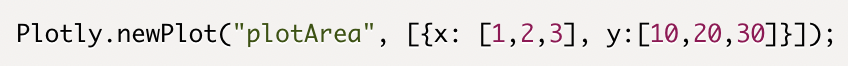
**Plotly.js Chart**

The Plotly.newPlot() method creates a new chart.

This method has two arguments:

1. The first argument in Plotly,newPlot() is "plotArea".
   * Corresponds to the ID of the <div> tag in the HTML document. (in this exmpl.)
2. The second argument is an array, as indicated by the square brackets.
   * Inside the array is an object, as notated by the curly brackets, in which values of x and y are specified.
   * The x and y values are contained inside arrays as well.



**Plotly.js BAR Chart**

Text

Description automatically generated

* It makes sense to assign the data to a variable, as it would be very unwieldy to place an entire dataset inside the Plotly.newPlot() function call.
* The trace object now specifies the chart as a bar chart with type: "bar".

Another example:

Text

Description automatically generated

Note: The data object is no longer enclosed in an array in the function call, but it is still enclosed inside an array in the variable assignment.

**Layout Options**

Title:

Graphical user interface, text, application

Description automatically generated

Axis labels:

Graphical user interface, text, application

Description automatically generated

Another example:

var trace = {

x: ["nonalcoholic beer", "nonalcoholic wine", "nonalcoholic martini", "nonalcoholic margarita", "ice tea", "nonalcoholic rum & coke", "nonalcoholic mai tai", "nonalcoholic gin & tonic"],

y: [22.7, 17.1, 9.9, 8.7, 7.2, 6.1, 6.0, 4.6],

type: "bar"

};

var data = [trace];

var layout = {

title: "'Bar' Chart",

xaxis: {title: "Drinks"},

yaxis: {title: "% of Drinks Ordered"}

};

Plotly.newPlot("plotArea", data, layout);

Chart

Description automatically generated

**Plotly.js PIE Chart**

Graphical user interface, text, application

Description automatically generated

In trace, instead of x and y, the keys are labels and values.

Chart, pie chart

Description automatically generated

**DOCUMENTATION:** <https://plotly.com/javascript/basic-charts/>

**Array map() method:**

* The map method will iterate over each element in the array.
* Performs some logic on the element.
* And finally, returns a new array with the modify elements. (SAME # of elements as the original array)

Like a for loop, it can perform an operation to every element of an array.

var numbers = [1,2,3,4,5];

var doubled = numbers.map(function(num){

return num \* 2;

});

console.log(doubled);

**Note:** In the anonymous function inside the map() method, the parameter name **num** is arbitrary. It could have been named anything else.

**Note:** map() is not limited to numerical operations.

🡪 In this example, map() is used to extract a specific property from each object in an array (city name):

var cities = [

{

"Rank": 1,

"City": "San Antonio ",

"State": "Texas",

"Increase\_from\_2016": "24208",

"population": "1511946"

},

{

"Rank": 2,

"City": "Phoenix ",

"State": "Arizona",

"Increase\_from\_2016": "24036",

"population": "1626078"

},

{

"Rank": 3,

"City": "Dallas",

"State": "Texas",

"Increase\_from\_2016": "18935",

"population": "1341075"

}

];

var cityNames = cities.map(function(city){

return city.City;

});

console.log(cityNames);

🡪 In this example, map() is used to extract the population of each city:

var population = cities.map(function(popu){

return popu.City;

});

console.log(population);

BOTH examples on the console:

Text

Description automatically generated

**The filter() method:**

* It accepts another function as its parameter.
* filter() performs an operation on every element in the original array.
* filter() does not necessarily return an array whose length is the same as the original array.

EXAMPLE:

Text

Description automatically generated

* The **map()** method transforms every element of the original array, and so the size of the transformed array is the same as that of the original array.
* The **filter()** method returns an array of values that meet certain criteria. Values in the original array that do not fulfill the condition are filtered out.

Text

Description automatically generated Text

Description automatically generated

The anonymous function inside map() and filter() can be simplified as an arrow function:

var numbers = [1,2,3,4,5];

var doubled = numbers.map(num => num \* 2);

console.log(doubled);

**The sort() method:**

Example:

var familyAge = [3,2,39,37,9];

var sortedAge = familyAge.sort((a,b) => a - b);

console.log(sortedAge);

sortedAge returns the array [2,3,9,37,39].

* sort() takes in an anonymous function.
* During each iteration, the anonymous function, an arrow function in this case, compares one element of the array (a) with another element in the array (b). From a, it subtracts b.
  + If the result is negative (i.e., b is larger than a) then it stays put.
  + If the result of the subtraction is positive, the order of the two elements is reversed.

var familyAge = [3,2,39,37,9];

var sortedAge = familyAge.sort((anElement,anotherElement) => anElement -

anotherElement);

* The first two elements that are compared might be 3 and 2. The variable anElement is assigned to 3, and anotherElement to 2.
* The arrow function performs the subtraction anElement - anotherElement, or 3 - 2.
  + Since the result is positive (3 - 2 = 1), the order of the two numbers is reversed.
* The sort()method compares another pair of elements in the array, example 37 and 39.
  + Since 37 - 39 is a negative number, their ordering is kept.
* The process is repeated until the array is sorted.

Appending **reverse()** to the above sorts the array in descending order:

Graphical user interface, text

Description automatically generated with medium confidence

Another option is to type **b – a**:

var familyAge = [3,2,39,37,9];

var sortedAge = familyAge.sort((a,b) => b - a);

console.log(sortedAge);

Graphical user interface, text, application

Description automatically generated

**The slice() method:**

To select a subset of data.

var integers = [0,1,2,3,4,5];

var slice1 = integers.slice(0,2);

* The slice()method begins selecting the array at index position 0, and stops right before reaching index position 2.
  + Above, it returns elements at index positions 0 and 1, but not 2.
  + Below, it returns elements at index positions 0, 1 and 2, but not 3.

Graphical user interface, website

Description automatically generated

To slice to the end of an array, you can omit the second argument:

var words = ['seal', 'dog', 'scorpion', 'orangutan', 'salamander'];

words.slice(3, );

The elements sliced here are ['orangutan', 'salamander'].

**The reverse() method:**

It reverses the indexes of the array.

**Text

Description automatically generated**

In the console:

**A screenshot of a computer

Description automatically generated with medium confidence**

**Inspect an API call with D3.json()**

**JSON**, JavaScript Object Notation

* It is a data format that sorts and presents data in the form of key-value pairs.
* It looks much like a Python dictionary and can be traversed through using dot notation.

**D3** is primarily a data visualization library, but:

* its **d3.json()** method will allow to read external JSON files, as well as place calls to external web APIs for data.
  + The d3.json() returns a JavaScript promise:
    - It places an API call to the URL and executes subsequent lines of code only when the API call is complete.

**In order to LOAD the library to the HTML file:**

<script src="https://cdnjs.cloudflare.com/ajax/libs/d3/5.9.7/d3.min.js"></script>

Graphical user interface, text, application

Description automatically generated

Text

Description automatically generated

**The actual API call to SpaceX is made in these two lines of code:**

In the first line, the URL to the SpaceX is defined.

In the second line, d3.json() method places a call to SpaceX's API, retrieves the data, and prints it to the browser console:

* The d3.json() returns a JavaScript promise:
  + it places an API call to the URL and executes subsequent lines of code only when the API call is complete.
* Once the data is retrieved, it is assigned the *arbitrary* parameter name receivedData by the arrow function and printed in the console.
  + The d3.json().then() method ensures that the data is received before executing the arrow function.
* In summary, a JavaScript promise in this case waits for the data retrieval to finish before moving on to the next code.

const url = "https://api.spacexdata.com/v2/launchpads";

d3.json(url).then(receivedData => console.log(receivedData));

The code to retrieve the full\_name of the Vandenberg Air Force Base would look like this:

d3.json(url).then(spaceXResults => console.log(spaceXResults[0].full\_name));

**How to see the actual data:**

When we click the html file to open in the browser, nothing is printed to the console. We get this CORS error message:

Graphical user interface, application

Description automatically generated

The short explanation is that, for security reasons, a local server must be run when loading an external file into a JavaScript script file.

To load the page:

* Navigate to the directory where samples.json and index.html, as well as the script file, plots.js, are located.
* Open the command line (Terminal) and type the following:

python -m http.server

* You should see the following message in the command line:

Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...

137.0.0.1 - - [30/Oct/2019 13:23:53] "GET / HTTP/1.1" 304 -

* Navigate to the given port number in your browser: localhost:8000. 🡨 Use this as an URL
  + When you open the console now, you will see the data:
    - Then examine the results in the browser console

**! IMPORTANT**

When reading an external data file such as a CSV or JSON file into a script, you must run a server. You cannot directly open index.html with your browser. *(That’s why we go over the steps above)*

What we see in the console (in this example):

Text

Description automatically generated

The metadata array contains objects, each of which contains details of a volunteer, such as age, location, ethnicity, ID number, and weekly washing frequency of the belly button.

**If we want to extract only the wfreq of each person into a new array:**

Graphical user interface, text, application

Description automatically generated

d3.json("samples.json").then(function(data){  
    wfreq = data.metadata.map(person => person.wfreq);  
    console.log(wfreq);  
});

The map() method is used to extract the wfreq property from each “person” (arbitrary name) in the data.metadata array.

**If we want to sort the wfreq array in descending order:**

d3.json("samples.json").then(function(data){  
    wfreq = data.metadata.map(person =>  
person.wfreq).sort((a,b) => b - a);  
    console.log(wfreq);  
});

Notice that the sort() method returns b - a to return the results in descending order. It would return a - b to return the results in ascending order.

**To delete null values from the sorted wfreq array.**

d3.json("samples.json").then(function(data){  
    wfreq = data.metadata.map(person =>  
person.wfreq).sort((a,b) => b - a);  
    filteredWfreq = wfreq.filter(element => element !=  
null);  
    console.log(filteredWfreq);  
});

**object.entries() method**

The Object.entries() method allows access to both an object's keys and values.

* It returns each key-value pair as an array.
* If you are given the following object:

**researcher1** = {

name: 'Roza',

age: 34,

hobby: 'Hiking'

};

* Use Object.entries() to print each key-value pair inside an array:

console.log(Object.entries(researcher1));

* Result in the console:

Text

Description automatically generated

**forEach() method**

The forEach() method allows access to each element of an array.

This method iterates through each element in an array.

* In this case, there are smaller arrays, each including two elements, inside an outer array.
* To access these elements, the argument ([first, second]) is used, where first and second are arbitrarily chosen for convenience.
  + They could have been named ([x, y]) or ([key, value]).

researcher1.forEach(([first, second]) => console.log(first  
+ ": " + second));

**Parsing numbers as strings:**

Sometimes, we have arrays that include numbers but are passed as strings:

Text

Description automatically generated

In general, this could break when trying to operate with it.

We can parse it as an INT to make sure we are using it as a number:

parseInt()

Graphical user interface, text

Description automatically generated

**JavaScript Event Listeners**

**Creating a dropdown menu:**

The <select> tag is used to create a dropdown menu.

The <option> tag is used to create each menu option.

On the below HTML file:

* There are links to two CDNs: **D3** and **Plotly**.
* The **<select>** tag indicates a dropdown menu. Its id is "selectOption".
* The dropdown menu has two options, as indicated by the two **<option>** tags.
* The option values "option1" and "option2" are internal names for each dropdown menu option.
* First Option and Second Option are the text displayed in the browser for each menu option.
* A plc tag links to script.js, a JavaScript file.

Graphical user interface, text, application

Description automatically generated

On the script.js:

Graphical user interface, text, application

Description automatically generated

**The d3.selectAll() method:**

d3.selectAll().on("change", ); creates an event listener that triggers the custom function every time a change takes place to the HTML element specified by selectAll().

The d3.selectAll() method creates an event listener.

Whenever there is a change to the HTML body, the updatePage() function is called.

That is, when an event occurs on the page, such as selection of a dropdown menu option, the updatePage() function is triggered.

Text

Description automatically generated

**The Plotly.restyle() method:**

The Plotly.restyle() method is used to re-render the page on the browser.

This method is more efficient than calling the Plotly.newPlot() method, as it does not create a brand new chart from scratch, but instead **modifies the previously displayed chart with the updated information**.