**WEATHER APP NOTES.**

**1.getElement vs querySelector vs querySelectorAll**

 **Flexibility**:

* querySelector and querySelectorAll allow the use of any valid CSS selector, giving more flexibility in how elements are selected (e.g., combinations of classes, IDs, attributes, pseudo-classes, etc.).
* getElement methods are more limited, with each method focusing on a specific way to select elements (ID, class, or tag).

 **Return Types**:

* getElementById: returns a single element or null if no match is found.
* getElementsByClassName and getElementsByTagName: return **live HTMLCollections**.
* querySelector: returns a first single element or null if no match is found.
* querySelectorAll: returns a **static NodeList**.

 **Performance**:

* In most cases, getElementById is the fastest since it directly accesses an element by ID, while querySelector has more overhead due to CSS selector parsing

**2.What is an api?**

An API (Application Programming Interface) is a set of rules and protocols that allow different software applications to communicate with each other. It defines how requests and data are sent and received between systems, enabling them to interact and share information or functionality.

**3.What is open weather app api?**

The OpenWeather API is a service provided by **OpenWeather** that allows developers to access real-time weather data, forecasts, and historical weather information for any location around the world

**4.blur()**

In JavaScript, the blur() method is used to remove focus from an element, typically an input or other form control. When an element loses focus, certain actions may be triggered, such as form validation or UI changes.

**5.async function, a promise and await keyword**

An **async function** is a function that allows you to write asynchronous code in a way that looks and behaves more like synchronous code (i.e., it executes in a readable, step-by-step fashion).It automatically returns a promise. By using an async function, you can pause the execution of the function until a promise is resolved or rejected.

A **promise** in JavaScript is an object that represents the eventual completion (or failure) of an asynchronous operation and its resulting value. Promises are used to handle asynchronous tasks, such as fetching data from a server, in a more readable and manageable way.

The **await keyword** can only be used inside an async function. It makes JavaScript wait until the promise it’s called on is settled (resolved or rejected). Once the promise is settled, it either returns the resolved value or throws an error.

**6.fetch()**

The **fetch()** function in JavaScript is a modern and powerful way to make network requests to retrieve resources such as files, APIs, or data from web servers. It is built into most modern browsers and returns a **promise.**

**7.what is a json?**

**JSON (JavaScript Object Notation)** is a lightweight data format used to represent structured data. It is easy for both humans to read and write, and for machines to parse and generate. JSON is commonly used for data exchange between a server and a client in web applications.

**8.template literals in js `${}`**

**Template literals** in JavaScript are a way to work with strings that offer more flexibility compared to traditional string concatenation. They allow you to embed variables and expressions directly within a string using placeholders, and they also support multi-line strings easily.

const a = 10;

const b = 20;

const result = `The sum of a and b is ${a + b}.`; console.log(result); // Output: The sum of a and b is 30.

**9.destructuring an array**

**Nested array destructuring** in JavaScript allows you to extract values from arrays within arrays, providing a way to handle complex data structures in a concise and readable manner.

**Basic Syntax**

To destructure nested arrays, you use a combination of square brackets to match the structure of the array.

**Example: Basic Nested Destructuring**

javascript

Copy code

const nestedArray = [1, [2, 3], 4];

// Extract values from the nested array

const [first, [second, third], fourth] = nestedArray;

console.log(first); // 1

console.log(second); // 2

console.log(third); // 3

console.log(fourth); // 4

In this example:

* first gets the value 1.
* second and third are extracted from the inner array [2, 3].
* fourth gets the value 4.

**Skipping Elements**

You can skip elements in the nested arrays by leaving spaces in the destructuring pattern.

**Example: Skipping Elements**

javascript

Copy code

const nestedArray = [1, [2, 3, 4], 5];

// Skip the third element of the inner array

const [first, [second], ] = nestedArray;

console.log(first); // 1

console.log(second); // 2

In this example:

* The third element of the inner array (4) is skipped.

**Default Values**

Default values can be used for nested destructuring to handle cases where the nested array or its elements might be undefined.

**Example: Default Values**

javascript

Copy code

const nestedArray = [1, [2]]; // Inner array has only one element

// Provide default values

const [first, [second, third = 3]] = nestedArray;

console.log(first); // 1

console.log(second); // 2

console.log(third); // 3 (default value)

In this example:

* Since the inner array [2] does not have a third element, third takes the default value 3.

**Destructuring with Rest Parameter**

You can use the rest parameter (...) to capture the remaining elements in a nested array.

**Example: Rest Parameter**

javascript

Copy code

const nestedArray = [1, [2, 3, 4], 5];

// Capture the remaining elements

const [first, [second, ...rest], last] = nestedArray;

console.log(first); // 1

console.log(second); // 2

console.log(rest); // [3, 4]

console.log(last); // 5

In this example:

* rest captures the remaining elements of the inner array [3, 4].

**10.destructuring an object (also contains an array as an attribute)**

Destructuring an object that contains an array as one of its attributes can be done in a way that extracts values from both the object and the array simultaneously. This allows you to work with nested data structures more easily and concisely.

### Example Object Structure

Suppose you have an object that contains an array as an attribute:

javascript

Copy code

const data = {

title: 'Weather Report',

location: 'New York',

forecasts: [

{ date: '2024-09-18', temperature: 22, description: 'Sunny' },

{ date: '2024-09-19', temperature: 20, description: 'Cloudy' }

],

source: 'WeatherAPI'

};

### Destructuring Example

To destructure this object and extract values from both the object and the array inside it, you can use the following code:

javascript

Copy code

const {

title,

location,

forecasts: [

{ date: firstDate, temperature: firstTemp, description: firstDesc },

{ date: secondDate, temperature: secondTemp, description: secondDesc }

],

source

} = data;

**11.toLocaleDateString()**

The toLocaleDateString() method in JavaScript is used to convert a Date object into a string representing the date in a locale-sensitive format. This method allows you to specify different options for date formatting, making it versatile for displaying dates in various formats based on locale and preferences.

**Basic Syntax**

javascript

Copy code

date.toLocaleDateString([locales], [options])

* **locales (optional)**: A string with a BCP 47 language tag (e.g., 'en-US', 'fr-FR') or an array of such strings that represent the locale to use for formatting. If omitted, the default locale of the environment is used.
* **options (optional)**: An object that allows you to customize the date format. This object can include properties like year, month, day, weekday, and more.

**Example Usage**

**Basic Example:**

javascript

Copy code

const date = new Date();

console.log(date.toLocaleDateString()); // Output might be: "9/17/2024" (format varies by locale)

In this example, the date is formatted according to the default locale settings of the environment.

**Specifying a Locale:**

javascript

Copy code

const date = new Date();

console.log(date.toLocaleDateString('en-US')); // Output: "9/17/2024" (for U.S. locale)

console.log(date.toLocaleDateString('fr-FR')); // Output: "17/09/2024" (for French locale)

Here, the date is formatted according to the specified locales ('en-US' and 'fr-FR').

**Customizing Format:**

javascript

Copy code

const date = new Date();

const options = {

year: 'numeric',

month: 'long',

day: 'numeric' };

console.log(date.toLocaleDateString('en-US', options)); // Output: "September 17, 2024"

console.log(date.toLocaleDateString('fr-FR', options)); // Output: "17 septembre 2024"

In this example:

* year: 'numeric' specifies that the year should be shown in numeric form.
* month: 'long' specifies that the full name of the month should be used.
* day: 'numeric' specifies that the day should be shown in numeric form.

**Summary**

* **toLocaleDateString()**: Converts a Date object to a locale-sensitive string.
* **locales**: Optionally specify a locale to format the date according to regional settings.
* **options**: Customize the date format using properties like year, month, day, weekday, etc.

This method is useful for displaying dates in a user-friendly format that adheres to regional and language conventions.

**12.nextFiveDays function**

The nextFiveDays function is an asynchronous function that retrieves and displays weather forecast data for the next five days for a specified city. Here's a detailed breakdown of what each part of the function does:

### Function Breakdown

javascript

Copy code

async function nextFiveDays(city) {

const forcastData = await getFetchData('forecast', city);

const timeTaken = '12:00:00';

const todayDate = new Date().toISOString().split('T')[0];

forcastContainer.innerHTML = '';

forcastData.list.forEach(forcastWeather => {

if (forcastWeather.dt\_txt.includes(timeTaken) && !forcastWeather.dt\_txt.includes(todayDate)) {

updateForcastItems(forcastWeather);

}

});

}

### Detailed Explanation

1. **async function nextFiveDays(city)**:
   * This defines an asynchronous function named nextFiveDays that takes a single parameter city, which represents the city for which the weather forecast will be fetched.
2. **const forcastData = await getFetchData('forecast', city);**:
   * This line calls the getFetchData function with the endpoint 'forecast' and the city name.
   * The await keyword is used to pause the execution of the nextFiveDays function until getFetchData resolves its promise and returns the forecast data.
   * The result is stored in the forcastData variable.
3. **const timeTaken = '12:00:00';**:
   * This defines a constant timeTaken that represents the specific time of day (12:00 PM) for which the forecast data should be considered.
4. **const todayDate = new Date().toISOString().split('T')[0];**:
   * This line gets the current date, formats it to ISO string format, and splits it at the 'T' character to get only the date part (YYYY-MM-DD).
   * The result is stored in the todayDate variable, representing today's date.
5. **forcastContainer.innerHTML = '';**:
   * This clears any existing content within the forcastContainer element. This element is presumably used to display the forecast data.
6. **forcastData.list.forEach(forcastWeather => { ... })**:
   * This iterates over each item in the forcastData.list array, where each item represents a forecast data entry.
7. **if (forcastWeather.dt\_txt.includes(timeTaken) && !forcastWeather.dt\_txt.includes(todayDate))**:
   * This condition checks if the forecast timestamp (forcastWeather.dt\_txt) includes the specific time ('12:00:00') and does not include today's date (todayDate).
   * This ensures that only forecast data for times that are not today and are at 12:00 PM are processed.
8. **updateForcastItems(forcastWeather);**:
   * If the condition is met, this function is called with the forcastWeather object.
   * The updateForcastItems function is presumably responsible for updating the UI with the relevant forecast information.

### Summary

* **Purpose**: Fetch and display weather forecast data for the next five days at 12:00 PM for a specified city.
* **Process**:
  + Fetch data from the API.
  + Clear the previous forecast content.
  + Iterate through the forecast data.
  + Filter for entries that are at 12:00 PM and not for today.
  + Update the UI with the filtered forecast data.

This function is typically used in weather applications or dashboards to present weather forecasts in a user-friendly format.