# Fetching Data From URLs

#### **Learning Outcome Addressed**

  3. Write async functions to fetch data from multiple sources

Follow the instructions in Next Tech to complete the activity.

### Fetching Data From Multiple Sources Asynchronously

A web page pulls data from multiple sources using asynchronous calls. The web page will use the data when it is received, but not necessarily in the order that it was requested. An example of this is a website that shows multiple pictures on the main page. The page may load those pictures asynchronously and show them as the data is received. If this is the case, then the pictures will show up at different times depending on how fast the request goes out and the response comes back.

As shown in the videos this week, an asynchronous fetch is a request from one website to another for data. Using the await keyword, you can wait for a response to come back before continuing to execute your code. The following code may be used as an example on how to fetch data from a URL while using the await keyword:

async function a() {  
  let url = 'https://fonts.googleapis.com/css2?family=Tangerine:wght@700&display=swap';  
  const res = await fetch(url);  
  const data = await res.text();  
  console.log(data);  
}

Another way to wait for a response to come back before continuing to execute your code is with a promise (p is the promise in the below example):

p.then(onFulfilled[, onRejected]);  
   
p.then(value => {  
  // Fulfillment  
}, reason => {  
  // Rejection  
});  
// This code may execute before the fulfillment or rejection due to the asynchronous nature of the request

For more information on promises feel free to read all about them on MDN - <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise>

### In this activity, there are two steps

1. **First, you will fetch all the weather data from the OpenWeather API**
2. **Second, you will identify the specific weather data to show based on a given city name**

### Step 1 - Your first step in this activity is to fetch data from the OpenWeather API

You will use the following URLs. But first, replace YOUR-API-KEY-HERE with the API key you generated on [OpenWeather](https://home.openweathermap.org/api_keys" \t "_blank).

* https://api.openweathermap.org/data/2.5/weather?q=London,uk&APPID=YOUR-API-KEY-HERE
* https://api.openweathermap.org/data/2.5/weather?q=Houston&APPID=YOUR-API-KEY-HERE
* https://api.openweathermap.org/data/2.5/weather?q=Lisbon&APPID=YOUR-API-KEY-HERE
* https://api.openweathermap.org/data/2.5/weather?q=Budapest&APPID=YOUR-API-KEY-HERE

To accomplish this task, you need to:

* Call the makeRequest function to asynchronously fetch the data from the URLs and return the data.
* Store the data in the provided Array[] variable named dataArray, which is a parameter in the fetchData function, using the Array.push method.
* Run logic to check which data are for the city Lisbon
* Call the addLisbonDataToDocument function described in the second half of this task with your Lisbon data and the dataArray that you filled up.

The structure of the array should look like this:

[data1, data2, data3, data4];

where each data variable holds the result from each of the four URLs provided.

The code for fetching, storing, and printing the fetched data should all reside within the fetchData function in the provided fetchUrls.js JavaScript file.

An example response from one of these URLs will result in the following data:

Note that the URL appears to be getting weather data related to a certain city which is specified in the URL. In the URL, this parameter is sent as *q=[cityname]*, such as London, Budapest, Houston, or Lisbon. In the response, you can also see a field called "name" where that city should match whatever was sent in the URL.

{  
  "coord": { "lon": -0.13, "lat": 51.51 },  
  "weather": [  
    {  
      "id": 802,  
      "main": "Clouds",  
      "description": "scattered clouds",  
      "icon": "03n"  
    }  
  ],  
  "base": "stations",  
  "main": {  
    "temp": 285.96,  
    "feels\_like": 286.14,  
    "temp\_min": 285.37,  
    "temp\_max": 286.48,  
    "pressure": 1021,  
    "humidity": 93  
  },  
  "visibility": 10000,  
  "wind": { "speed": 0.5, "deg": 0 },  
  "clouds": { "all": 40 },  
  "dt": 1604960439,  
  "sys": {  
    "type": 1,  
    "id": 1414,  
    "country": "GB",  
    "sunrise": 1604905727,  
    "sunset": 1604938804  
  },  
  "timezone": 0,  
  "id": 2643743,  
  "name": "London",  
  "cod": 200  
}

Hints:

* Use the *await* keyword for both sending the request out to URL and fetching the data response as shown in the fetch example above.
* Be sure to parse the received data as text, just like the example, in order to proceed with the second step of this exercise.

**Finding Specific Data in the Results**

### Reference data for a specific URL within an array

### Step 2 - Identify weather data to show based on a given city name

Now that you have fetched all of the weather data in step one, your task is to identify the weather data for the city of **Lisbon**. You will need to:

* Write logic to search through the weather data array for the weather data pertaining to the city **Lisbon**
* Send that weather data along with your full array to the addLisbonDataToDocument function which takes it and adds it to the DOM so you can see it in your browser.

The code to store the text of a JSON response inside an object is shown below:

...  
const response = await fetch(url);  
const data = await response.text();  
const object = JSON.parse(data); // Stored in the format like

{title: 'dataTitle', dataSize: 42}  
console.log("object:", object);  
...

* Print out some fields of that response using **dot notation** to see how logic can check specific values of the response. This will help you determine which data are relevant to which city. Dot notation for the object variable could be printed in your console like this: console.log(object.dataSize). This would show us the number 42.
* Call the addlisbonDataToDocument(yourDataHere). It should include the following:
  + Lisbon data as the first parameter
  + dataArray with all of the data as the second parameter
* Look at the function comment in the fetchUrls.js file to see an example. addlisbonDataToDocument(yourDataHere) is the function that takes the results of your data and adds it to the DOM so you can see it in your browser.

Hints:

* All code written for this portion should also be in the *fetchUrls.js* JavaScript file provided. You will need to insert it in the correct place in your asynchronous code to ensure that it does not execute until all the data have been stored.
* Print out your data responses at each step of the process and then try to print out specific fields. You can use this information to write logic based on certain field values in the data.
* Remember that you can access the fields of objects by using dot notation as follows: *object.fieldYouWantTheValueOf*
* Although there are multiple ways to complete this exercise, be sure to store the data as directed and pass it in the correct order to the provided *addLisbonDataToDocument* function, i.e., *addLisbonDataToDocument(dataForLisbon, arrayWithAllData);*

Tasks

Looping through the array of URLs provided, fetch the data asynchronously and store the responses in the other array, dataArray. This test will only succeed once you complete the second task and send both parameters to the addLisbonDataToDocument function correctly.

**1**

You will need to use write logic which checks the name field of your object response and then sends that to the addLisbonDataToDocument function as the first argument and your full dataArray variable as the second argument.