PART 2

LAB SESSION

✓ **Arrays**

Lab 8a: Create Daily ETHUSD price array (15min)

- In this lab session, the objective is to use the historical ETHUSD data extracted from a public API **CryptoWatch** to construct an array of objects containing 2 properties `date` and `price` in descending order by `date`.
  - `date` is to be represented as a string in `yyyy-mm-dd` format.
  - `price` is a number representing the end of day USD price of ETH.

Lab 8a: Create Daily ETHUSD price array (15min)

1. Download the test data file from https://raw.githubusercontent.com/RoyLai-InfoCorp/fin535/main/fin535-ethusd.json from terminal

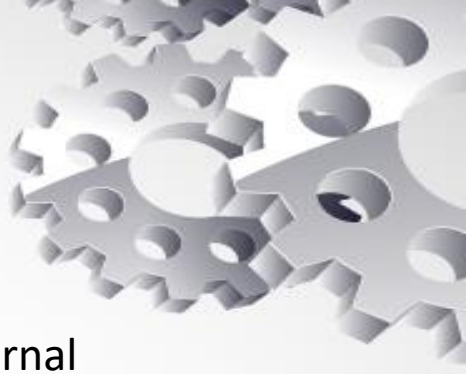
```
$ wget https://raw.githubusercontent.com/RoyLai-InfoCorp/fin535/main/fin535-ethusd.json
```

The file contains ETHUSD candlestick data represented in JSON format that looks similar to this.

```
{
    "result":{
        "86400":[
            [1458086400,12.645,13.421,11.98,13.1,10895.36,0],
            [1458172800,13.1,13.89,12.409,12.548,6127.951,0],
            [1458259200,12.548,12.62,10.436,11.06,26283.299,0],
            [1458345600,11.08,11.198,8.338,10.99,45318.516,0],
            [1458432000,10.99,11.17,9.776,10.573,19636.695,0],
....
}
```

JSON is created by serializing a Javascript object into a string. In other words, it is the string representation of an object. Once the JSON string is loaded into a Javascript variable, it can be accessed like any Javascript object.

Lab 8a: Create Daily ETHUSD price array (15min)



2. Create the file `8a-ethusd-daily.js`.
3. Load the JSON data into a javascript object. The keyword `require` can be used to import an external module(package) to be used by the code or, in this case, importing a JSON file into a Javascript object.

```
// This is used to import the JSON file into a javascript object
const json = require('./fin535-ethusd.json');
```

Lab 8a: Create Daily ETHUSD price array (15min)

4. The structure of the JSON data is as follows:
   - The object contain a property `result`.
   - The object from `result` contains a property `86400`. There is no need to associate any meaning to this number but it represents the number of seconds in one day.
   - The property `86400` contains an array of array with each inner array representing the Open-High-Low-Close candlestick price represented by 7 values in this order:
     - [0] = CloseTime
     - [1] = OpenPrice
     - [2] = HighPrice
     - [3] = LowPrice
     - [4] = ClosePrice
     - [5] = Volume
     - [6] = QuoteVolume
   - To construct the daily end of day price using this data, we only require the CloseTime and ClosePrice.
   - Example:

```
// This is used to load the file directly into a javascript object
const json = require('./fin535-ethusd.json');

// This will read and display the CloseTime and ClosePrice for the 1st array
const price0 = json.result['86400'][0];
console.log('CloseTime: ',price0[0]);
console.log('ClosePrice: ',price0[4]);
```

Lab 8a: Create Daily ETHUSD price array (15min)

5. Notice that the CloseTime is represented as a large number. This number is representing time in seconds. We will use the **date2string** module created in **lab 7** to convert the string into `yyyy-mm-dd` format.
6. Copy **date2string.js** from **lab 7** into current folder and import into the script.

```
// This is used to load the file directly into a javascript object
const json = require('./fin535-ethusd.json');
const date2string = require('./date2string.js');

// This will read and display the CloseTime and ClosePrice for the 1st array
const price0 = json.result['86400'][0];

// The date is in seconds by the function expects milliseconds so muliply 1k
console.log("CloseTime: ", date2string(price0[0] * 1000));
console.log('ClosePrice: ',price0[4]);
```

The result should show:

```
CloseTime:  16-2-2016
ClosePrice:  13.1
```

Lab 8a: Create Daily ETHUSD price array (15min)

7. Now that we are able to successfully extract and parse a single date price, we will apply the same logic to create a new array containing object {date, price}.
   - Notice that the data provided are listed in ascending order by time, we will use the `array.slice().reverse()` function to reverse the order of the array. The purpose of using `slice()` before reversing is to create a new copy so that reverse will not mutate the original copy.
   - We will then construct a new array by chaining the result using the `array.map()` function.

```
array.map((price) => ({
    date: date2string(price[0] * 1000),
    price: price[4],
}))
```

The final JS code is shown below.

```
// This is used to load the file directly into a javascript object
const json = require("./fin535-ethusd.json");
const date2string = require("./date2string.js");

// This will read and create a new array of {date, price} object
const dailyPrices = json.result["86400"].slice().reverse().map((price) => ({
    date: date2string(price[0] * 1000),
    price: price[4],
}));
console.log(dailyPrices);
```

Lab 8b: Create 10-Day Moving Average Array (15min)

- The purpose of this lab is to make further use of the array functions to compute 10-day moving average using the daily price data created in previous lab.
- The 10-day moving average is simply the average of last 10 days of ETHUSD from a particular date. For example, the moving average price on 2022-05-30 is the average of the daily prices from 2022-05-21 to 2022-05-30 and the moving average on 2022-05-01 is the average from 2022-04-22 to 2022-05-01.
- The script should return JSON object called "ETHUSD" that contains an array for 30 days of date-10ma value and date should be in `yyyy-mmm-dd` format.
- Example:

```
$ node daily.js

{
    "ETHUSD": [
        {
            "date":"2022-05-01",
            "10-ma":"xxxxx"
        },
        {
            "date":"2022-05-02",
            "10-ma":"xxxxx"
        },
        ...
    ]
}
```

Lab 8b: Create 10-Day Moving Average Array (15min)

1. Copy and paste from `8a-ethusd-daily.js` into `8b-ethusd-10ma.js` and open `8b-ethusd-10ma.js` in VS Code.
2. Use `array.reduce()` and `array.slice()` to create a moving average function `ma` that returns the average of an array given 3 parameters:
   - numArray - an array of numbers
   - pos - the index (position) of an item in the array
   - len - the number of items after `pos`

```
const num = [1,2,3,4,5]
const ma = (numArray, pos, len)=>{
    return numArray.slice(pos,pos+len).reduce((prev,curr)=>prev+curr)/len;
}
// this will return the moving average of 3 values
console.log(ma(num,0,3))    // average([1,2,3])=2
console.log(ma(num,1,3))    // average([2,3,4])=3
console.log(ma(num,2,3))    // average([3,4,5])=4
```

1. Complete the code by applying the ma function to return 10-day moving average of the ETHUSD price.

Lab 8b: Create 10-Day Moving Average Array (15min)

Answer:

```
// This is used to load the file directly into a javascript object
const json = require("./fin535-ethusd.json");
const date2string = require("./date2string.js");

// This will read and create a new array of {date, price} object
const dailyPrices = json.result["86400"].slice().reverse().map((price) => ({
    date: date2string(price[0] * 1000),
    price: price[4],
}));

// Extract the array of prices to be used for calculating average.
let priceArray = dailyPrices.map((x) => x.price);

// Moving average function that returns moving average determines by `len` days
const ma = (numArray, pos, len) => {
    return (
        numArray.slice(pos, pos + len).reduce((prev, curr) => prev + curr) / len
    );
};

// Use map(x,idx) and pass indx into ma as position
const maPrices = dailyPrices
    .slice(0, dailyPrices.length - 10)
    .map((x, idx) => ({
        date: x.date,
        price: ma(priceArray, idx, 10),
    }));

console.log(maPrices);
```

Lab 8c: Create 10-Day Moving Average Array (15min)

We will be using this 10-day moving average function to create an API service in the next lesson. But before that, we will convert the functions and export it via a module so that it can be reusable.

1. Create the file `maPrices.js` in VS Code.

```
// Convert epoch date into `yyyy-mm-dd`
const date2string = (date) => {
    const d = new Date(date);
    const yyyy = d.getFullYear();
    const mm = d.getMonth() + 1;
    const dd = d.getDate();
    return `${yyyy}-${mm}-${dd}`;
};

// Moving average function that returns moving average determines by `len` days
const ma = (numArray, pos, len) => {
    return (
        numArray.slice(pos, pos + len).reduce((prev, curr) => prev + curr) / len
    );
};
```

Lab 8c: Create 10-Day Moving Average Array (15min)

2. Create a function called **getDailyPrices** that will take in a **json** object representing the raw price data and return an array of **{date,price}** object in ascending order by date.

Lab 8c: Create 10-Day Moving Average Array (15min)

ANSWER:

```
// This will read and create a new array of {date, price} object
const getDailyPrices = (json) =>
    json.result["86400"]
        .slice()
        .reverse()
        .map((price) => ({
            date: date2string(price[0] * 1000),
            price: price[4],
        }));
```

Lab 8c: Create 10-Day Moving Average Array (15min)

3. Create a function called `**maPrices**` that will take in 2 parameters: a `**json**` object representing the raw price data and a `**days**` integer representing the moving average days. This will generalise the function to compute moving averages for any number of days.

Lab 8c: Create 10-Day Moving Average Array (15min)

ANSWER:

```
// This will return the moving average from daily price by number of days
const maPrices = (json, days) => {
    // This will read and create a new array of {date, price} object
    const dailyPrices = getDailyPrices(json);

    // Extract the array of prices to be used for calculating average.
    let priceArray = dailyPrices.map((x) => x.price);

    // Use map(x,idx) and pass indx into ma as position
    return dailyPrices.slice(0, dailyPrices.length - days).map((x, idx) => ({
        date: x.date,
        price: ma(priceArray, idx, days),
    }));
};
```

Lab 8c: Create 10-Day Moving Average Array (15min)

4. Export the function in a module.

```
module.exports = { date2string, getDailyPrices, maPrices, ma };
```

5. Create the file `**8c-ethusd-import.js**`. Create the Javascript code on your own by importing `**maPrices**` and using it to compute the 10-day moving average.

Lab 8c: Create 10-Day Moving Average Array (15min)

ANSWER:

```
// This is used to load the file directly into a javascript object
const json = require("./fin535-ethusd.json");
const { maPrices } = require("./maPrices.js");
const ma10 = maPrices(json,10);
console.log(ma10);
```