/create",
options);
fetchRes.then(res =>
    res.json()).then(d => {
        console.log(d)
    })
</script>
</body>
</html>
```

3.6 REACTJS - COMPONENT LIFE CYCLE

GQ. What are the different lifecycle methods in React? **(10 Marks)**

GQ. Explain React component life cycle. **(10 Marks)**

There are four phases in a component's lifecycle. They are:

1. Initial phase
2. Mounting phase
3. Updating phase
4. Unmounting phase

1. Initial Phase

This phase is considered the birth phase in a ReactJS component's lifecycle. In this, the component begins its way to the DOM. During this phase, the component includes both default Props and the initial State.

The initial phase includes the following methods:

- getDefaultProps(): It defines the default value of this.props.
- getInitialState(): It defines the default value of this.state.

2. Mounting Phase

In the mounting phase, you create the component instance and insert it into the DOM.

It has the following methods :

- **componentWillMount( ) :** You call this method before the rendering of a component into the DOM. The component will not re-render when you call setState() in this method.
- **componentDidMount( ) :** You call this method after rendering a component and placing it on the DOM. Then, you can perform DOM querying operations.
- **render( ) :** You define this method in every component. You cannot return a single root HTML node element. However,

you can return a null or false value if you don't want to render anything.

3. Updating Phase

In this phase, you can handle the user interaction and communication with the hierarchy of components. In this phase, the priority is to make sure that the component's latest version is displayed. Unlike the previous stages, the updating phase repeats over and over again. It includes the following methods:

- **componentWillReceiveProps() :** You call this method when a component receives a new prop.
- **shouldComponentUpdate() :** You call this method when a component decides it wants to make changes to the DOM. You can control the component's behavior when it gets updated.
- **componentWillUpdate() :** You call this method before the component update. You cannot use the this.setState() method to change the component state. Further, if shouldComponentUpdate() returns false, then this method will not be called.
- **render() :** You call this to inspect this.props and this.state. It returns any of the following: Arrays and fragments, String and Number, Booleans or null, React elements.
- **componentDidUpdate():** You immediately call this method when the component is updated.

4. Unmounting Phase

The unmounting phase is the last phase of the React component lifecycle. It is invoked in a situation when a component instance is unmounted and destroyed from the DOM. It contains only one method, as given below:

- **componentWillUnmount():** You call this component before a component is unmounted and eliminated permanently. This method performs cleanup tasks like event listening, canceling network requests, invalidating timers, cleaning up DOM elements, etc.

Example

Here is an example which shows the methods called at each state.

complife.jsx

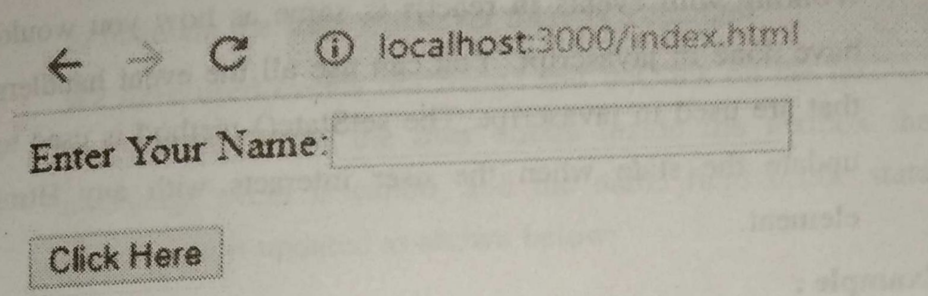
```
import React from 'react';
import ReactDOM from 'react-dom';
class COMP_LIFE extends React.Component {
constructor(props) {
super(props);
this.state = {name: ''};
this.UpdateName = this.UpdateName.bind(this);
this.testclick = this.testclick.bind(this);
 }
UpdateName(event) {
this.setState({name: event.target.value});
 }
testclick(event) {
alert("The name entered is: "+ this.state.name);
 }
componentDidMount() {
console.log('Mounting State : calling method
componentDidMount');
 }
shouldComponentUpdate() {
console.log('Update  State : calling method
shouldComponentUpdate');
return true;
 }
componentDidUpdate() {
console.log('Update  State : calling method
componentDidUpdate')
 }
componentWillUnmount() {
console.log('UnMounting State : calling method
componentWillUnmount');
 }
render() {
return (
<div>
      Enter Your Name:<input type="text"
value={this.state.name} onChange={this.UpdateName}
/><br/>
<h2>{this.state.name}</h2>
<input type="button" value="Click Here"
onClick={this.testclick} />
</div>
  );
 }
}

export default COMP_LIFE;
```

index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import COMP_LIFE from './complife.jsx';
ReactDOM.render(
<COMP_LIFE />,
document.getElementById('root')
);
```

When you check the output in the browser



(1C1)**Fig. 3.6.1**

In browser console you get:

```
Mounting State : calling method componentDidMount          complife.jsx:30
```

Oncethe user enters name in the textbox in console following messages are displayed:

```
Mounting State : calling method componentDidMount          complife.jsx:30
Update  State : calling method shouldComponentUpdate       complife.jsx:34
Update  State : calling method componentDidUpdate          complife.jsx:39
Update  State : calling method shouldComponentUpdate       complife.jsx:34
Update  State : calling method componentDidUpdate          complife.jsx:39
Update  State : calling method shouldComponentUpdate       complife.jsx:34
Update  State : calling method componentDidUpdate          complife.jsx:39
Update  State : calling method shouldComponentUpdate       complife.jsx:34
Update  State : calling method componentDidUpdate          complife.jsx:39
Update  State : calling method shouldComponentUpdate       complife.jsx:34
Update  State : calling method componentDidUpdate          complife.jsx:39
Update  State : calling method shouldComponentUpdate       complife.jsx:34
Update  State : calling method componentDidUpdate          complife.jsx:39
Update  State : calling method shouldComponentUpdate       complife.jsx:34
Update  State : calling method componentDidUpdate          complife.jsx:39
```

3.11 CREATING A SINGLE-PAGE APP IN REACT USING REACT ROUTER

SiA single page application (SPA) is essentially a webpage that interacts with the web browser dynamically by rewriting the current web page with the data obtained from the webserver. Hence, in a single page application, the webpage does not reload the page during its runtime and instead works within a browser. ngle-page applications are websites that have pages that load inline within the same page.

3.11.1 Single Page Application vs Multi-Page Application

1. Single-page Application

- **Reloading :** Single-page applications work inside a browser and do not require page reloading during webpage executing.
- **UI/UX:** Offers outstanding user experience by imitating a natural environment with the browser by eliminating wait time and page reloads. It consists of a single web page that loads all content using JavaScript. It requests the markup and data independently and renders pages straight to the browser.
- **Examples:** Gmail, Google Maps, Facebook, GitHub.

2. Multi-page Application

- **Reloading :** Multi-page applications work in the traditional way where every change, like displaying the data or submitting data back to the server, renders new pages from the server.
- **UI/UX :** Multi-page applications are larger applications with huge chunks of content, so the user experience is limited compared to single-page applications.
- **Examples :** eBay and Amazon

To build a single page app using React, follow the steps mentioned below:

1. Create a react app in your desired location using the following command:

   npx create-react-app app-name

2. Install react-router-dom for routing requests by executing the following command:

npm install react-router-dom

3. Wrap the App component.

There are two types of React routers, BrowserRouter (makes URLs like example.com/about) and HashRouter (makes URLs like example.com/#/about). We re using BrowserRouter in this example and use it to wrap the App component.

src/index.js file should include the following code:

```
import React from 'react'
import { render } from 'react-dom'
import { BrowserRouter } from 'react-router-dom'
import App from './App'
render(
<BrowserRouter>
<App />
</BrowserRouter>,
 document.querySelector('#root')
)
```

4. Create a file named src/pages/HomePage.js with the following code:

```
import React from "react";
export default function HomePage() {
return (
<>
<h1>HomePage</h1>
<p>This is HomePage subtitle</p>
</>
);
}
```

5. Create a file named src/pages/UserPage.js with the following code:

```
import React from "react";
import { useParams } from "react-router-dom";
export default function UserPage() {
let { id } = useParams();
return (
<>
<h1>Hello User {id}</h1>
<p>This is User Profile page</p>
</>
);
}
```

6. Decide and incorporate the routers that you want to use using Switch and Route. Switch groups all routes together and ensures that they take the precedence from top-to-bottom. Route, on the other hand, defines individual routes.

App.js file should include the decided routes.

```
import React from 'react'
import { Route, Switch } from 'react-router-dom'
// We will create these two pages in a moment
import HomePage from './pages/HomePage'
import UserPage from './pages/UserPage'
export default function App() {
 return (
<Switch>
<Route exact path="/" component={HomePage} />
<Route path="/:id" component={UserPage} />
</Switch>
 )
}
```

The above code matches the root route (/) to HomePage and matches other pages dynamically to UserPage.

7. Link to a page within the SPA using Link.

In the **src/pages/HomePage.js** file, include the following code:

```
import React from 'react'
import { Link } from 'react-router-dom'
export default function HomePage() {
 return (
<div className="container">
<h1>Home </h1>
<p>
<Link to="/your desired link">Your desired link.</Link>
</p>
<
 )
}
```

You can now run the code and view the development server available at http://localhost:3000.

3.8 REACTJS EVENTS

GQ. How do you create an event in React? **(10 Marks)**

- Events are actions that are triggered by either system generated events or user actions. Few examples of events are loading a web page, resizing a window, pressing keys, etc.
- React's event handling system is called synthetic events and is similar to the system on DOM elements.
- Working with events in reactjs is same as how you would have done in javascript. You can use all the event handlers that are used in javascript. The setState() method is used to update the state when the user interacts with any Html element.

Example :

events.jsx

```
import React from 'react';
import ReactDOM from 'react-dom';
class EventTest extends React.Component {
 constructor(props) {
  super(props);
  this.state = {name: ''};

  this.UpdateName = this.UpdateName.bind(this);
  this.testclick = this.testclick.bind(this);
 }
 UpdateName(event) {
  this.setState({name: event.target.value});
 }

 testclick(event) {
  alert("The name entered is: "+ this.state.name);
 }
 render() {
  return (
<div>
      Enter Your Name:<input type="text"
value={this.state.name} onChange={this.UpdateName}
/><br/>
<h2>{this.state.name}</h2>
```



```
<input type="button" value="Click Here"
onClick={this.testclick} />
</div>
  );
 }
}
export default EventTest;
```

- For the input fields, we need to maintain the state, so for that react provides a special method called setState, which helps to maintain the state whenever there is a change.
- We have used events onChange and onClick on the textbox and button. When the user enters inside the textbox the onChange event is called, and the name field inside state object state is updated as shown below:

```
UpdateName(event) {
  this.setState({name: event.target.value});
 }
```

index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import EventTest from './events.jsx';

ReactDOM.render(
<EventTest />,
  document.getElementById('root')
);
```

3.9 WORKING WITH CSS IN REACTJS

GQ. What are the different ways to style a React component ? **(10 Marks)**

3.9.1 Inline CSS in ReactJS

Example to understand the working of inline css in reactjs.

addstyle.jsx

```
import React from 'react';
import ReactDOM from 'react-dom';
const h1Style = {
  color: 'red'
};
```

```
return(
<div>This is main component.</div>
);
  }
}
```

Example : Class as Component

Here is a ReactJS example that uses a class as a component.

test.jsx

```
import React from 'react';
import ReactDOM from 'react-dom';

class Hello extends React. Component {
render() {
return<h1>Hello, Welcome to ReactJs!</h1>;
  }
 }
 export default Hello;
```

We can use Hello component in index.js file as follows:

index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import Hello from './test.jsx';
ReactDOM.render(
<Hello />,
document.getElementById('root')
);
```

The Component Hello is used as an Html tag i.e., <Hello />.

3.4 REACTJS STATE

GQ. Explain React state. **(5 Marks)**

- In ReactJS, the state contains information about the components and can change over time. It can either be a response to user action or a system event. Data collected in a state is a private object.
- Components within the state are called stateful components. They regulate the component's behavior and make it more interactive.
- You can access or modify the state only inside the component or directly by the component. You should always simplify your state as much as possible and reduce the number of stateful components. For example, if you have ten components requiring data from the state, you should create a single container component that will keep the state for all the components.

Example of State

Here is a working example on how to use state inside a class.

test.jsx

```
import React from 'react';
import ReactDOM from 'react-dom';
class Hello extends React.Component {
constructor(props) {
super(props);
this.state = {
msg: "Hello, Welcome to ReactJs!"
   }
  }
render() {
return<h1>{this.state.msg}</h1>;
  }
}
export default Hello;
```

index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import Hello from './test.jsx';

ReactDOM.render(
<Hello />,
document.getElementById('root')
);
```

3.5 REACTJS PROPS

GQ. What are props in React? **(5 Marks)**

Props are nothing but properties you can use inside a component. They are read-only variables that store the value of attributes of a tag. Props are similar to function arguments because

you can pass them to the components in a similar way as arguments.

Example : Props to Function Component

Here is an example of passing props to a function component.

```
import React from 'react';
import ReactDOM from 'react-dom';
function Hello(props) {
return<h1>{props.msg}</h1>;
}
constHello_comp = <Hello msg="Hello, Welcome to ReactJs!" />;
export default Hello_comp;
```

As shown above, we have added msg attribute to <Hello /> Component. The same can be accessed as props inside Hello function, which is an object that will have the msg attribute details, and the same is used as an expression.

The component is used in index.js as follows:

index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import Hello_comp from './test.jsx';

ReactDOM.render(
Hello_comp,
document.getElementById('root')
);
```

Example : Props to Class Component

To access the props in a class we can do it as follows:

test.jsx

```
import React from 'react';
import ReactDOM from 'react-dom';

class Hello extends React.Component {
render() {
return<h1>{this.props.msg}</h1>;
  }
}
export default Hello;
```

The msg attribute is passed to the component in index.js as follows :

```
import React from 'react';
import ReactDOM from 'react-dom';
import Hello from './test.jsx';
ReactDOM.render(
<Hello msg="Hello, Welcome to ReactJs!" />,
document.getElementById('root')
);
```

3.6 REACTJS - COMPONENT LIFE CYCLE

GQ. What are the different lifecycle methods in React? **(10 Marks)**

GQ. Explain React component life cycle. **(10 Marks)**

Module 3

There are four phases in a component's lifecycle. They are:

1. Initial phase
2. Mounting phase
3. Updating phase
4. Unmounting phase

1. Initial Phase

This phase is considered the birth phase in a ReactJS component's lifecycle. In this, the component begins its way to the DOM. During this phase, the component includes both default Props and the initial State.

The initial phase includes the following methods:

- getDefaultProps(): It defines the default value of this.props.
- getInitialState(): It defines the default value of this.state.

2. Mounting Phase

In the mounting phase, you create the component instance and insert it into the DOM.

It has the following methods :

- **componentWillMount( ) :** You call this method before the rendering of a component into the DOM. The component will not re-render when you call setState() in this method.
- **componentDidMount( ) :** You call this method after rendering a component and placing it on the DOM. Then, you can perform DOM querying operations.
- **render( ) :** You define this method in every component. You cannot return a single root HTML node element. However,

⏭ 1.9 DOM (DOCUMENT OBJECT MODEL)

GQ. What is DOM? Explain different types of DOM.

(10 Marks)

The Document Object Model (DOM) is a programming interface for HTML and XML (Extensible Markup Language) documents. It defines the logical structure of documents and the way a document is accessed and manipulated. The Document Object Model (DOM) is a W3C standard. It defines a standard for accessing documents like HTML and XML.

DOM is a way to represent the webpage in a structured hierarchical way so that it will become easier for programmers and users to glide through the document. With DOM, we can easily access and manipulate tags, IDs, classes, Attributes, or Elements using commands or methods provided by the Document object.

DOM defines the objects and properties and methods (interface) to access all XML elements. It is separated into 3 different parts / levels :

- **Core DOM :** standard model for any structured document
- **XML DOM :** standard model for XML documents
- **HTML DOM :** standard model for HTML documents

1.9.1 XML DOM

- XML DOM is a standard object model for XML. XML documents have a hierarchy of informational units called nodes; DOM is a standard programming interface of describing those nodes and the relationships between them.
- As XML DOM also provides an API that allows a developer to add, edit, move or remove nodes at any point on the tree in order to create an application.

Advantages of XML DOM

The following are the advantages of XML DOM.

(1) XML DOM is language and platform independent.

(2) XML DOM is traversable - Information in XML DOM is organized in a hierarchy which allows developer to navigate around the hierarchy looking for specific information.

(3) XML DOM is modifiable - It is dynamic in nature providing the developer a scope to add, edit, move or remove nodes at any point on the tree.

Disadvantages of XML DOM

(1) It consumes more memory (if the XML structure is large) as program written once remains in memory all the time until and unless removed explicitly.

(2) Due to the extensive usage of memory, its operational speed, compared to SAX is slower.

1.9.2 HTML DOM

The HTML DOM is a standard object model and programming interface for HTML.

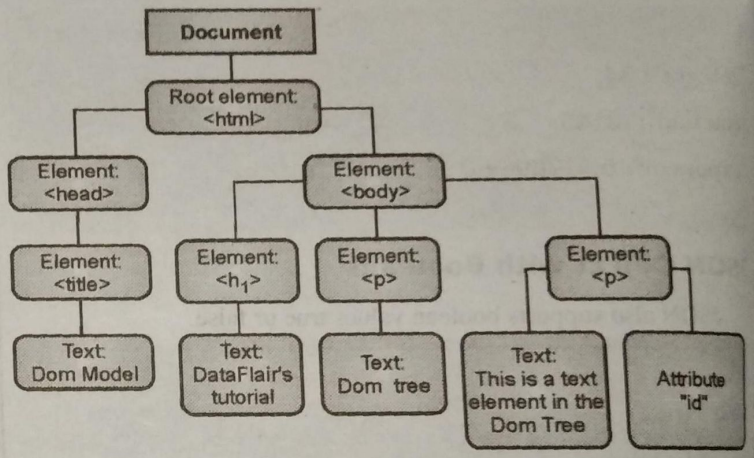
It defines :

- The HTML elements as objects
- The properties of all HTML elements
- The methods to access all HTML elements
- The events for all HTML elements

Properties of document object

The properties of the document object that can be accessed and modified by the document object.

- **Window Object :** Window Object is always at top of the hierarchy.
- **Document object :** When an HTML document is loaded into a window, it becomes a document object.
- **Form Object :** It is represented by form tags.
- **Link Objects :** It is represented by link tags.
- **Anchor Objects :** It is represented by a href tags.
- **Form Control Elements :** Form can have many control elements such as text fields, buttons, radio buttons, and checkboxes, etc.



(1A11)**Fig. 1.9.1 : Document Object Model**

Methods of Document Object

- write("string"): writes the given string on the document.
- getElementById(): returns the element having the given id value.
- getElementsByName(): returns all the elements having the given name value.
- getElementsByTagName(): returns all the elements having the given tag name.
- getElementsByClassName(): returns all the elements having the given class name.

1.9.3 Levels of DOM: Level 0: Provides a Low-level Set of Interfaces

Level 1 : DOM level 1 can be described in two parts: CORE and HTML.

- CORE provides low-level interfaces that can be used to represent any structured document.

HTML provides high-level interfaces that can be used to represent HTML documents.

evel 2 : Consists of six specifications: CORE2, VIEWS, VENTS, STYLE, TRAVERSAL, and RANGE.

CORE2: extends the functionality of CORE specified by DOM level 1.

- VIEWS: views allow programs to dynamically access and manipulate the content of the document.
- EVENTS: Events are scripts that are either executed by the browser when the user reacts to the web page.
- STYLE: allows programs to dynamically access and manipulate the content of style sheets.
- TRAVERSAL: allows programs to dynamically traverse the document.
- RANGE: allows programs to dynamically identify a range of content in the document.

Level 3: Consists of five different specifications: CORE3, LOAD and SAVE, VALIDATION, EVENTS, and XPATH.

- CORE3: extends the functionality of CORE specified by DOM level 2.
- LOAD and SAVE: allows the program to dynamically load the content of the XML document into the DOM document and save the DOM Document into an XML document by serialization.
- VALIDATION: allows the program to dynamically update the content and structure of the document while ensuring the document remains valid.
- EVENTS: extends the functionality of Events specified by DOM Level 2.
- XPATH: XPATH is a path language that can be used to access the DOM tree.

Example of DOM manipulation

```
<!DOCTYPE html>
<html>
<head>
  <title>DOM manipulation</title>
</head>
<body>
Enter Value 1 <input type="text" id="val1">
  <br>
  <br>
  Enter Value 2<input type=".text" id="val2">
  <br>
  <br>
  <button onclick="getAdd()">Click To Add</button>
  <br>
  <p id="result"></p>
  <!--Javascript code inside body tag -->
  <script type="text/javascript">
  functiongetAdd()
{
//it will fetch the value of input with id val1
const
num1=Number(document.getElementById('val1').value);
//It will fetch the value of input with id val2
const
num2=Number(document.getElementById('val2').value);
//addition
const add=num1+num2;
console.log(add);
//displaying the result in paragraph using dom
document.getElementById('result').innerHTML="Addition :
"+add;
//it will change the color of paragraph with red
document.getElementById('result').style.color='red';
}
  </script>
</body>
</html>
```

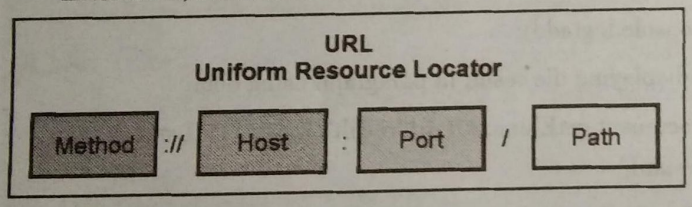
1.10 URL (UNIFORM RESOURCE LOCATOR)

GQ. Explain the term URL in detail. **(10 Marks)**

- A client that wants to access the document in an internet needs an address and to facilitate the access of documents. HTTP uses the concept of Uniform Resource Locator (URL).

The Uniform Resource Locator is a standard way of specifying any kind of information on the internet.

- A URL is located in the address bar or search bar at the top of the browser window. The URL is always visible in the desktop computers and laptop unless your browser is being displayed in full screen. In most of the smartphones and tablets, when you scroll down the page, the URL will disappear and only show the domain when visible. To visible the address bar, you need to scroll up the page. And, if only the domain is shown and you want to see full address, tapping on the address bar to show the full address.
- A URL has two main components:
  - **Protocol identifier :** For the URL http://example.com, the protocol identifier is http.
  - **Resource name :** For the URL http://example.com, the resource name is example.com.
- The URL sends users to a specific resource online such as video, webpage, or other resources. When you search any query on Google, it will display the multiple URLs of the resource that are all related to your search query. The displayed URLs are the hyperlink to access the webpages.
- A URL (Uniform Resource Locator) contains the information, which is as follows:



(1A12)**Fig.1.10.1 : Uniform Resource Locator**

- **Method :** The method is the protocol used to retrieve the document from a server. For example, HTTP.
- **Host :** The host is the computer where the information is stored, and the computer is given an alias name. Web pages are mainly stored in the computers and the computers are given an alias name that begins with the characters "www". This field is not mandatory.
- **Port :** The URL can also contain the port number of the server, but it's an optional field. If the port number is included, then it must come between the host and path and it should be separated from the host by a colon.
- **Path :** Path is the pathname of the file where the information is stored. The path itself contain slashes that separate the directories from the subdirectories and files.

Example

https://www.loophole.com/jtp.htm, it indicates the jtp.htm is a file located on the server with the address of loophole.com

http:// or https://

The http is a protocol that stands for Hypertext Transfer Protocol. It tells the browser to which protocol will be preferred to use for accessing the information that is specified in the domain. The https (Hypertext Transfer Protocol Secure) is an enhanced protocol as compared to http as it concerned with security. It provides the surety that the information, which is transmitted over HTTP is secure and encrypted. The colon (:) and two forward slashes (//) are used to separate the protocol from the rest of the part of the URL.

www

The www is used to distinguish the content, which stands for World Wide Web. This portion of the URL can be left out many times, as it is not required. For instance, if you type "http://javatpoint.com," you will still get the javatpoint website. For an important subpage, this portion can also be substituted, which is known as a subdomain.

loopholetech.com

The loopholetech.com is the domain name for the website, and the .com is called TLD or suffix. It helps to identify the location or type of the website. For example, ".org" stands for an organization, ".co.uk" stands for the United Kingdom, and ".com" is for commercial. There are various types of domain suffixes available; you are required to register the name through a domain registrar to get a domain.

jtp.htm

The jtp.htm is the name of the web page, and the .htm is the file extension of the web page, which describes the file is an HTML file. There are many other file extensions available on the internet such as .php, .html, .xml, .jpg, .gif, .asp, .cgi, etc.

1.11 URI

GQ. Explain the term URI in detail. **(10 Marks)**

- A URI or Uniform Resource Identifier is a string identifier that refers to a resource on the internet. It is a string of characters that is used to identify any resource on the internet using location, name, or both.
- A URI has two subsets; URL (Uniform Resource Locator) and URN (Uniform Resource Number). If it contains only a name, it means it is not a URL. Instead of directly URI, we mostly see the URL and URN in the real world.

- Headers
- Cookies

HTTPS also uses the SSL/TLS protocol for authentication. SSL/TLS uses digital documents known as X.509 certificates to bind cryptographic key pairs to the identities of entities such as websites, individuals, and companies. Each key pair includes a private key, which is kept secure, and a public key, which can be widely distributed. Anyone with the public key can use it to:

- Send a message that only the possessor of the private key can decrypt.
- Confirm that a message has been digitally signed by its corresponding private key.

If the certificate presented by an HTTPS website has been signed by a publicly trusted certificate authority (CA), such as SSL.com, users can be assured that the identity of the website has been validated by a trusted and rigorously-audited third party.

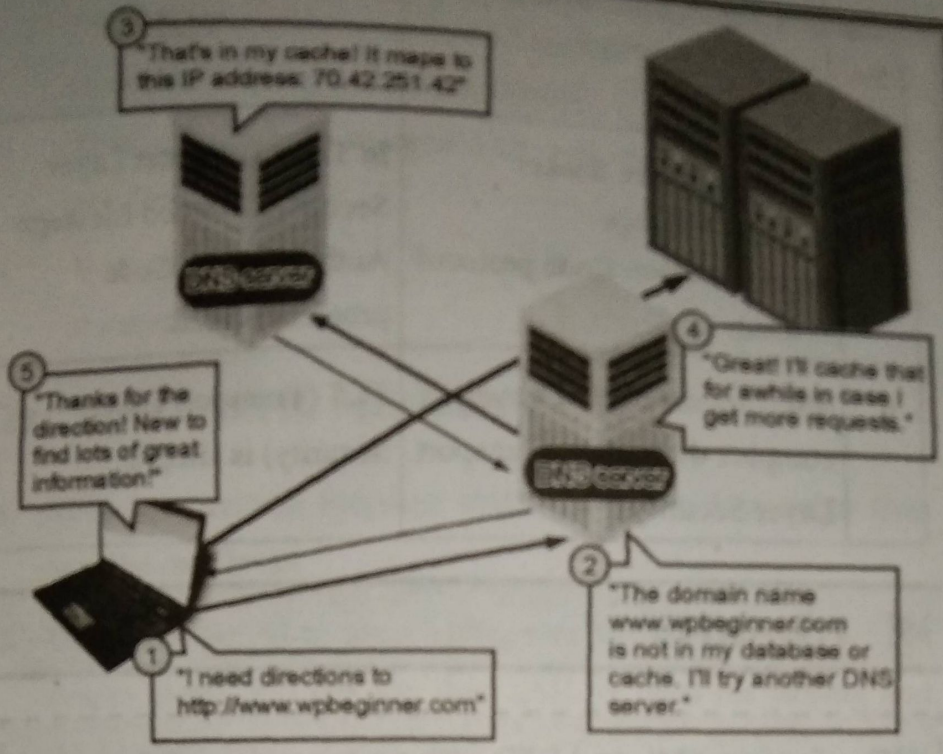
1.5 DNS (DOMAIN NAME SYSTEM)

GQ. What is DNS? Explain working of DNS. **(10 Marks)**

- Domain name is the address of your website that people type in the browser URL bar to visit your website.
- In simple terms, if your website was a house, then your domain name will be its address.
- The Internet is a giant network of computers connected to each other through a global network of cables.
- Each computer on this network can communicate with other computers. To identify them, each computer is assigned an IP address.
- It is a series of numbers that identify a particular computer on the internet. A typical IP address looks like this: 66.249.66.1. Now an IP address like this is quite difficult to remember. Imagine if you had to use such numbers to visit your favourite websites.
- Domain names were invented to solve this problem. Now if you want to visit a website, then you don't need to enter a long string of numbers. Instead, you can visit it by typing an easy to remember domain name in your browser's address bar. For example, wpbeginner.com.
- DNS stands for "Domain Name System". It's a system that lets you connect to websites by matching human-readable domain names (like wpbeginner.com) with the unique ID of the server where a website is stored. An application layer protocol defines how the application processes running on different systems, pass the messages to each other.
- The Domain Name System is the phonebook of the Internet. Humans access information online through domain names, like nytimes.com or espn.com. Web browsers interact through Internet Protocol (IP) addresses. DNS translates domain names to IP addresses so browsers can load Internet resources.
- DNS is a directory service that provides a mapping between the name of a host on the network and its numerical address. It is required for the functioning of the internet. Each node in a tree has a domain name, and a full domain name is a sequence of symbols specified by dots. DNS is a service that translates the domain name into IP addresses. This allows the users of networks to utilize user-friendly names when looking for other hosts instead of remembering the IP addresses.
- Each device connected to the Internet has a unique IP address which other machines use to find the device. DNS servers eliminate the need for humans to memorize IP addresses such as 192.168.1.1 (in IPv4), or more complex newer alphanumeric IP addresses such as 2400:cb00:2048:1::c629:d7a2 (in IPv6).

1.5.1 How Does DNS Work?

- The internet is a giant network of computers. Each device connected to the internet is assigned a unique IP address which helps other computers identify it. This IP address is a string of numbers with periods that looks like this: 192.124.249.166
- Now imagine if you had to remember such long strings of numbers to visit your favourite websites. They are hard to remember and don't tell you anything about the website you'll see if you enter them in a browser.
- Domain names were invented to solve this problem by using alphabets and allowing users to select easy to remember names for their websites. DNS or Domain Name System basically translates those domain names into IP addresses and points your device in the right direction. A domain name and its matching IP address is called a "DNS record".
- Here is a simple way to understand how DNS works in four steps.



(1A9)**Fig. 1.5.1 : Working of DNS**

Suppose you want to visit our site at www.wpbeginner.com.

1. You open your browser and type www.wpbeginner.com in the address bar and hit Enter on the keyboard. Immediately there is a quick check to see if you have visited our website previously. If the DNS records are found in your computer's DNS cache, then the rest of the DNS lookup is skipped and you will be taken directly to www.wpbeginner.com.
2. If no DNS records are found, then a query is sent to your local DNS server. Typically, this is your Internet provider's server and is often called a "resolving nameserver".
3. If the records are not cached on the resolving nameserver, then the request is forwarded to what's called a "root nameserver" to locate the DNS records. Root nameservers are designated servers around the world that are responsible for storing DNS data and keeping the system working smoothly. Once the DNS record is found on the root nameserver, it's cached by your computer.
4. Now that the DNS records are located, a connection to the server where the website is stored will be opened and www.wpbeginner.com will be displayed on your screen.

Let's understand the differences in a tabular form.

Sr. No.	HTTP	HTTPS
1.	The full form of HTTP is the Hypertext Transfer Protocol.	The full form of HTTPS is Hypertext Transfer Protocol Secure.
2.	It is written in the address bar as http://.	It is written in the address bar as https://.
3.	The HTTP transmits the data over port number 80.	The HTTPS transmits the data over port number 443.
4.	It is unsecured as the plain text is sent, which can be accessible by the hackers.	It is secure as it sends the encrypted data which hackers cannot understand.
5.	It is mainly used for those websites that provide information like blog writing.	It is a secure protocol, so it is used for those websites that require to transmit the bank account details or credit card numbers.
6.	It is an application layer protocol.	It is a transport layer protocol.
7.	It does not use SSL.	It uses SSL that provides the encryption of the data.

Sr. No.	HTTP	HTTPS
8.	Google does not give the preference to the HTTP websites.	Google gives preferences to the HTTPS as HTTPS websites are secure websites.
9.	The page loading speed is fast.	The page loading speed is slow as compared to HTTP because of the additional feature that it supports, i.e., security.