Run this code using 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. It's the easiest way to 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);
```

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);
```

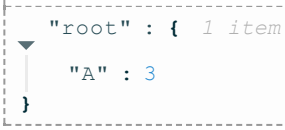
```
let v2 = o.A;
display(v2);
```

```
1
```

```
1
```

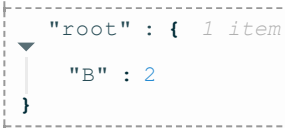
Set a field

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

display(o);
```



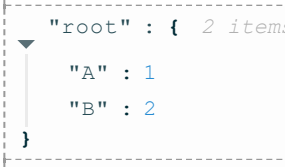
Delete a field

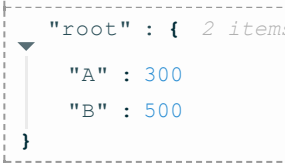
```
let o = { A: 1, B: 2 };
delete o["A"];                                  // Delete a field value
delete o.A;

display(o);
```



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 unchanged
display(c);                                     // Cloned object is modified
```





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 unchanged
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);
```

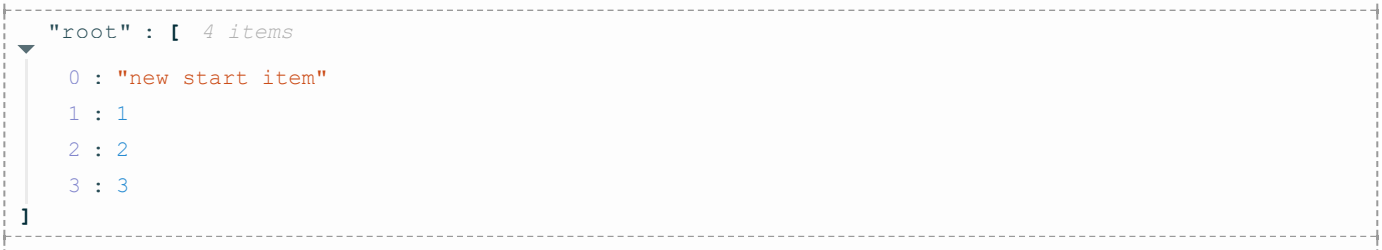
```
a[3] = 32;                                  // Set value at index
display(a);
```

```
6
```

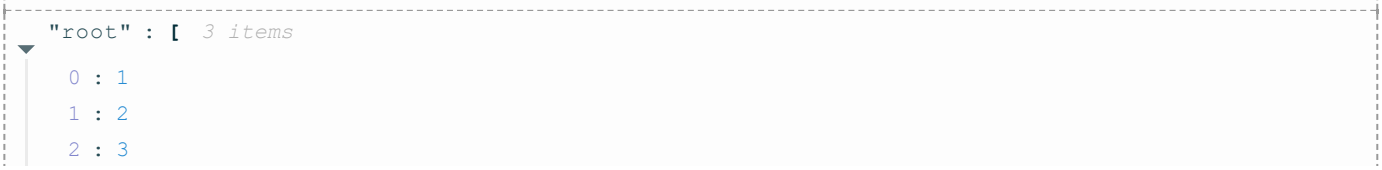
Adding and removing items

```
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);
```


```
new end item
```




```
new start item
```



```
]
```

Concatenate arrays

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

Extracting portions of an array

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

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


```
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);
```


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

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

```
    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);
```


```
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
```


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

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);
```

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);
```

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"
]
```