Vue is a relatively new JavaScript library for creating HTML from data. Released just a few years ago, in 2014, it has seen very rapid growth in popularity.

Vue has several features that make it good for beginners:

- The Vue documentation is quite good, with a nice introductory video that demonstrates coding with Vue.
    - There are many tutorial web pages, but be sure they are talking about Vue 2.
- You can start using it just by adding some scripts to your HTML pages.
- You can start using it just be learning a few simple concepts.
- Those conceptual ideas also appear in the more popular, but far more complicated Angular and React frameworks.
- As your needs and skills mature, you can switch to a command line tool with webapp building features, again very similar to ones in Angular and React.

Here, we'll show just enough Vue to let you do roughly the same sort of simple dynamic HTML that you could do with jQuery and MustacheJS, to show how Vue makes this kind of code simpler and more robust.

Motivating Example

To make the discussion of Vue concrete, assume we want to make a simple HTML page displaying the daily currency exchange data.

To get today's rates, we can use https://exchangeratesapi.io/. It returns JSON like this (trimmed for length)

```
{
  "base": "EUR",
  "date": "2018-01-19",
  "rates": {
    "AUD": 1.5302,
    "BGN": 1.9558,
    "BRL": 3.9312,
    "CAD": 1.5246,
    ...
    "USD": 1.2255,
    "ZAR": 14.955
  }
}
```

The data is given using ISO 4217 three-letter currency codes. By default, rates are for the Euro. To use a different base, add **?base=***code* to the URL.

To get the full names of the currencies, we can use https://openexchangerates.org/api/currencies.json to get a JSON object with the full names of each currency code. Here's a trimmed example of the JSON returned for https://openexchangerates.org/api/currencies.json

```
{
  "AED": "United Arab Emirates Dirham",
  "AFN": "Afghan Afghani",
  "ALL": "Albanian Lek",
  "AMD": "Armenian Dram",
  ...
  "ZAR": "South African Rand",
  "ZMW": "Zambian Kwacha",
  "ZWL": "Zimbabwean Dollar"
}
```

Note that this object includes many currencies not provided by **fixer.io**.

Our goal is to create a web page that lists the rates for the day, with full names, in a simple table.

We'll put our JavaScript in the file **currency.js**. To get our data, we use this example code. It's OK if you don't know what this code does. All you need to know is that we can use **fetchJsonList()** to get the data from those two URLs.

```
var nameUrl = 'https://openexchangerates.org/api/currencies.json';
var rateUrl = 'https://api.fixer.io/latest';

function fetchJson(url, init) {
  return fetch(url, init).then(function(response) {
    if (response.ok) {
      return response.json();
    }
    throw new Error(response.statusText)
  });
}

function fetchJsonList(urls, init) {
  return Promise.all(urls.map(url => fetchJson(url, init)));
}
```

Loading Vue

To use Vue on a web page, just include the Vue library before any code that uses it. In our case, that means in our HTML page, we'll have

```
<script src="https://unpkg.com/vue/dist/vue.js"></script>
<script src="./currency.js"></script>
```

This loads the *development* version of Vue. The development version is a larger file and runs slightly slower than the *production* version, but it gives much better error messages. When you deploy your web site to the public, you can switch to the production version, if you wish, by using this line instead:

```
<script src="https://cdn.jsdelivr.net/npm/vue"></script>
```

You can also, of course, download and use local copies of either file.

Using Vue to connect data to HTML

The fundamental goal of Vue is to connect data with HTML, in such a way that updating the data in JavaScript automatically updates the HTML.

Let's define an initial HTML file, **index.html**:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Exchange Rates</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/css/bootstrap.min.css"
      rel="stylesheet">
  </head>
  <body>
    <div class="container" id="app">
    </div>

    <script src="https://unpkg.com/vue/dist/vue.js"></script>
    <script src="./currency.js"></script>
  </body>
</html>
```

We don't know what the HTML will look like for showing the data, but we do know that we want that data to appear inside the container labeled **app**. We also have the code to load the Vue library and our JavaScript file, along with Bootstrap, to make our page nicer looking.

Now we connect our HTML to data, in our **currency.js** file, by create a **Vue** object. We put the following at the top of **currency.js**:

```
var app = new Vue({
  el: '#app',
  data: {
    ...
  }
});
```

The variable **app** holds the Vue object. We could use any name we want.

A Vue object is configured by passing it a JSON object. The **el** field specifies the ID of the section of the HTML we want Vue to update with data for us.

The **data** field is where we're going to store the currency data we get from the web. Since we know we're going to get rates and names, let's add fields for those two items. We don't know the values, but we know they will be JSON objects, not arrays, so we'll start with empty objects. In **currency.js**:

```
var app = new Vue({
  el: '#app',
  data: {
    names: {},
    rateData: {}
  }
});
```

We could have used **rates** for the field name, but that we'd have two fields named **rates**. To avoid writing things like **rates.rates**, we use **rateData** for the data returned from the rate server.

Now let's **fetchJsonList()** at the end of our JavaScript file to get the JSON data from the servers and store it in our Vue object. In **currency.js**:

```
var app = new Vue({
   ...
});

var nameUrl = 'https://openexchangerates.org/api/currencies.json';
var rateUrl = 'https://api.fixer.io/latest';
...

fetchJsonList([nameUrl, rateUrl]).then(function(values) {
  app.names = values[0];
  app.rateData = values[1];
}).catch(function(error) {
  alert(error);
});
```

Note that we write **app.names** and **app.rateData**, not **app.data.names** and **app.data.rateData**. That's because Vue did some magic with the **data** information when the Vue object was created. Vue defined setters for each of the fields list in **data**. So when we write **app.names = ...** we are actually calling a setter method to store the data. That method is defined by Vue to take care of updating our HTML for us.

> This is why there's an important rule about creating Vue objects: You must include any fields you plan to change when you create it, so that the setters will be created that manage the HTML for you. That's why we created the **names** and **rateData** fields with empty objects.

Defining a Vue HTML template

Now we know what our **data** looks like. It has

- a **rateData** field containing an object with
    - a **date** field containing the date when the query was made
    - a **base** field containing a currency code
    - a **rates** field containing an object where
        - each key is a currency code
        - the value is the exchange for that currency, relative to the base
- a **names** field containing an object where
    - each key is a currency code
    - the value is the full name of that currency

We would like this data to appear in a simple table. The table will have three columns:

- the currency code
- the currency name
- the currency's current exchange rate, relative to the Euro

We can start to fill in our HTML now.

```
<div class="container" id="app">
  <table class="table">
    <tr>
      <th>Currency</th><th>Full Name</th><th>Rate</th>
    </tr>
  </table>
</div>
```

Now for the important part. We want to create a row for each currency, i.e., we want to iterate over the keys in the **rateData**.

In Vue, you do this by adding special forms that define an HTML template to display the data. These forms are designed to be as readable as possible. Here's how to do what we want here:

```
<div class="container" id="app">
  <table class="table">
    <tr>
      <th>Currency</th><th>Full Name</th><th>Rate</th>
    </tr>
    <tr v-for="(value, key) in rateData.rates">
      <td>{{key}}</td><td>{{names[key]}}</td><td>{{value}}</td>
    </tr>
  </table>
</div>
```

The **v-for** attribute is how you tell Vue to repeat some HTML for a set of data. There are two common forms:

```
<tr v-for="item in items">...</tr>
```

This repeats the **tr** HTML template for every item in the list **items**, where **items** is the name of a field in your **data** object. This is for arrays.

In our case, we have an object with keys, so we use the second form:

```
<tr v-for="(value, key) in obj">...</tr>
```

This repeats the **tr** template for every key-value pair in the object **obj**, where **obj** is the name of a field in your **data** object.

v-for can be used to repeat any kind of HTML element, e.g.,

- <tr v-for= ...> to make a row in a table for every data item
- <li v-for= ...> to make a list element for every data item
- <p v-for= ...> to make a paragraph for every data item
- and so on

Inside the **tr** above we have this line:

```
<td>{{key}}</td><td>{{names[key]}}</td><td>{{value}}</td>
```

The **{{...}}** notation tells Vue to insert the value of the expression given inside the braces. This expression can be

- a field in our **data** object
- a variable defined by **v-for**
- an expression using fields and variables

{{...}} only works inside HTML content, i.e., where text would appear. If you want to insert some JavaScript value into an attribute, you use **v-bind:**. See the Vue

documentation for examples.

When the **v-for** iterates over the data, it sets **key** and **value** to successive keys and values from **rateData.rates** object. The line of code we have here says to put

- the key, i.e., the currency code, in the first column
- the value, i.e., the exchange rate, in the third column
- the value returned by **names[key]** in the second column

names is the JSON object of currency names, so **names[key]** will return the full name of the currency code in **key**.

Exercise

Test your understanding of how this code works. Add a **H1** element in the HTML before the table, that gives the date and base currency for this data, e.g., "Exchange rates for the Euro as of 2018-01-19".