Class 17 - Creating a real time weather app

Agenda

1. What is JSON? and How to work with JSON data
2. Create a weather app using an API.
3. We will be using the "weather API" for getting the data of real-time weather and use that data in our app.

title: Introduction to JSON

1. JSON, which stands for JavaScript Object Notation, is a simple format for storing and sharing information over internet. It's designed to be easy for us to read and write, and at the same time, simple for computers to understand and process.
2. Although JSON originates from JavaScript, it has become a universal language that almost any programming environment can understand, from Python to PHP, making it incredibly versatile.
3. JSON is lightweight, it doesn't add significant overhead, which is crucial for fast data transmission over the internet.

Structure of JSON

1. JSON's format is reminiscent of JavaScript object literal syntax, but with specific restrictions to ensure its data is easily

exchanged across diverse programming environments. It mainly consists of two structures:

2. Objects
    a. Objects in JSON are collections of key/value pairs, similar to dictionaries or maps in other programming languages.
    b. An object starts with a left brace { and ends with a right brace }.
    c. Inside the object, keys (always in double quotes) and their corresponding values are separated by a colon :, and each key-value pair is separated by a comma ,. For example:

```
{
 "name": "John Doe",
 "age": 30,
 "isEmployed": true
}
```

3. Arrays
    a. Arrays are ordered lists of values, starting with a left bracket [and ending with a right bracket]. The values within an array are separated by commas, and they can be of any JSON-supported data type. For example:

```
["apple", "banana", "cherry"]
```

4. Supported data types for values
    a. The values in JSON can be:
    b. Strings (text wrapped in double quotes " "),
    c. Numbers (integers or floats),

d. Objects (another JSON object),
e. Arrays,
f. true or false (boolean values),
g. null (to represent a null value).

How JSON Works with JavaScript

1. A primary use of the JSON format is for transferring data between different systems.
2. This is particularly effective because many of the internet's data transmission protocols, such as HTTP, are designed to handle text-based data.
3. JSON, which is inherently a text format, aligns perfectly with these protocols.
4. JSON's text-based nature allows it to be universally understood and processed across various platforms
5. So Now as JSON is text(string) based so to transfer data from one system another we need to convert data that we need to transfer into JSON string
6. Think of JSON as the perfect packaging material for your data. It wraps up your information in a neat, compact format — much like packing your items into a box and sealing it up. This box (or JSON string) is designed to be lightweight, making it ideal for travel across the vast expanse of the internet.
7. Once the package arrives at its destination, your friend (or another computer system) can easily open it up and take out the

items (data). They'll find everything intact and organized, just as you sent it. JSON, in essence, acts as the universal packaging method, ensuring your data can be sent, received, and understood across different platforms and programming languages.

8. Let's create a simple example to illustrate both parsing and stringifying with toys as our data. First, we'll "pack" our toy data into a JSON string using JSON.stringify(), and then we'll "unpack" it using JSON.parse().
9. code

```
// Define our toys as a JavaScript object
var toys = {
 toy1: { name: "Teddy Bear", color: "Brown" },
 toy2: { name: "Race Car", color: "Red" },
 toy3: { name: "Doll", color: "Pink" },
};
// Now Here this is in  Javascript object Format and in
this format data cannot be sent over the internet as this
is not universally understood by the data transfer
protocols

// Pack our toys into a box (convert our toys object into a
JSON string)
var toysJSONString = JSON.stringify(toys);
console.log("Packed Toys:", toysJSONString);

// Here We have converted the javascript object into JSON
string now this is understood by the protocolos
```

```
// Now, let's say this JSON string is sent over the
internet.
// On the other side, we receive this packed box of toys as
a JSON string.

// Now as we are recieving a JSON string , so while writing
javascript code we will not be able to apply object
properties in a JSON string , so now to convert the string
again to object we will parse it

// Unpack our toys from the box (parse the JSON string back
into a JavaScript object)
var unpackedToys = JSON.parse(toysJSONString);
console.log("Unpacked Toys:", unpackedToys);

// Now, 'unpackedToys' is a JavaScript object with our
toys,
// and we can access and work with them like before.
console.log("Let's play with the:",
unpackedToys.toy1.name); // Accessing the Teddy Bear
```

10. In this example:
    a. We start with our toys organized in a JavaScript object.
    b. We "pack" these toys using JSON.stringify(), turning them into a string that can be easily sent over the internet.
    c. We simulate receiving this packed string on the other side and then "unpack" it using JSON.parse(), turning it back into a JavaScript object that we can work with.

    d. Finally, we access one of the toys from our unpacked object to show that it's back to its original, usable form.

Summary: Why Use JSON?

1. JSON is popular for several reasons:
2. It is text, and can be read and written by humans.
3. It is easy for machines to parse and generate.
4. It is fully compatible with JavaScript and many other languages, making it a good choice for data interchange on the web.
5. In summary, JSON is a universally accepted format for structuring data, making it an essential part of modern web development. Its simplicity, ease of use, and compatibility with many programming languages have made it a preferred choice for data interchange on the internet.

Introduction to API

1. API and the Use of JSON Data
    a. APIs play a pivotal role in enabling different software systems to communicate with each other.
    b. They act as intermediaries that allow applications to exchange data and functionality easily and securely.
    c. JSON, with its simplicity and efficiency, is widely used as the format for exchanging data in API communications.

2. An API is a set of rules and protocols that allows one software application to access the data or functionality of another application, server, or service.
3. APIs are like menus in a restaurant; the menu provides a list of dishes you can order, along with a description of each dish.
4. When you specify which dish you want, the kitchen (the system) prepares the dish and serves it. In the digital world, APIs work similarly: they allow developers to request specific data or actions, and the system responds accordingly.

The Role of JSON in API

1. JSON is the medium through which data is exchanged between clients and servers in many APIs. It serves as the "language" that both the requesting and the responding parties understand. Here's how JSON is used in the context of APIs:
2. Requesting Data: When an application requests data from a server, it sends an API call, which is essentially a request made over the internet. This request often includes data formatted as a JSON string. For example, an application might request the details of a user by sending a JSON string that specifies the user ID.
3. Responding to Requests: Upon receiving the request, the server processes it and sends back a response. This response is typically also in JSON format, containing the data requested or

confirmation of the action taken. For instance, a server might respond with a JSON object that includes the user's details.

4. Ease of Integration: The reason JSON is so popular in API development is its ease of integration with most programming languages. Libraries in Python, JavaScript, Ruby, and many others can easily parse JSON strings into native data structures of the respective language (like dictionaries in Python or objects in JavaScript) and vice versa.
5. API Endpoints: APIs are structured around **endpoints**, each serving a specific function. An endpoint for retrieving user data might return a JSON object with user attributes, while another endpoint for posting user data might accept JSON-formatted user details.

Weather app

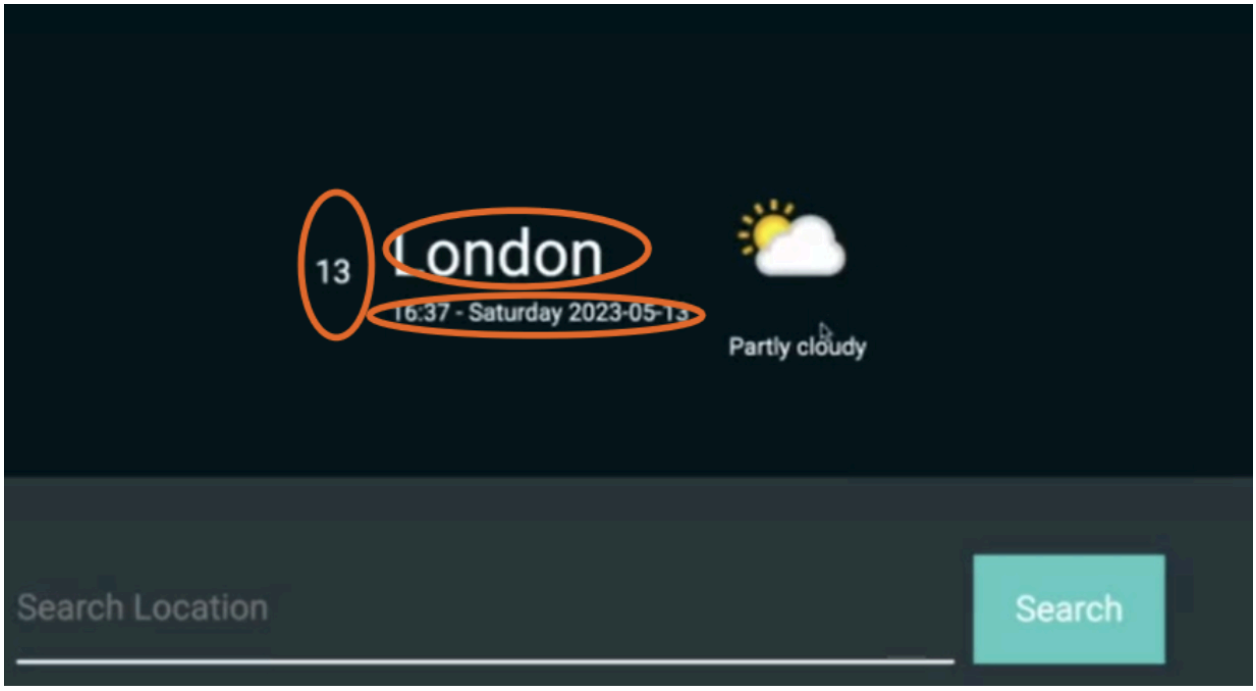
1. Functionality expected
    a. It will have a search bar where you can enter any cities name an their will be a button to get it's weather Info.
    b. As soon as you click on the button you should be served with that cities weather Information in a decent looking card
    c. These properties should be shown compulsorily-
        i. City Name
        ii. Temperature
        iii. Condition (hot , humid , cold , Rainy etc)

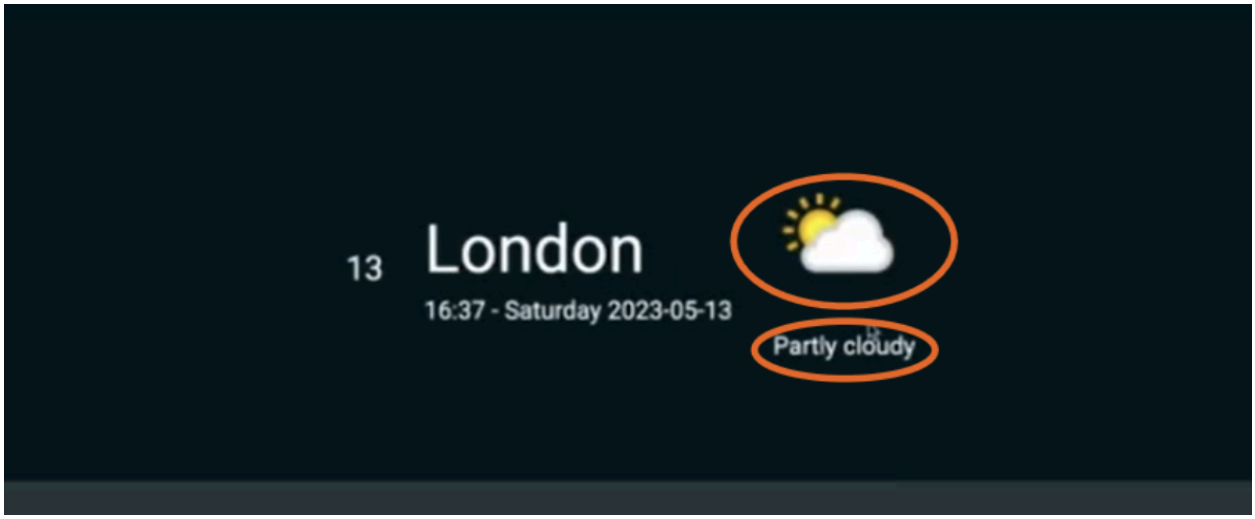
    iv. current date and time in the city
    v. Rest you can add any other property that you want to show!
2. So , This is a very simple app , we just need data for this , and how we are going to get the data?
    a. Using an API!
3. Steps to get the data from an API
    a. The application sends a request to the weather API's endpoint, sending the location name
    b. The weather API processes this request and responds with a JSON object containing weather details like temperature, humidity, and weather conditions
    c. The application then parses this JSON response and uses the data to display the weather information to the use

Creating Weather app

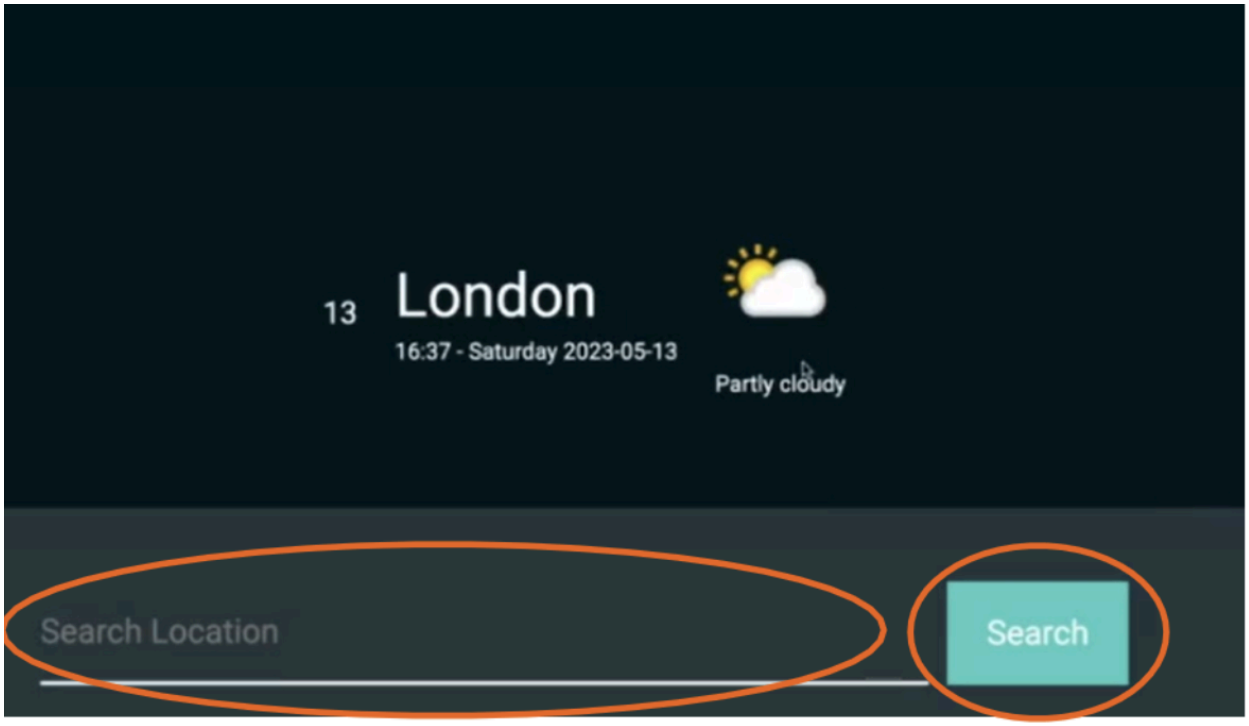
1. To start creating this app we need access to API from where we are going to get the data
2. For our app we will be using the API from here -
    a. https://www.weatherapi.com/
3. Find the API key and paste it under the API explorer tab.
4. Select the protocol as HTTPS.
5. Click on show response. You can see the JSON format in the response section.

Initial files

1. Create index.html, style.css and index.js files in VSCode.
2. Inside the index.html file, we will be creating various divisions for various parameters like temperature, location, time and date etc.
3. 

4. In the other division for weather condition, we will display an emoji along with the weather condition.
5. 

6. Create a form which contains an input field and a button also.

7. 

8. Starter html

```
<!DOCTYPE html>
<html lang="en">
 <head>
   <meta charset="UTF - 8" />
   <meta http-equiv="X - UA - Compatible" content="IE = edge" />
   <meta name="viewport" content="width = device-width, initial-scale = 1.0" />
   <title>Weather app Class</title>

   <link rel="stylesheet" href="style.css" />
 </head>
 <body>
   <div class="container">
     <div class="weather">
       <div class="temp">20</div>
       <div class="time_location">
         <p>Location</p>
         <span>Random time and Date</span>
       </div>

       <div class="weather_condition">
         <p><img src="" alt="" /></p>
         <span>Condition</span>
```

```
      </div>
    </div>
  </div>

  <div class=”nav”>
    <form>
      <input type="text" placeholder="Search_location" class="searchField" />
      <button type="submit">Search</button>
    </form>
  </div>
 </body>

 <script src="index.js"></script>
</html>
```

9. Styling code

```
@import
url("https://fonts.googleapis.com/css2?family=Economica&family=Grape+Nuts&famil
y=Roboto:wght@100;300;400;700;900&display=swap");

* {
 margin: 0%;
 padding: 0;
 font-family: "Roboto", sans-serif;
}

.container {
 width: 100%;
 height: 100vh;
 background-color: #01161e;
 display: flex;
 justify-content: center;
 align-items: center;
}

.weather {
 z-index: 2;
 display: flex;
```

```
 align-items: center;
 color: white;
}

.weather > div {
 margin: 0.625rem;
}

.weather1 {
 font-size: 4rem;
}

.weather p {
 font-size: 2rem;
}
.weather span {
 font-size: 0.75rem;
}

.weather3 span {
 margin: 0.3rem;
}

.weather3 img {
 width: 2rem;
}
.nav {
 height: 100px;
 padding: 1rem 0;
 position: absolute;
 bottom: 0;
 width: 100%;
 background-color: rgba(180, 177, 177, 0.202);
 display: flex;
 justify-content: center;
 align-items: center;
}

.nav form {
 width: 70%;
 grid-template-columns: 5fr 1fr;
 display: grid;
```

```
}

.searchField {
 font-size: 1.1rem;
 outline: none;
 color: white;
 background-color: transparent;
 padding: 1rem 0;
 border: none;
 border-bottom: 2px solid white;
 width: 98%;
}

.nav form button {
 background-color: #4ecdc4;
 font-size: 1.1rem;
 color: white;
 outline: none;
 border: none;
 cursor: pointer;
}
.time_location{
 display: flex;
 flex-direction: column;
 justify-content: center;
 align-items: center;
}
```

10. index.js - javascript
    a. We will see how to get data from an API. We will use the fetch API. The fetch() method will be used here.
        i. Type fetch method in the browser and just show the docs as a glimpse.

b. The API URL will be passed to the fetch() method as a parameter. We will understand how to use the fetch() method inside our JS code.
c. const url = https://api.weatherapi.com/v1/current.json?key=5d9564dfb4cc4b00bbe104259242003&q=Patna&aqi=yes
    i. Change it to accept any value
    ii. const url = `https://api.weatherapi.com/v1/current.json?key=35af7ff606db422880d141328231305&q=${target}&aqi=yes`
d. Here we have wrapped the url in backticks so that we can use the string template literals which denoted bt the ${} symbol.
e. Now as API in an external data source so the program may take some time to get the data from there , and whenever there are some heavy time consuming operation what do we use?
    i. Asynchronous Programming!
    ii. Here we will just use an Async Function which will be responsible for getting data from the API and to get the data from an API we use the fetch method
f. Key points
    i. In APIs, we will be using the asynchronous JS.
    ii. We will be using async and await keywords which are important to work asychronously

iii. The async keyword needs to be added before the function which tells the function that it is asynchronous.

iv. The await keyword tells that it needs wait till the data comes and the promise resolves

v. const response = await fetch(url)

g. Lets run the code till now

```
const target = "Pune";
async function fetchData(target) {
 try {
   const url =
`https://api.weatherapi.com/v1/current.json?key=5d9564
dfb4cc4b00bbe104259242003&q=${target}&aqi=yes`;
   const response = await fetch(url);
   console.log(response);
 } catch (error) {
   console.log(error);
 }
}

fetchData(target);
```

h. Here we are fetching a JSON file across the network. The simplest use of fetch() takes one argument — the path to the resource you want to fetch — and does not directly return the JSON response body but instead returns a promise.

i. The promise on settling returns a **Response** object (stream).

j. So, to extract the JSON body content from the Response object, we use the **json()** method, which returns a second promise that resolves with the result of parsing the response body text as object.

k. Update code

```
try {
   const url =
`https://api.weatherapi.com/v1/current.json?key=5d9564
dfb4cc4b00bbe104259242003&q=${target}&aqi=yes`;
   const response = await fetch(url);
   const data = await response.json()
   console.log(data)
 } catch (error) {
   console.log(error);
 }
```

l. Now we will extract dynamic values from the object.

   i. Look into the response format in browser, to obtain temperature, we will go under current and then in temp_c

code

```
try {
   const url =
`https://api.weatherapi.com/v1/current.json?key=5d9564dfb4
cc4b00bbe104259242003&q=${target}&aqi=yes`;
   const response = await fetch(url);
   const data = await response.json();

   console.log(data);
```

```
    const currentTemp = data.current.temp_c;
    const currentCondition = data.current.condition.text;
    const locationName = data.location.name;
    const localTime = data.location.localtime;
    const conditionEmoji = data.current.condition.icon;
    console.log(currentTemp, currentCondition,
locationName, localTime, conditionEmoji);
 }
```

DOM update

1. To print and display the data on the website, we will use DOM. The data has to be changed dynamically and for that, DOM has to be used. We will use the classes earlier written in HTML to target and dynamically set values. The code for DOM part-
2. Code append in try block

```
// DOM code
    const temperatureField = document.querySelector(".temp");
    const cityField = document.querySelector(".time_location
p");
    const dateField = document.querySelector(".time_location
span");
    const emojiField =
document.querySelector(".weather_condition img");
    const weatherField =
document.querySelector(".weather_condition span");
    const searchField = document.querySelector(".searchField");
    const form = document.querySelector("form");
```

3. Now, we want to make our search bar work. We will add an event listener to it. Whenever there is an event performed, the event listener will execute the callback function passed as a parameter.
4. The search function changes the value of the target(the target city) to the value that we have entered. After typing the city and pressing the submit button, the form will get submitted and the "submit" event will get invoked.

```
form.addEventListener("submit", search);


   //search- callback function


   function search() {
     target = searchField.value;
     fetchData(target);
   }
```

5. Complete code till now

```
const target = "Pune";
const temperatureField = document.querySelector(".temp");
const cityField = document.querySelector(".time_location p");
const dateField = document.querySelector(".time_location span");
const emojiField = document.querySelector(".weather_condition img");
const weatherField = document.querySelector(".weather_condition span");
const searchField = document.querySelector(".searchField");
const form = document.querySelector("form");

async function fetchData(target) {
 try {
   const url =
`https://api.weatherapi.com/v1/current.json?key=5d9564dfb4cc4b00bbe104259242003
&q=${target}&aqi=yes`;
   const response = await fetch(url);
```

```
    const data = await response.json();
    console.log(data);
    const currentTemp = data.current.temp_c;
    const currentCondition = data.current.condition.text;
    const locationName = data.location.name;
    const localTime = data.location.localtime;
    const conditionEmoji = data.current.condition.icon;
    console.log(
      currentTemp,
      currentCondition,
      locationName,
      localTime,
      conditionEmoji
    );
    // DOM code
    form.addEventListener("submit", search);
    //search- callback function

    function search() {
      target = searchField.value;
      fetchData(target);
    }
  } catch (error) {
    console.log(error);
  }
}

fetchData(target);
```

6. Run the code and add breakpoint to see how even after we enter Patna and click submit, ultimately the function is called again with value Pune
7. After we run the code, we see that even after entering the name of the city in the search bar, the results are not getting updated. It is because, the form is submitting the value somewhere and

page is getting refreshed. This is the nature of the form that after it is submitted, the page is refreshed.

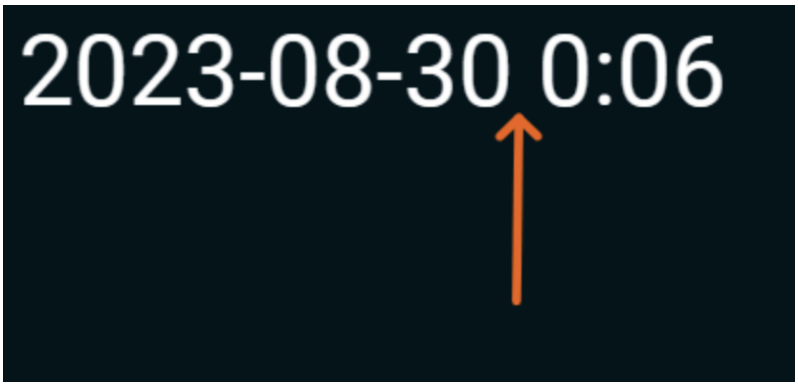
8. For that, we use the preventDefault() method. Our code for the search() function becomes
9. Now, we will update the data on our page. A function updateDOM() will be created which will contain the values to be updates as parameters.
10. How do we update the temp, img and other part from the parameters
    a. Discuss via code that we will use inner text property
    b. Code

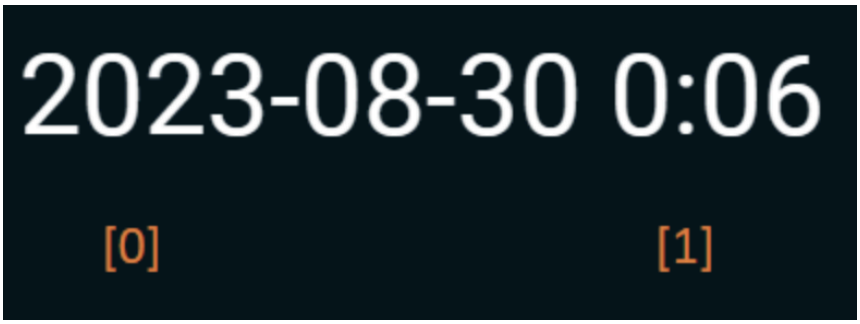
```
function updateDOM(temp, locationName, time, emoji, condition) {
 temperatureField.innerText = `${temp}°C`;
 cityField.innerText = locationName;
 dateField.innerText = time;
 emojiField.src = emoji;
 weatherField.innerText = condition;
}
```

    c. Call the updateDOM method from fetchData
    d. code

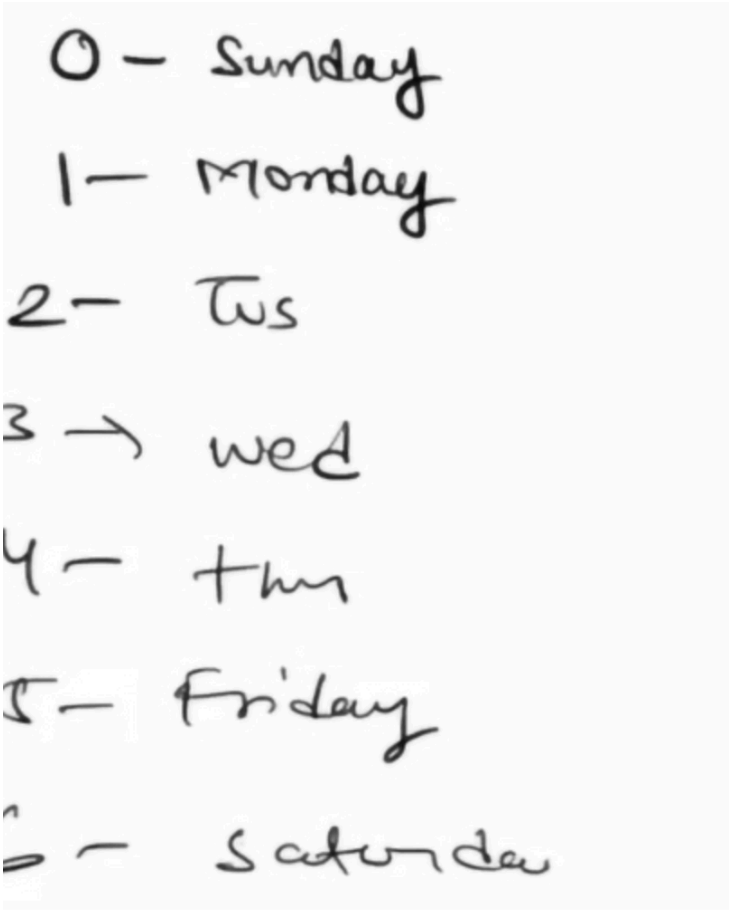
```
// DOM code
   form.addEventListener("submit", search);
   updateDOM(currentTemp, locationName, localTime, conditionEmoji, currentCondition);
```

    e. Discuss the DOM update via debugger

11. Add Day (MOnday, Tue, etc here)
    a. Since the date and time is separated by a space, we will use the split() method to separate them.
    b. 

    c. The split() method splits the value at the delimiter provided and returns an array. Here. after splitting the above value, we get the date at 0th position and time at 1st position.
    d. 

    e. const exactTime = time.split(" ")[1]
    f. const exactdate = time.split(' ')[0]
    g. To convert date to day, we follow-
    h. const exactDate = new Date(exactdate).getDay()
    i. The output is a number which indicate the day in form of a number.



j.

k. We will write another function that converts the day number to day.

```
function getDayFullName(num) {
 switch (num) {
   case 0:
     return "Sunday";

   case 1:
     return "Monday";

   case 2:
     return "Tuesday";

   case 3:
```

```
      return "Wednesday";

    case 4:
      return "Thursday";

    case 5:
      return "Friday";

    case 6:
      return "Saturday";

    default:
      return "Don't Know";
  }
}
```

Complete code

```
let target = "Pune";
const temperatureField = document.querySelector(".temp");
const cityField = document.querySelector(".time_location p");
const dateField = document.querySelector(".time_location span");
const emojiField = document.querySelector(".weather_condition img");
const weatherField = document.querySelector(".weather_condition
span");
const searchField = document.querySelector(".searchField");
const form = document.querySelector("form");

async function fetchData(target) {
 try {
   const url =
`https://api.weatherapi.com/v1/current.json?key=5d9564dfb4cc4b00bbe10
4259242003&q=${target}&aqi=yes`;
   const response = await fetch(url);
   const data = await response.json();
```

```
    console.log(data);
    const currentTemp = data.current.temp_c;
    const currentCondition = data.current.condition.text;
    const locationName = data.location.name;
    const localTime = data.location.localtime;
    const conditionEmoji = data.current.condition.icon;
    console.log(
      currentTemp,
      currentCondition,
      locationName,
      localTime,
      conditionEmoji
    );
    // DOM code
    form.addEventListener("submit", search);
    updateDOM(
      currentTemp,
      locationName,
      localTime,
      conditionEmoji,
      currentCondition
    );
    //search- callback function

    function search(e) {
      e.preventDefault();
      target = searchField.value;
      fetchData(target);
    }
  } catch (error) {
    console.log(error);
  }
}

fetchData(target);
```

```
function updateDOM(temp, locationName, time, emoji, condition) {
 temperatureField.innerText = `${temp}°C`;
 cityField.innerText = locationName;
 emojiField.src = emoji;
 weatherField.innerText = condition;

 const exactTime = time.split(" ")[1];
 const exactdate = time.split(" ")[0];
 const exactDay = getDayFullName(new Date(exactdate).getDay());
 dateField.innerText = `${exactTime}   ${exactDay}   ${exactdate}`
 console.log(exactDay);
}

function getDayFullName(num) {
 switch (num) {
   case 0:
     return "Sunday";

   case 1:
     return "Monday";

   case 2:
     return "Tuesday";

   case 3:
     return "Wednesday";

   case 4:
     return "Thursday";

   case 5:
     return "Friday";

   case 6:
```

```
    return "Saturday";

  default:
    return "Don't Know";
 }
}
```