Week 8: Weather API

This week we extend our understanding of API's and ways of incorporating external data sources into a p5.js application. The exercise described below loads historic weather data from https://open-meteo.com.

Key concepts

Asynchronous operations

It takes time for your p5.js sketch to grab data from a remote source. As such, if the draw function has to wait for data to load then this can cause any animations to stop functioning. Therefore, the **loadJSON();** function in p5.js can fetch data in the background while **draw()** continues to do its work. The ability of separate JavaScript functions to perform different tasks at the same time (rather than one after another) is the very definition of an asynchronous function - the origins of which can be traced back to 1965, if not before.

Functions

Functions in JavaScript and other programming languages provide a way to repeatedly run segments of code without having to rewrite them multiple times.

JSON

Stands for **J**ava**S**cript **O**bject **N**otation and is a text format for storing and transporting data between separate software applications or systems. The JSON formatted data we will work with this week looks a bit like the following code. With Javas

```
{
  "latitude": 51.5,
  "longitude": -0.25,
  "generationtime_ms": 7.1010589599609375,
  "utc_offset_seconds": 0,
  "timezone": "Europe/London",
  "timezone_abbreviation": "GMT",
  "elevation": 69,
  "daily_units": {
    "time": "iso8601",
    "temperature_2m_max": "°C",
    "rain_sum": "mm"
  },
  "daily": {
    "time": [
      "1963-11-11"
    ],
    "temperature_2m_max": [
      13
    ],
    "rain_sum": [
      5.6
    ]
  }
}
```

Working with JSONin JavaScript

In p5.js we load and access JSON formated data using the loadJSON() function. For example, if we write the following code into JavaScript browser console we can create a a JSON object then access its variables via a dot syntax.

```
js = {"latitude":55.6, "longitude":33.7}; // Create a JSON object.
js.longitude;                             // Prints the 'longitude'
```

Constructing a URL for an API

Uniform **R**esource **L**ocator's (URL's) provide a human-readable address to access information on a remote web server. In addition to providing a unique web address for information on the internet URL's also allow you to pass variables to a server to obtain specific information. For example, the following URL provides access to the open-access weather API provide by open-meteo.com:

```
https://api.open-meteo.com/v1/forecast
```

We have to add variables to the URL above to instruct Meteo's API to send us specific data. Adding latitude, longitude, daily, and timezone variables after a question mark at the end of the word 'forecast' will return JSON formatted data. For example, accessing the following URL (all on one line) into a browser returns a forecast for the maximum temperature in London over the next seven days. Note how each variable is separated by an ampersand (&) symbol:

```
https://api.open-meteo.com/v1/forecast?latitude=51.5002&longitude=-0.1262&daily=temperature_2m_max&timezone=auto
```

The URL above should return JSON data that looks something like this. which is a little difficult to read without either posting into a JSON formater such as https://www.jsonformatter.io or viewing in a browser such as Firefox which provides default functionality for clearly displaying JSON data:

```
{"latitude":51.5,"longitude":-0.120000124,"generationtime_ms":0.8929967880249023,"utc_offset_seconds":0,"time
zone":"Europe/London","timezone_abbreviation":"GMT","elevation":6.0,"daily_units":{"time":"iso8601","temperat
ure_2m_max":"°C"},"daily":{"time":["2022-11-11","2022-11-12","2022-11-13","2022-11-14","2022-11-15","2022-11-
16","2022-11-17"],"temperature_2m_max":[15.8,17.2,16.5,14.8,11.5,12.6,14.0]}}
```

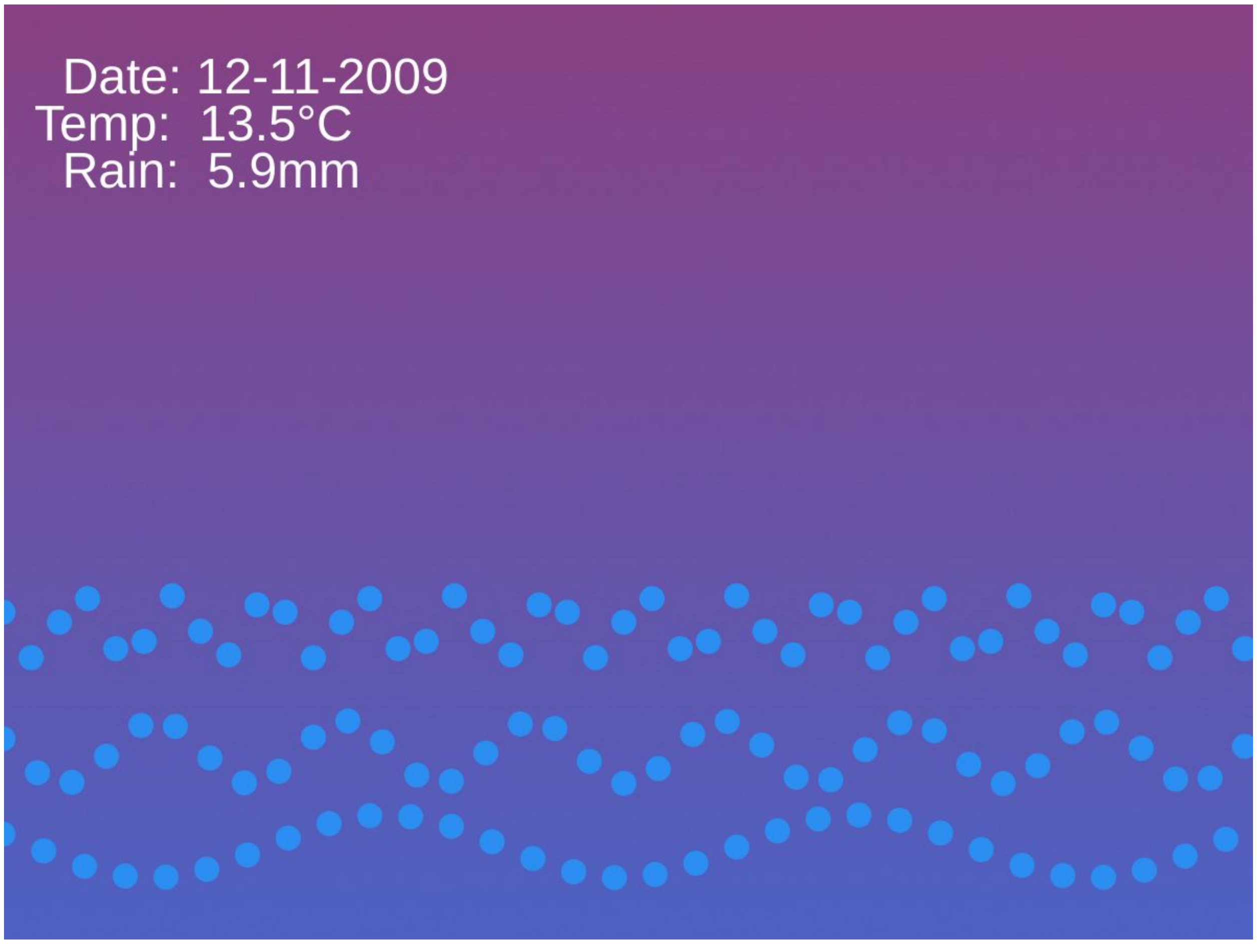
Open Meteo's URL builder

Open Meteo helpfully provide online forms to help you construct URL's for their API.

- Build a forecast URL: https://open-meteo.com/en/docs#api_form
- Access historic data: https://open-meteo.com/en/docs/historical-weather-api#api_form

Exercise: Using the weather API

Now lets use the weather API. We are going to make a sketch which looks something like the following image. The sketch loads the total rainfall and the maximum temperature in London for the current day and month, for a random year over the past sixty years. The amount of rain determines how fast the dots move in a sine wave pattern.



Step 1: Create a new p5.js sketch

First create a new Create a new p5.js within VScode with the menu item '**View->Command Palette->Create p5.js Project'**.
I've created mine in a folder called **weather-api** in a **week8-api** folder within my weekly examples. Note: If '**Create p5.js Project'** is not available in the command pallet then you need to install Sam Lavigne's VScode extension for p5.js:

1. Visit Sam's VScode marketplace

2. Click the "Download Extension" link at which will download a file ending in .vsix

2. In '**View->Command Pallete'** search for and run the '**Extensions: Install from VSIX'** command

3. Select the .vsix file you just downloaded and the extension should be installed

Step 2: Initialise variables and setup()

In our sketch, define some global variables, initialises a canvas to draw in, and create a timer with **setInterval().** The timer calls a function called **countdown()** every second asynchronously to the draw function. In setup() we also load an image called '**temp.png**' which you should download into your sketch folder - this image is also available in this weeks example on GIT.

```
let weatherjson = false;
let weatherloaded = 0;
let counter = 0;
let yr, ukdate;
let theta = 0.0; // Calculating speed of waves
let img;

function setup() {
  createCanvas(400, 400);
  setInterval(countdown, 1000); // Call 'setcounter' every second
  img = loadImage('temp.png');
}
```

Step 3: Create the countdown() function

The countdown function counts down from thirty to zero by decreasing the **counter** variable by one each time the **countdown()** function is called by the **setInterval()** as specified above. The function also builds an **open-meteo.com** formatted URL so we can request the maximum temperature and amount of rain for a specific date in London. Once the URL is constructed the **loadJSON()** function loads the JSON object asynchronously to the countdown timer and draw functions.

```
function countdown(){
  // Variables to work with in the rest of the script
  let m = month();
  let d = day();
  counter--;

  // Timer to load JSON data every 30 seconds
  if(counter<0){
    // Reset countdown to 30 secs
    counter = 30;
    yr = int(random(1963, 2022));
    // Build date strings for the API URL
    let apidate = `${yr}-${m}-${d}`;       // Create a date string in the form of 2022-11-09
    ukdate = `${d}-${m}-${yr}`;            // Create a UK date string in the form of 09-11-2022
    // Build the weather URL
    let weatherurl = "https://archive-api.open-meteo.com/v1/era5?";
    weatherurl += `latitude=51.5002&longitude=-0.1262`;               // Lat and lon for London
    weatherurl += `&start_date=${apidate}&end_date=${apidate}`;       // Start and end dates
    weatherurl += "&daily=temperature_2m_max,rain_sum&timezone=auto"; // Request temperature and rain
    // Retrieve the JSON from the 'weatherurl, then call loadedweather()
    loadJSON(weatherurl, loadedweather);
  }
}
```

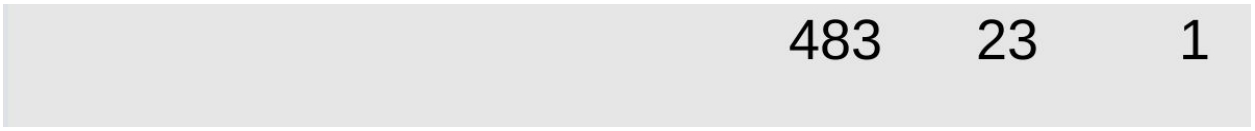
Step 4: Receiving the weather JSON

The LoadJSON() function specified above triggers a **loadedweather()** function once the data is loaded. The function looks like the following and simply saves the loaded json variable into our global variable '**weatherjson**' which we initialised at the top of our sketch.

```
// Called once the JSON is loaded
function loadedweather(json){
  weatherjson = json;
}
```

Step 5: Lets draw something!

In our draw function add a grey background. We also add some text to illustrate the asynchronicity of this sketch where the draw function runs (its framerate) at a different rate to the countdown timer and the loadJSON functions. Your sketch should display some numbers at the top, which update at different timescales, and should look like so:



```
function draw() {
  // Draw a grey background and the timer
  background(230);
  textSize(18);
  text(frameCount, width-150, 18);    // How quickly draw() updates
  text(counter, width-90, 18);        // The countdown timer
  text(weatherloaded, width-25, 18);  // How quickly the JSON updates

  // If the first JSON hasn't loaded then don't draw any more
  if(weatherjson===false) return;
}
```

Step 5: Add some variables and a graphic

Continue adding code to your draw function BEFORE the closing }. The temp and rain variables simply target items named in the weatherjson variable using a dot syntax. Be sure to maintain the indentation of the code so it is clearly associated with the draw() function.

```
  // Otherwise get the date, temp, and rain
  let temp = weatherjson.daily.temperature_2m_max;
  let rain = weatherjson.daily.rain_sum;

  // Add gradiated image to the background
  let pos = map(temp, -20, 40, -1000, 0);
  image(img, 0, pos); // Position is linked to the temp

  // Draw the date, temp, and rainfall text
  let x = 10;
  let y = 30;
  textAlign(LEFT);
  text(`  Date: ${ukdate}`, x, y);
  text(`Temp:  ${temp}°C`, x, y+15);
  text(`  Rain:  ${rain}mm`, x, y+30);
```

Step 6: Advanced exercise: Add some waves

If you want to add some more advanced graphics, then write the following code at end of our draw() function. This provides the basis of generating some dots animated in a wave-like form. Note how we pass four variables (Ypos, spacing, freq, speed) to a mywave() function which we are about to create. **Note:** Be sure to end the draw() function with a closing } bracket!

```
  // Draw some waves
  noStroke();
  fill(43, 142, 240);
  y = height-100;
  speed = map(rain, 0, 15, 0.01, 0.09); // Map rain to wave speed
  mywave(y, 9,  30.0, speed);      // Ypos, spacing, freq, speed
  mywave(y+40, 11,  60.0, speed);  // Ypos, spacing, freq, speed
  mywave(y+70, 13,  150.0, speed); // Ypos, spacing, freq, speed

} // Indicates the end of the draw function!
```

Step 7: The newwave() function

After the closing } bracket of draw() write the following function. This code is evolved from https://p5js.org/examples/math-sine-wave.html. We don;t expect you to understand the maths. Rather, this provides a neat illustration of how you can use functions to make make code re-usable.

```
// Developed from:  https://p5js.org/examples/math-sine-wave.html
function newwave(ypos, xspace, freq, speed){
  let dx = (TWO_PI/freq)*xspace;        // Calc wave frequency
  let w = floor((width+xspace)/xspace); // Calc spacing
  let yvals = new Array(w);             // Store height values
  theta += speed;                       // Calc speed of all waves

  // For every x value calculate a y value with sine function
  let x = theta;
  for (let i=0; i < yvals.length; i++) {
    yvals[i] = sin(x)*10.0; // Height of wave
    x += dx;
  }

  // Draw the wave with an ellipse at each location
  for (let x = 0; x<yvals.length; x++) {
    ellipse(x*xspace, ypos+yvals[x], 8, 8);
  }
}
```

Step 8: The final code

The entire sketch should look like this. Be sure to concentrate on your indentation as it makes thing MUCH easier for you to debug.

```
// See https://open-meteo.com/en/docs#api_form
let weatherjson = false;
let weatherloaded = 0;
let counter = 0;
let yr, ukdate;
let theta = 0.0; // Calculating speed of waves
let img;

function setup() {
  createCanvas(400, 300);
  setInterval(countdown, 1000); // Call 'setcounter' every second
  img = loadImage('temp.png');
}

function countdown(){
  // Variables to work with in the rest of the script
  let m = month();
  let d = day();
  counter--;

  // Timer to load JSON data every 30 seconds
  if(counter<0){
    // Reset countdown to 30 secs
    counter = 30;
    yr = int(random(1963, 2022));
    // Build date strings for the API URL and to draw
    let apidate = `${yr}-${m}-${d}`;
    ukdate = `${d}-${m}-${yr}`;
    // Build the weather URL (see https://open-meteo.com/en/docs#api_form)
    let weatherurl = "https://archive-api.open-meteo.com/v1/era5?";
    weatherurl += `latitude=51.5002&longitude=-0.1262`;
    weatherurl += `&start_date=${apidate}&end_date=${apidate}`;
    weatherurl += "&daily=temperature_2m_max,rain_sum&timezone=auto";
    // Load the JSON
    loadJSON(weatherurl, loadedweather);
  }
}

// Called once the JSON is loaded
function loadedweather(json){
  weatherjson = json;
  weatherloaded++;
}


function draw() {
  // Draw a grey background and the timer
  background(230);
  fill(255);
  textSize(16);
  text(frameCount, width-150, 18);
  text(counter, width-90, 18);
  text(weatherloaded, width-25, 18);

  // If the JSON hasn't loaded then don't go any further
  if(weatherjson===false) return;

  // Otherwise get the date, temp, and rain
  let temp = weatherjson.daily.temperature_2m_max;
  let rain = weatherjson.daily.rain_sum;

  // Add gradiated image to the background
  let pos = map(temp, -20, 40, -1000, 0);
  image(img, 0, pos); // Position is linked to the temp
  // Draw the date, temp, and rainfall text
  let x = 10;
  let y = 30;
  textAlign(LEFT);
  text(`  Date: ${ukdate}`, x, y);
  text(`Temp:  ${temp}°C`, x, y+15);
  text(`  Rain:  ${rain}mm`, x, y+30);

  // Draw some waves
  noStroke();
  fill(43, 142, 240);
  y = height-100;
  speed = map(rain, 0, 15, 0.01, 0.09); // Map rain to wave speed
  newwave(y, 9,  30.0, speed);      // Ypos, spacing, freq
  newwave(y+40, 11,  60.0, speed);  // Ypos, spacing, freq
  newwave(y+70, 13,  150.0, speed); // Ypos, spacing, freq
}

// Derived from:  https://p5js.org/examples/math-sine-wave.html
function newwave(ypos, xspace, freq, speed){
  let dx = (TWO_PI/freq)*xspace;        // Calc wave frequency
  let w = floor((width+xspace)/xspace); // Calc spacing
  let yvals = new Array(w);             // Store height values
  theta += speed;                       // Calc speed of all waves

  // For every x value calculate a y value with sine function
  let x = theta;
  for (let i=0; i < yvals.length; i++) {
    yvals[i] = sin(x)*10.0; // Height of wave
    x += dx;
  }

  // Draw the wave with an ellipse at each location
  for (let x = 0; x<yvals.length; x++) {
    ellipse(x*xspace, ypos+yvals[x], 8, 8);
  }
}
```

Challenges / homework

Challenge A (easy):

You may notice that the ‘counter’, ‘frames’, and ‘weatherloaded’ variables do not display. Can you adjust the sketch so these variables appear again?

Challenge B (easy):

Can you change the look of the waves by altering the variables which are passed to our newwave() function?

Challenge C (difficult):

Can you control the number of waves and their position in response to the changing rainfall data?

Challenge D (v.difficult):

Can you combine incorporate the earthquake data from last week into this sketch?