

Why Loop in JavaScript When You Can Map, Filter or Reduce?

Scott McAllister
@stmcallister

Thought of the Day

"For loops are broken"

```

function incrementYear(sheetId) {
  // get sheet
  smartsheet.sheets.getSheet({ id: sheetId }).then(function(sheet) {
    // for each column title listed in the dateColumnNames array, get the columnId for that column and add it to the dateColumnIds array.
    for (var y = 0; y < dateColumnNames.length; y++) {
      for (var i = 0; i < sheet.columns.length; i++) {
        if (sheet.columns[i].title == dateColumnNames[y]) {
          dateColumnIds.push(sheet.columns[i].id);
        }
      }
    }

    // loop the rows in the sheet
    for (var r = 0; r < sheet.rows.length; r++) {
      var row = {
        "id": sheet.rows[r].id,
        "cells": []
      };

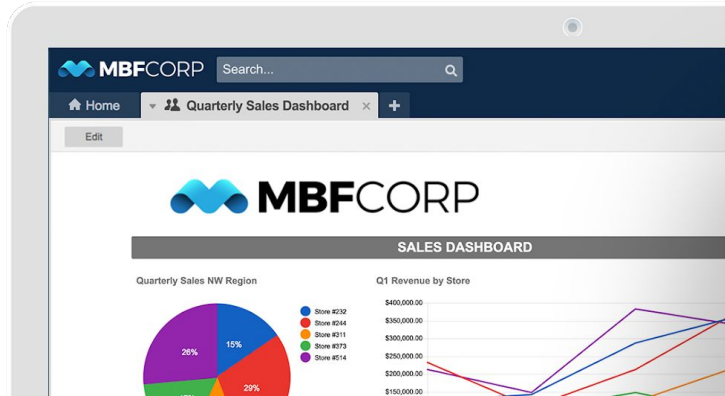
      // loop the cells in the row
      for (var c = 0; c < sheet.rows[r].cells.length; c++) {
        for (var x = 0; x < dateColumnIds.length; x++) {
          // for each cell with a columnId that matches a dateColumnId check the value of the date
          if (sheet.rows[r].cells[c].columnId == dateColumnIds[x]) {
            var columnValueDate = new Date(sheet.rows[r].cells[c].value);

            // if the date value is in the past, increment the year
            if (columnValueDate < new Date()) {
              // set the value to new year
              var newDate = new Date(columnValueDate.setFullYear(columnValueDate.getFullYear() + 1));
              // create a cell object for updating the sheet
              var cell = {
                "columnId": sheet.rows[r].cells[c].columnId,
                "value": newDate
              };
              // add the cell object to updated row object
              row.cells.push(cell);
            }
          }
        }
      }

      // if the row contains cells add the row to the rowsToUpdate array
      if (row.cells.length > 0) {
        rowsToUpdate.push(row);
      }
    }
  });
}

```

Smartsheet



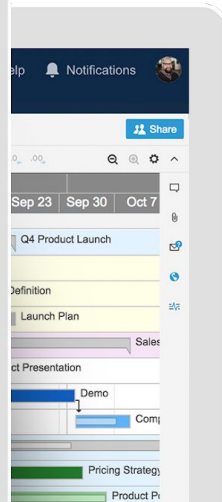
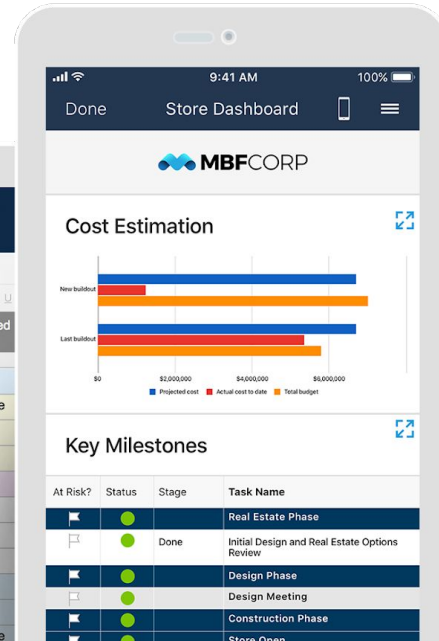
MBFCORP

Quarterly Sales Dashboard

File Alerts & Actions Forms

Gantt View Filter Arial 10 B I U

	At R...	R...	Task Name	Assigned To
1			Q4 Product Launch	
2			Planning Phase	Therese
3			Market Reqs Definition	Joe
4			Launch Plan	Brian
5			Sales Tools	
6			Prospect Presentation	Joe
7			Demo	Jenny
8			Competitive Positioning	Shawn
9			Product Marketing	
10			Pricing Strategy	
11			Product Positioning	Therese



The Clan McAllister



smartsheet Search... ? Help Notifications

Home Crossover Twilio Roster Twilio Log QRScanner Family Birthdays Demo x

File Alerts & Actions Forms Share

Grid View Filter Arial 10 B I U S A

			Name	Birthday
1			McAllister Family Contact	
2				
3			Mom & Dad	
4			Mom	06/21/18
5			Dad	01/13/18
6			Bryan & JoAnna	
7			Bryan	06/17/18
8			JoAnna	05/09/18
9			Theo	03/10/18
10			Max	09/27/18
11			Lainey	03/12/18
12			Evelyn	05/04/18
13			Julia and Dale	
14			Dale	09/27/18
15			Julia	04/04/18
16			Ben	03/10/18
17			Jacob	03/30/18
18			Emma	05/11/18
19			Joseph	02/18/18

Birthday Sheet

```

function incrementYear(sheetId) {
  // get sheet
  smartsheet.sheets.getSheet({ id: sheetId }).then(function(sheet) {
    // for each column title listed in the dateColumnNames array, get the columnId for that column and add it to the dateColumnIds array.
    for (var y = 0; y < dateColumnNames.length; y++) {
      for (var i = 0; i < sheet.columns.length; i++) {
        if (sheet.columns[i].title == dateColumnNames[y]) {
          dateColumnIds.push(sheet.columns[i].id);
        }
      }
    }

    // loop the rows in the sheet
    for (var r = 0; r < sheet.rows.length; r++) {
      var row = {
        "id": sheet.rows[r].id,
        "cells": []
      };

      // loop the cells in the row
      for (var c = 0; c < sheet.rows[r].cells.length; c++) {
        for (var x = 0; x < dateColumnIds.length; x++) {
          // for each cell with a columnId that matches a dateColumnId check the value of the date
          if (sheet.rows[r].cells[c].columnId == dateColumnIds[x]) {
            var columnValueDate = new Date(sheet.rows[r].cells[c].value);

            // if the date value is in the past, increment the year
            if (columnValueDate < new Date()) {
              // set the value to new year
              var newDate = new Date(columnValueDate.setFullYear(columnValueDate.getFullYear() + 1));
              // create a cell object for updating the sheet
              var cell = {
                "columnId": sheet.rows[r].cells[c].columnId,
                "value": newDate
              };
              // add the cell object to updated row object
              row.cells.push(cell);
            }
          }
        }
      }

      // if the row contains cells add the row to the rowsToUpdate array
      if (row.cells.length > 0) {
        rowsToUpdate.push(row);
      }
    }
  });
}

```

Problems with this code

- Complex
- Hard to read
- Lots generic code that is not reusable

JavaScript Tools That Can Help

Functional programming can make some of this pain go away

Let's look at three basic functions that can help us today

- `filter()`
- `map()`
- `reduce()`

Why Functional?

"As software becomes more and more complex, it is more and more important to structure it well. Well-structured software is easy to write and to debug, and provides a collection of modules that can be reused to reduce future programming costs."

-- John Hughes, *Why Functional Programming Matters*

Dissection of a For Loop

```
let numbers = [1,2,3,4,5,6];  
let bigNums = [];  
for (let i = 0; i < numbers.length; i++) {  
    if (numbers[i] > 5) {  
        bigNums.push(numbers[i]);  
    }  
}
```

Dissection of a For Loop

```
let numbers = [1,2,3,4,5,6];  
let bigNums = [];  
for (let i = 0; i < numbers.length; i++) {  
  if (numbers[i] > 5) {  
    bigNums.push(numbers[i]);  
  }  
}
```

Dissection of a For Loop

```
let numbers = [1,2,3,4,5,6];  
let bigNums = [];  
for (let i = 0; i < numbers.length; i++) {  
  if (numbers[i] > 5) {  
    bigNums.push(numbers[i]);  
  }  
}
```

```
let bigNums = numbers.filter((number) => number > 5);
```

Couple of Concepts

- Higher-order Functions
- Arrow Functions

Higher-order Functions

"A higher-order function is a function that takes one or more functions as arguments."

-- Closurebridge

Higher-order Functions

"A higher-order function is a function that takes one or more functions as arguments."

-- Closurebridge

Arrow Functions

- Added in ES6
- Shortens function syntax
- Does not have its own `this`

Arrow Functions

```
let bigNums = numbers.filter(function (number) {  
    return number > 5;  
});
```

Arrow Functions

```
let bigNums = numbers.filter(function (number) {  
    return number > 5;  
});
```

```
let bigNums = numbers.filter((number) => number > 5);
```

Arrow Functions

```
let bigNums = numbers.filter(function (number) {  
    return number > 5;  
});
```

```
let bigNums = numbers.filter((number) => number > 5);
```

```
let bigNums = numbers.filter(number => number > 5);
```

Array.filter()

```
let numbers = [1,2,3,4,5,6];  
let bigNums = [];  
for (let i = 0; i < numbers.length; i++) {  
  if (numbers[i] > 5) {  
    bigNums.push(numbers[i]);  
  }  
}
```

```
let bigNums = numbers.filter((number) => number > 5);
```

Array.filter()

Creates a new array with all elements that pass the test implemented by the provided function

Array.map()

Creates a new array from the results of performing an action on every element in the calling array

Array.map()

```
const nums = [1,2,3,4,5];  
  
let dubs = [];  
  
for (let i = 0; i < nums.length; i++) {  
    dubs.push(nums[i] * 2);  
}
```


Array.map()

```
const nums = [1,2,3,4,5];  
  
let dubs = [];  
  
for (let i = 0; i < nums.length; i++) {  
    dubs.push(nums[i] * 2);  
}
```

Array.map()

```
const nums = [1,2,3,4,5];  
  
let dubs = [];  
  
for (let i = 0; i < nums.length; i++) {  
  dubs.push(nums[i] * 2);  
}
```

```
const dubs = nums.map((num) => num * 2);
```

Array.map()

```
const nums = [1,2,3,4,5];  
  
let dubs = [];  
  
for (let i = 0; i < nums.length; i++) {  
  dubs.push(nums[i] * 2);  
}
```

```
const dubs = nums.map((num) => num * 2);
```

Array.reduce()

Applies a function against an accumulator and each element in the array (from left to right) to reduce it to a single value.

Array.reduce()

```
const nums = [1,2,3,4,5];  
  
let sum = 0;  
  
for (let i = 0; i < nums.length; i++) {  
    sum = sum + nums[i];  
}
```

Array.reduce()

```
const nums = [1,2,3,4,5];
```

```
let sum = 0;
```

```
for (let i = 0; i < nums.length; i++) {
```

```
    sum = sum + nums[i];
```

```
}
```

Array.reduce()

```
const nums = [1,2,3,4,5];
```

```
let sum = 0;
```

```
for (let i = 0; i < nums.length; i++) {
```

```
    sum = sum + nums[i];
```

```
}
```

```
const sum = nums.reduce((total, num) => total + num, 0);
```

Array.reduce()

```
const nums = [1,2,3,4,5];
```

```
let sum = 0;
```

```
for (let i = 0; i < nums.length; i++) {
```

```
    sum = sum + nums[i];
```

```
}
```

```
const sum = nums.reduce((total, num) => total + num, 0);
```



```

14 // for each column title listed in the dateColumnNames array, get the column
15 for (var i = 0; i < sheet.columns.length; i++) {
16     if (sheet.columns[i].title === dateColumnName) {
17         dateColumnId = sheet.columns[i].id;
18         break;
19     }
20 }
21 // loop the rows in the sheet
22 for (var r = 0; r < sheet.rows.length; r++) {
23     var row = {
24         "id": sheet.rows[r].id,
25         "cells": []
26     };
27     // loop the cells in the row
28     for (var c = 0; c < sheet.rows[r].cells.length; c++) {
29         // for each cell with a columnId that matches a dateColumnId check
30         if (sheet.rows[r].cells[c].columnId === dateColumnId) {
31             var columnValueDate = new Date(sheet.rows[r].cells[c].value);
32
33             // if the date value is in the past, increment the year
34             if (columnValueDate < weekAgo) {
35                 // set the value to new year
36                 var newDate = new Date(columnValueDate.setFullYear(columnValueDate.getFullYear() + 1));
37                 // create a cell object for updating the sheet
38                 var cell = {
39                     "columnId": sheet.rows[r].cells[c].columnId,
40                     "value": newDate
41                 };
42                 // add the cell object to updated row object
43                 row.cells.push(cell);
44             }
45         }
46     }
47     // if the row contains cells add the row to the rowsToUpdate array
48     if (row.cells.length > 0) {
49         rowsToUpdate.push(row);
50     }

```

```

15 // for each column title listed in the dateColumnNames array, get the column
16 sheet.columns
17     .filter((column) => dateColumnName === column.title)
18     .map((column) => dateColumnId = column.id);
19
20 const rowsToUpdate = sheet.rows.map((row) => {
21     return {
22         id: row.id,
23         cells: row.cells
24             .filter((cell) => dateColumnId === cell.columnId &&
25                 new Date(cell.value) < weekAgo)
26             .map((cell) => {
27                 var columnValueDate = new Date(cell.value);
28                 cell.value = new Date(columnValueDate.setFullYear(columnValueDate.getFullYear() + 1));
29                 return cell;
30             })
31     };
32 }).filter((row) => row.cells && row.cells.length > 0);
33 // build options object to send as part of the update request for smartsheet
34 const options = {
35     "sheetId": sheetId,
36     "body": rowsToUpdate
37 };
38 // send updateRow request to Smartsheet
39 smartsheet.sheets.updateRow(options).then(function (data) {
40     console.log('updateRow: ' + JSON.stringify(data));
41 })
42 .catch(function (error) {
43     console.log(error);
44 });
45 ;
46
47
48 cute script
49 entYear(sheetId);
50

```

```

14 // for each column title listed in the dateColumnNames array, get the column
15 for (var i = 0; i < sheet.columns.length; i++) {
16     if (sheet.columns[i].title === dateColumnName) {
17         dateColumnId = sheet.columns[i].id;
18         break;
19     }
20 }
21 // loop the rows in the sheet
22 for (var r = 0; r < sheet.rows.length; r++) {
23     var row = {
24         "id": sheet.rows[r].id,
25         "cells": []
26     };
27     // loop the cells in the row
28     for (var c = 0; c < sheet.rows[r].cells.length; c++) {
29         // for each cell with a columnId that matches a dateColumnId check
30         if (sheet.rows[r].cells[c].columnId === dateColumnId) {
31             var columnValueDate = new Date(sheet.rows[r].cells[c].value);
32
33             // if the date value is in the past, increment the year
34             if (columnValueDate < weekAgo) {
35                 // set the value to new year
36                 var newDate = new Date(columnValueDate.setFullYear(columnValueDate.getFullYear() + 1));
37                 // create a cell object for updating the sheet
38                 var cell = {
39                     "columnId": sheet.rows[r].cells[c].columnId,
40                     "value": newDate
41                 };
42                 // add the cell object to updated row object
43                 row.cells.push(cell);
44             }
45         }
46     }
47     // if the row contains cells add the row to the rowsToUpdate array
48     if (row.cells.length > 0) {
49         rowsToUpdate.push(row);
50     }

```

```

15 // for each column title listed in the dateColumnNames array, get the column
16 sheet.columns
17     .filter((column) => dateColumnName === column.title)
18     .map((column) => dateColumnId = column.id);
19
20 const rowsToUpdate = sheet.rows.map((row) => {
21     return {
22         id: row.id,
23         cells: row.cells
24             .filter((cell) => dateColumnId === cell.columnId &&
25                 new Date(cell.value) < weekAgo)
26             .map((cell) => {
27                 var columnValueDate = new Date(cell.value);
28                 cell.value = new Date(columnValueDate.setFullYear(columnValueDate.getFullYear() + 1));
29                 return cell;
30             })
31     };
32 }).filter((row) => row.cells && row.cells.length > 0);
33 // build options object to send as part of the update request for smartsheet
34 const options = {
35     "sheetId": sheetId,
36     "body": rowsToUpdate
37 };
38 // send updateRow request to Smartsheet
39 smartsheet.sheets.updateRow(options).then(function (data) {
40     console.log('updateRow: ' + JSON.stringify(data));
41 })
42 .catch(function (error) {
43     console.log(error);
44 });
45
46
47
48 cute script
49 entYear(sheetId);
50

```

```

14 // for each column title listed in the dateColumnNames array, get the column
15 for (var i = 0; i < sheet.columns.length; i++) {
16     if (sheet.columns[i].title === dateColumnName) {
17         dateColumnId = sheet.columns[i].id;
18         break;
19     }
20 }
21 // loop the rows in the sheet
22 for (var r = 0; r < sheet.rows.length; r++) {
23     var row = {
24         "id": sheet.rows[r].id,
25         "cells": []
26     };
27     // loop the cells in the row
28     for (var c = 0; c < sheet.rows[r].cells.length; c++) {
29         // for each cell with a columnId that matches a dateColumnId check
30         if (sheet.rows[r].cells[c].columnId === dateColumnId) {
31             var columnValueDate = new Date(sheet.rows[r].cells[c].value);
32
33             // if the date value is in the past, increment the year
34             if (columnValueDate < weekAgo) {
35                 // set the value to new year
36                 var newDate = new Date(columnValueDate.setFullYear(columnValueDate.getFullYear() + 1));
37                 // create a cell object for updating the sheet
38                 var cell = {
39                     "columnId": sheet.rows[r].cells[c].columnId,
40                     "value": newDate
41                 };
42                 // add the cell object to updated row object
43                 row.cells.push(cell);
44             }
45         }
46     }
47     // if the row contains cells add the row to the rowsToUpdate array
48     if (row.cells.length > 0) {
49         rowsToUpdate.push(row);
50     }

```

```

15 // for each column title listed in the dateColumnNames array, get the column
16 sheet.columns
17     .filter((column) => dateColumnName === column.title)
18     .map((column) => dateColumnId = column.id);
19
20 const rowsToUpdate = sheet.rows.map((row) => {
21     return {
22         id: row.id,
23         cells: row.cells
24             .filter((cell) => dateColumnId === cell.columnId &&
25                 new Date(cell.value) < weekAgo)
26             .map((cell) => {
27                 var columnValueDate = new Date(cell.value);
28                 cell.value = new Date(columnValueDate.setFullYear(columnValueDate.getFullYear() + 1));
29                 return cell;
30             })
31     };
32 }).filter((row) => row.cells && row.cells.length > 0);
33
34 // build options object to send as part of the update request for Smartsheet
35 const options = {
36     "sheetId": sheetId,
37     "body": rowsToUpdate
38 };
39 // send updateRow request to Smartsheet
40 smartsheet.sheets.updateRow(options).then(function (data) {
41     console.log('updateRow: ' + JSON.stringify(data));
42 })
43 .catch(function (error) {
44     console.log(error);
45 });
46
47
48 // cute script
49 entYear(sheetId);
50

```



```

14 // for each column title listed in the dateColumnNames array, get the column
15 for (var i = 0; i < sheet.columns.length; i++) {
16     if (sheet.columns[i].title === dateColumnName) {
17         dateColumnId = sheet.columns[i].id;
18         break;
19     }
20 }
21 // loop the rows in the sheet
22 for (var r = 0; r < sheet.rows.length; r++) {
23     var row = {
24         "id": sheet.rows[r].id,
25         "cells": []
26     };
27     // loop the cells in the row
28     for (var c = 0; c < sheet.rows[r].cells.length; c++) {
29         // for each cell with a columnId that matches a dateColumnId check
30         if (sheet.rows[r].cells[c].columnId === dateColumnId) {
31             var columnValueDate = new Date(sheet.rows[r].cells[c].value);
32
33             // if the date value is in the past, increment the year
34             if (columnValueDate < weekAgo) {
35                 // set the value to new year
36                 var newDate = new Date(columnValueDate.setFullYear(columnValueDate.getFullYear() + 1));
37                 // create a cell object for updating the sheet
38                 var cell = {
39                     "columnId": sheet.rows[r].cells[c].columnId,
40                     "value": newDate
41                 };
42                 // add the cell object to updated row object
43                 row.cells.push(cell);
44             }
45         }
46     }
47     // if the row contains cells add the row to the rowsToUpdate array
48     if (row.cells.length > 0) {
49         rowsToUpdate.push(row);
50     }

```

```

15 // for each column title listed in the dateColumnNames array, get the column
16 sheet.columns
17     .filter((column) => dateColumnName === column.title)
18     .map((column) => dateColumnId = column.id);
19
20 const rowsToUpdate = sheet.rows.map((row) => {
21     return {
22         id: row.id,
23         cells: row.cells
24             .filter((cell) => dateColumnId === cell.columnId &&
25                 new Date(cell.value) < weekAgo)
26             .map((cell) => {
27                 var columnValueDate = new Date(cell.value);
28                 cell.value = new Date(columnValueDate.setFullYear(columnValueDate.getFullYear() + 1));
29                 return cell;
30             })
31     };
32 }).filter((row) => row.cells && row.cells.length > 0);
33 // build options object to send as part of the update request for smartsheet
34 const options = {
35     "sheetId": sheetId,
36     "body": rowsToUpdate
37 };
38 // send updateRow request to Smartsheet
39 smartsheet.sheets.updateRow(options).then(function (data) {
40     console.log('updateRow: ' + JSON.stringify(data));
41 })
42 .catch(function (error) {
43     console.log(error);
44 });
45 ;
46
47
48 cute script
49 entYear(sheetId);
50

```

```

14 // for each column title listed in the dateColumnNames array, get the column
15 for (var i = 0; i < sheet.columns.length; i++) {
16     if (sheet.columns[i].title === dateColumnName) {
17         dateColumnId = sheet.columns[i].id;
18         break;
19     }
20 }
21 // loop the rows in the sheet
22 for (var r = 0; r < sheet.rows.length; r++) {
23     var row = {
24         "id": sheet.rows[r].id,
25         "cells": []
26     };
27     // loop the cells in the row
28     for (var c = 0; c < sheet.rows[r].cells.length; c++) {
29         // for each cell with a columnId that matches a dateColumnId check
30         if (sheet.rows[r].cells[c].columnId === dateColumnId) {
31             var columnValueDate = new Date(sheet.rows[r].cells[c].value);
32
33             // if the date value is in the past, increment the year
34             if (columnValueDate < weekAgo) {
35                 // set the value to new year
36                 var newDate = new Date(columnValueDate.setFullYear(columnValueDate.getFullYear() + 1));
37                 // create a cell object for updating the sheet
38                 var cell = {
39                     "columnId": sheet.rows[r].cells[c].columnId,
40                     "value": newDate
41                 };
42                 // add the cell object to updated row object
43                 row.cells.push(cell);
44             }
45         }
46     }
47     // if the row contains cells add the row to the rowsToUpdate array
48     if (row.cells.length > 0) {
49         rowsToUpdate.push(row);
50     }

```

```

15 // for each column title listed in the dateColumnNames array, get the column
16 sheet.columns
17     .filter((column) => dateColumnName === column.title)
18     .map((column) => dateColumnId = column.id);
19
20 const rowsToUpdate = sheet.rows.map((row) => {
21     return {
22         id: row.id,
23         cells: row.cells
24             .filter((cell) => dateColumnId === cell.columnId &&
25                 new Date(cell.value) < weekAgo)
26             .map((cell) => {
27                 var columnValueDate = new Date(cell.value);
28                 cell.value = new Date(columnValueDate.setFullYear(columnValueDate.getFullYear() + 1));
29                 return cell;
30             })
31     };
32 }).filter((row) => row.cells && row.cells.length > 0);
33 // build options object to send as part of the update request for smartsheet
34 const options = {
35     "sheetId": sheetId,
36     "body": rowsToUpdate
37 };
38 // send updateRow request to Smartsheet
39 smartsheet.sheets.updateRow(options).then(function (data) {
40     console.log('updateRow: ' + JSON.stringify(data));
41 })
42 .catch(function (error) {
43     console.log(error);
44 });
45 ;
46
47
48 cute script
49 entYear(sheetId);
50

```

```

14 // for each column title listed in the dateColumnNames array, get the column
15 for (var i = 0; i < sheet.columns.length; i++) {
16     if (sheet.columns[i].title === dateColumnName) {
17         dateColumnId = sheet.columns[i].id;
18         break;
19     }
20 }
21 // loop the rows in the sheet
22 for (var r = 0; r < sheet.rows.length; r++) {
23     var row = {
24         "id": sheet.rows[r].id,
25         "cells": []
26     };
27     // loop the cells in the row
28     for (var c = 0; c < sheet.rows[r].cells.length; c++) {
29         // for each cell with a columnId that matches a dateColumnId check
30         if (sheet.rows[r].cells[c].columnId === dateColumnId) {
31             var columnValueDate = new Date(sheet.rows[r].cells[c].value);
32
33             // if the date value is in the past, increment the year
34             if (columnValueDate < weekAgo) {
35                 // set the value to new year
36                 var newDate = new Date(columnValueDate.setFullYear(columnValueDate.getFullYear() + 1));
37                 // create a cell object for updating the sheet
38                 var cell = {
39                     "columnId": sheet.rows[r].cells[c].columnId,
40                     "value": newDate
41                 };
42                 // add the cell object to updated row object
43                 row.cells.push(cell);
44             }
45         }
46     }
47     // if the row contains cells add the row to the rowsToUpdate array
48     if (row.cells.length > 0) {
49         rowsToUpdate.push(row);
50     }






```

```

15 // for each column title listed in the dateColumnNames array, get the column
16 sheet.columns
17     .filter((column) => dateColumnName === column.title)
18     .map((column) => dateColumnId = column.id);
19
20 const rowsToUpdate = sheet.rows.map((row) => {
21     return {
22         id: row.id,
23         cells: row.cells
24             .filter((cell) => dateColumnId === cell.columnId &&
25                 new Date(cell.value) < weekAgo)
26             .map((cell) => {
27                 var columnValueDate = new Date(cell.value);
28                 cell.value = new Date(columnValueDate.setFullYear(columnValueDate.getFullYear() + 1));
29                 return cell;
30             })
31     };
32     // build options object to send as part of the update request for smartsheet
33     const options = {
34         "sheetId": sheetId,
35         "body": rowsToUpdate
36     };
37     // send updateRow request to Smartsheet
38     smartsheet.sheets.updateRow(options).then(function (data) {
39         console.log('updateRow: ' + JSON.stringify(data));
40     })
41     .catch(function (error) {
42         console.log(error);
43     });
44 };
45
46
47
48 cute script
49 entYear(sheetId);
50

```

Compatibility

Method					
map()	Yes	9.0	1.5	Yes	Yes
reduce()	Yes	9.0	3.0	4	10.5
filter()	Yes	9.0	1.5	Yes	Yes


Performance

Performance Analysis JS

- <https://github.com/dg92/Performance-Analysis-JS>
- Generates object array of 1,000,000 elements
- Performs functions on the array
- Uses `console.time()` to track time per operation
- Compares functions with for loop

Performance

Test Object



```
{  
  "id": 1,  
  "upvotes": 440,  
  "downvotes": 469,  
  "commentCount": 548  
}
```

Performance

Filter

○○○

```
newData = [];  
const filterValue = Math.random(1,50);  
  
console.time('for loop');  
for(i=0; i<length; i++) {  
    if((+posts[i].upvotes*0.2 +  
        +posts[i].downvotes*0.3 +  
        +posts[i].commentCount*0.1)/3 > filterValue) {  
        newData.push(posts[i]);  
    }  
}  
console.timeEnd('for loop');
```

○○○

```
const filterValue = Math.random(1,50);  
  
console.time('js filter');  
newData = posts.filter(p => (  
    +p.upvotes*0.2 +  
    +p.downvotes*0.3 +  
    +p.commentCount*0.1  
)/3 > filterValue);  
console.timeEnd('js filter');
```

Performance

Map

○○○

```
newData=[];
console.time('for loop');
for(i=0; i<length; i++) {
  newData.push({
    id: posts[i].id,
    upvotes: (+posts[i].upvotes +
              +posts[i].commentCount)/divider,
    downvotes: posts[i].downvotes,
    commentCount: posts[i].commentCount
  });
}
console.timeEnd('for loop');
```

○○○

```
const divider = Math.random(1,300);

console.time('js map');
newData = posts.map(p => {
  return {
    id: p.id,
    upvotes: (+p.upvotes + +p.commentCount)/divider,
    downvotes: p.downvotes,
    commentCount: p.commentCount
  };
});
console.timeEnd('js map')
```

Performance

Reduce

○○○

```
avg = 0;
console.time('for loop');
for(i=0; i<length; i++) {
    avg += (+posts[i].upvotes +
           +posts[i].downvotes +
           +posts[i].commentCount)/3;
}
avg = avg/length;
console.timeEnd('for loop');
```

○○○

```
console.time('js reduce');
avg = posts.reduce((acc, p) =>
    acc+= (+p.downvotes +
          +p.upvotes +
          +p.commentCount)/3,0);

avg = avg/length;
console.timeEnd('js reduce')
```

Performance

Results - Node

	Map	Filter	Reduce
	Processing Time (ms)		
JS Function	555.480	183.204	22.370
For Loop	148.844	181.568	27.255

Times are averages from running performance analysis 5 times

References

Array.map - <https://mzl.la/1xVEqhz>

Array.filter - <https://mzl.la/1dDn8fQ>

Array.reduce - <https://mzl.la/2cjFCXN>

Arrow functions - <https://mzl.la/2z832gs>

References

Performance Analysis JS by Deepak Gupta

<https://github.com/dg92/Performance-Analysis-JS>

Functional Programming in JavaScript Exercises

<http://reactivex.io/learnrx/>

References

Smartsheet Increment Year Sample Code

<https://github.com/stmcallister/smartsheet-increment-year>

Demo Code <https://github.com/stmcallister/why-loop-demos>

Scott McAllister
@stmcallister
github.com/stmcallister

Smartsheet Platform
@SmartsheetDev
github.com/smartsheet-samples

developers.smartsheet.com