Run this notebook with Data-Forge Notebook

JavaScript data wrangling cheat sheet

Snippets of JS code that are good for working with data.

From the book Data Wrangling with JavaScript

LOGGING

Logging is your best friend. Use `console.log` to display, inspect and check your data.

```
console.log("Your logging here"); // General text logging for debugging
```

```
Your logging here
```

```
const arr = [1, 2, 3];              // Your data.
console.log(arr);
```

```
[ 1, 2, 3 ]
```

```
const obj = { A: 1, B: 2, C: 3 }; // Your data
console.log(obj);
```

```
{ A: 1, B: 2, C: 3 }
```

In Data-Forge Notebook you can also use the `display` function for formatted output:

```
const obj = { A: 1, B: 2, C: 3 }; // Your data

display(obj);
```

```
"root" : {  3 items
  "A" : 1
  "B" : 2
  "C" : 3
}
```

OBJECTS

Techniques for creating and modifying JavaScript objects.

Extract a field

```
let o = { A: 1, B: 2 };                 // Your data
let v1 = o["A"];                        // Extract field value
display(v1);
```

```
let v2 = o.A;
display(v2);
```

```
1
```

```
1
```

Set a field

```
let o = {};                              // Empty object
o["A"] = 3;                              // Set field value
o.A = 3;

display(o);
```

```
"root" : {  1 item
  "A" : 3
}
```

Delete a field

```
let o = { A: 1, B: 2 };
delete o["A"];                           // Delete a field value
delete o.A;

display(o);
```

```
"root" : {  1 item
  "B" : 2
}
```

Clone an object

```
let o = { A: 1, B: 2 };
let c = Object.assign({}, o);            // Clone an object
c.A = 300;
c.B = 500;

display(o);                              // Original object is unchangec
display(c);                              // Cloned object is modified
```

```
"root" : {  2 items
  "A" : 1
  "B" : 2
}
```

```
"root" : {  2 items
  "A" : 300
  "B" : 500
}
```

Replace fields in an object

```
let o = { A: 1, B: 2 };
let ovr = { B: 200 };
let c = Object.assign({}, o, ovr);       // Clone and override fields

display(o);                              // Original object is unchangec
display(c);                              // Cloned object has specified
```

```
"root" : {  2 items
  "A" : 1
  "B" : 2
}
```

```
"root" : {  2 items
  "A" : 1
  "B" : 200
}
```

ARRAYS

Techniques for creating and modifying JavaScript arrays.

Visit each item

```
let a = [1, 2, 3];                       // Your data
a.forEach(item => {
    console.log(item);                   // Visit each item in the array
});

// Or (old-style JS)
for (let i = 0; i < a.length; ++i) {
    const item = a[i];
    // Visit each item
}

// Or (using modern JS iterators)
for (const item of a) {
    // Visit each item
}
```

```
1
2
3
```

Getting and setting values

```
let a = [1, 2, 3, 4, 5, 6];              // Your data
let v = a[5];                            // Get value at index
display(v);
```

```
a[3] = 32;                                  // Set value at index
display(a);
```

```
6
```

```
"root" : [  6 items
  0 : 1
  1 : 2
  2 : 3
  3 : 32
  4 : 5
  5 : 6
]
```

Adding and removing items

```
let a = [1, 2, 3];

a.push("new end item");                     // Add to end of array
display(a);

let last = a.pop();                         // Remove last element
display(last);
display(a);

a.unshift("new start item");                // Add to start of array
display(a);

let first = a.shift();                      // Remove first element
display(first);
display(a);
```

```
"root" : [  4 items
  0 : 1
  1 : 2
  2 : 3
  3 : "new end item"
]
```

```
new end item
```

```
"root" : [  3 items
  0 : 1
  1 : 2
  2 : 3
]
```

```
"root" : [  4 items
  0 : "new start item"
  1 : 1
  2 : 2
  3 : 3
]
```

```
new start item
```

```
"root" : [  3 items
  0 : 1
  1 : 2
  2 : 3
]
```

Concatenate arrays

```
let a1 = [1, 2, 3];
let a2 = [4, 5, 6];
let a = a1.concat(a2);                    // Concatenate arrays
display(a);
```

```
 "root" : [  6 items
   0 : 1
   1 : 2
   2 : 3
   3 : 4
   4 : 5
   5 : 6
]
```

Extracting portions of an array

```
let a = [1, 2, 3, 4, 5];

let e = a.slice(0, 3);                        // Extract first 3 elements
display(e);
```

```
 "root" : [  3 items
   0 : 1
   1 : 2
   2 : 3
]
```

```
let a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13];

let e = a.slice(5, 11);                       // Extract elements 5 to 10
display(e);
```

```
 "root" : [  6 items
   0 : 6
   1 : 7
   2 : 8
   3 : 9
   4 : 10
   5 : 11
]
```

```
let a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

let e = a.slice(-4, -1);                      // Negative indicies relative t
display(e);
```

```
 "root" : [  3 items
   0 : 7
```

```
    1 : 8
    2 : 9
]
```

```
let a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

let e = a.slice(-3);                    // Extract last three elements
display(e);
```

```
  "root" : [  3 items
    0 : 8
    1 : 9
    2 : 10
]
```

Clone an array

```
let a = [1, 2, 3, 4, 5];
let c = a.slice();                      // Clone array
c[2] = 2230;
display(a);                             // Original array is unchanged
display(c);                             // Cloned array is modified
```

```
  "root" : [  5 items
    0 : 1
    1 : 2
    2 : 3
    3 : 4
    4 : 5
]
```

```
  "root" : [  5 items
    0 : 1
    1 : 2
    2 : 2230
    3 : 4
    4 : 5
]
```

Find an element in an array

```
let a = [1, 2, 3, 4, 5];
let i = a.indexOf(3);                   // Find index of item in array
if (i >= 0) {
    let v = a[i];                       // The value exists, extract it
    display(v);
}
```

```
3
```

Sorting an array

```
let a = ["Pineapple", "Orange", "Apple", "Bananna"];
a.sort();
display(a);
```

```
"root" : [  4 items
  0 : "Apple"
  1 : "Bananna"
  2 : "Orange"
  3 : "Pineapple"
]
```

```
let a = ["Pineapple", "Orange", "Apple", "Bananna"];
let c = a.slice();                       // Clone the original
c.sort();                                // Sort array without modifying
display(a);                              // Original array is unmodified
display(c);                              // Cloned array is sorted
```

```
"root" : [  4 items
  0 : "Pineapple"
  1 : "Orange"
  2 : "Apple"
  3 : "Bananna"
]
"root" : [  4 items
  0 : "Apple"
  1 : "Bananna"
  2 : "Orange"
  3 : "Pineapple"
]
```

```
let a = [10, 20, 8, 15, 12, 33];
a.sort((a, b) => b - a);                 // Customize sort with a user-c
display(a);
```

```
"root" : [  6 items
  0 : 33
  1 : 20
  2 : 15
  3 : 12
  4 : 10
  5 : 8
]
```

FUNCTIONAL JAVASCRIPT

Functional-style array manipulation techniques.

Filter

Filter an array with `filter` and a user-defined predicate function.

```
let a = [10, 20, 8, 15, 12, 33];

function predicate(value) {
    return value > 10;                  // Retain values > 10
}

let f = a.filter(v => predicate(v));    // Filter array
display(f);
```

```
"root" : [  4 items
    0 : 20
    1 : 15
    2 : 12
    3 : 33
]
```

Transform

Transform an array with map and a user-defined transformation function.

```
let a = [1, 2, 3, 4, 5];

function transform(value) {
    return value + 1;                   // Increment all values by one.
}

let t = a.map(v => transform(v));       // Transform array
display(t);
```

```
"root" : [  5 items
    0 : 2
    1 : 3
    2 : 4
    3 : 5
    4 : 6
]
```

Aggregation

Aggregate an array with reduce and a user-defined aggregation function.

```
let a = [1, 2, 3, 4, 5];

function sum(a, b) {
    return a + b;                       // Produces the sum of all valu
}

let t = a.reduce(sum, 0)  // Reduce the array by summing the total of a
display(t);
```

```
15
```

REGULAR EXPRESSIONS

Use regular expressions to match and extract search patterns in text.

```
let re = /search pattern/;                    // Define regular expression

// Or
re = new RegExp("search pattern");

// Or add options
re = /search pattern/ig                       // Case insensitive + global

let source = "your text data that contains the search pattern";
let match = re.exec(source);                 // Find first match.
display(match);

while ((match = re.exec(source)) !== null) {
    // Find each  match in turn.
}
```

```
"root" : [ 1 item
  0 : "search pattern"
]
```

READ AND WRITE TEXT FILES

In Node.js we can read and write text files using the fs module functions `fs.readFileSync` and `fs.writeFileSync`.

After you run this code cell, check out the contents of the file `my-text-file.txt` that has been written out to your file system.

```
const fs = require('fs');

const textData = "My text data";
fs.writeFileSync("./my-text-file.txt", textData);

const loadedTextData = fs.readFileSync("./my-text-file.txt", "utf8");
display(loadedTextData);
```

```
My text data
```

DATA FORMATS

Serialize and deserialize JSON data

JavaScript already contains the functions you need to to serialize and deserialize data to and from the JSON format.

Use `JSON.stringify` to convert your data to JSON, then use `JSON.parse` to convert it back.

```
const data = [
    { item: "1" },
    { item: "2" },
    { item: "3" }
];
const jsonData = JSON.stringify(data);          // Serialize (encode) t

display(jsonData);

const deserialized = JSON.parse(jsonData);      // Deserialize (decode)
display(deserialized);
```

```
[{"item":"1"},{"item":"2"},{"item":"3"}]
```

```
  "root" : [  3 items
      0 : {  1 item
        "item" : "1"
      }
      1 : {  1 item
        "item" : "2"
      }
      2 : {  1 item
        "item" : "3"
      }
]
```

Read and write JSON data files

If we combine the `fs` functions with the `JSON` functions we can now read and write JSON data files.

After you run this code cell, check out the contents of the file `my-json-file.json` that has been written out to your file system.

```
const fs = require('fs');

const data = [
    { item: "1" },
    { item: "2" },
    { item: "3" }
];
fs.writeFileSync("./my-json-file.json", JSON.stringify(data));

const deserialized = JSON.parse(fs.readFileSync("./my-json-file.json",
display(deserialized);
```

```
  "root" : [  3 items
      0 : {  1 item
        "item" : "1"
      }
      1 : {  1 item
        "item" : "2"
      }
      2 : {  1 item
```

```
        "item" : "3"
    }
]
```

Serialize and deserialize CSV data

Let's not forget about working with CSV data, it's a staple of the data science community!

Unfortunately JavaScript doesn't provide us with functions to do this, so we'll turn to the excellent PapaParse library available via npm.

Note the use of the `dynamicTyping` option - this is quite important as it causes PapaParse to deserialized CSV columns that contain numbers and booleans (unfortunately it doesn't help with dates).

```
const Papa = require('papaparse');

const data = [
    { item: "1", val: 100 },
    { item: "2", val: 200 },
    { item: "3", val: 300 }
];
const csvData = Papa.unparse(data);                        // Serialize (e

display(csvData);

const options = { dynamicTyping: true, header: true };
const deserialized = Papa.parse(csvData, options);      // Deserialize
display(deserialized.data);
```

```
item,val 1,100 2,200 3,300
```

```
"root" : [ 3 items
    0 : { 2 items
        "item" : 1
        "val" : 100
    }
    1 : { 2 items
        "item" : 2
        "val" : 200
    }
    2 : { 2 items
        "item" : 3
        "val" : 300
    }
]
```

Read and write CSV data files

We can also combine the `fs` functions with PapaParse and be able to read and write CSV data files.

After you run this code cell, check out the contents of the file `my-csv-file.csv` that has been written out to your file system.

```
const fs = require('fs');
const Papa = require('papaparse');

const data = [
    { item: "1", val: 100 },
    { item: "2", val: 200 },
    { item: "3", val: 300 }
];
fs.writeFileSync("./my-csv-file.csv", Papa.unparse(data));

const options = { dynamicTyping: true, header: true };
const deserialized = Papa.parse(fs.readFileSync("./my-csv-file.csv", "u
display(deserialized.data);
```

```
"root" : [  3 items
    0 : {  2 items
        "item" : 1
        "val" : 100
    }
    1 : {  2 items
        "item" : 2
        "val" : 200
    }
    2 : {  2 items
        "item" : 3
        "val" : 300
    }
]
```

Getting data from a REST API

Use the `axios` module to retreive data from a REST API (with data from JSONPlaceholder).

```
const axios = require('axios');
const response = await axios("https://jsonplaceholder.typicode.com/todo
const data = response.data;
display(data.slice(0, 5));
```

```
"root" : [  5 items
    0 : {  4 items
        "userId" : 1
        "id" : 1
        "title" : "delectus aut autem"
        "completed" : false
    }
    1 : {  4 items
        "userId" : 1
        "id" : 2
        "title" : "quis ut nam facilis et officia qui"
        "completed" : false
    }
    2 : {  4 items
```

```
        "userId" : 1
        "id" : 3
        "title" : "fugiat veniam minus"
        "completed" : false
    }
    3 : {  4 items
        "userId" : 1
        "id" : 4
        "title" : "et porro tempora"
        "completed" : true
    }
    4 : {  4 items
        "userId" : 1
        "id" : 5
        "title" : "laboriosam mollitia et enim quasi adipisci quia provident illum"
        "completed" : false
    }
]
```