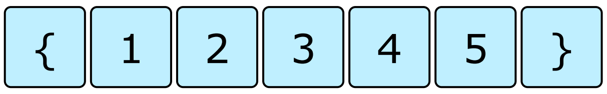
Sets

This chapter will look at how to work with Sets in JavaScript. We will see at how we can create Sets , access elements of a Set and various other functionalities on Sets.

Introduction

A set is an unordered collection of related members in which no member occurs more than once. A set that doesnt contain any element is often called null set. Below is a typical representation of set:



The most common set operations are given below

Union - This is wherein a new set is constructed by combining the members of one set with the members of another set.

Intersection - This is wherein a new set is constructed by adding all the members of one set that also exist in a second set.

Difference - This is wherein a new set is constructed by adding all the members of one set except those that also exist in a second set.

Let's now look at how we can manage and work with sets using JavaScript.

Set Representation in JavaScript

The below code snippet is the most typical representation of a set in javacsript

```
function Set() {
  this.set = {};
  this.size = 0;
}
```

The set is represented as an Object and the values of the set can be inserted as properties. The size is maintained separately using a size variable. There are other ways of implementing sets in Javascript. We have picked Object for it's simiplicity.

Adding items to the set

Now let's look at how we can add elements into our set. The below snippet of code showcases how we can add values to our set data structure.

```
Set.prototype.add = function(data) {
  if (!(this.set.hasOwnProperty(data))) {
    this.set[data] = 'true';
    this.size++;
  }
}
```

The first check is done to see if the data actually exists in our set. This is done by using the hasOwnProperty method which is available for Javascript objects. If not then the data value is added to the set by setting the property of the object to the data which is passed to the add function. The associated value is set to true , just to indicate that the property is part of the set. Technically in this implementation of set the value can be set to anything.

Removing an item from the set

Below is the code snippet that can be used for removing an item from the set

```
Set.prototype.remove = function(data) {
  if (this.set.hasOwnProperty(data)) {
    delete this.set[data];
    this.size--;
  }
}
```

First we ensure that the data value to be removed is available in the set. This is done by using the hasOwnProperty function of JavaScript objects. If the property does exist , then it is removed from the set data structure. At the same time, the size value is decremented to indicate that an element has been removed from the set.

> Run the code below to see add in action

JavaScript

HTML

CSS (SCSS)

```
var myset = new Set();
myset.add(1);
myset.add(2);
myset.add(3);
myset.print();
console.log('removing 2');
myset.remove(2);
myset.print();
```

Console

Clear

```
{1, 2, 3}
removing 2
{1, 3}
```

Membership in the set

One of the key functionalities of sets is the determination of whether a value actually belongs to a set. The following code snippet shows how this can be accomplished.

```
Set.prototype.member = function(data) {
  if (this.set.hasOwnProperty(data)) {
    return true;
  } else {
    return false;
  }
}
```

The above code uses a partial implementation of what is used for remove functionality of sets.

Quiz 1

Q

An ideal representation of a set in javascript is represented as

Check Answers

Exercise 1

As a first exercise, let's implement a method size that returns the number of items in set.

> ***Some tests are failing right now. Implement 'size' to fix them.***

JavaScript

HTML

CSS (SCSS)

```
Set.prototype.sizeOfSet = function() {
  /////// TODO ///////
  ////// IMPLEMENT THIS /////
  ////// RUN AFTER IMPLEMENTING THIS /////
  ////// TESTS SHOULD PASS /////
  return undefined;
}
```

```
// This is the function that runs test cases.
runEvaluation();
```

Console

Clear

```
*** There is some bug lurking there. See failed test cases ***

Here are the tests that ran:

Test case FAILED for Empty Set Size. Result: undefined. Expected: 0

Test case FAILED for Set Size. Result: undefined. Expected: 1

Test case FAILED for Set Size. Result: undefined. Expected: 2
```

Union of two sets

Now let's look at how we can carry out the union of two sets in Javascript. The union of two sets returns a set containing all elements in either this or the other set. Let's look at the code snippet which can be used to accomplish this.

```
Set.prototype.union = function(secondset) {
  var unionset = new Set();
  for (var key in this.set) {
    if (this.set.hasOwnProperty(key)) {
      unionset.add(key);
    }
  }
  for (var key in secondset.set) {
    if (!unionset.set.hasOwnProperty(key)) {
      unionset.add(key);
    }
  }
  return unionset;
}
```

Intersection of two sets

The intersection of two sets returns a set containing all elements that exist in both sets. Here goes the implementation for intersection for our set data structure

```
Set.prototype.intersect = function(secondset) {
  var inter = new Set();
  for (var key in this.set) {

    if (secondset.set.hasOwnProperty(key)) {
      inter.add(key);
    }
  }
  return inter;
}
```

Sidebar on ES6

ES6 has a built-in Set object. It is implemented now in some browsers and in near future will get full support. We recommend that instead of implementing your own Set structure leverage the one thats provided by ES6. A key advantage of this Set Object is that it doesn't coerce all keys to a string like the Object does so you can have both 6 and "6" as separate keys.

Exercise 2

Let us implement a set different (A - B) method. Fix the following function to return all the elements that exist in the current set but not in the *otherset* passed as parameter.

> Run the following code. Implement difference method so that the tests pass.

JavaScript

HTML

CSS (SCSS)

```
Set.prototype.difference = function(secondset) {
  //TODO IMPLEMENT THIS
}


// This is the function that runs test cases.
runEvaluation();
```

Console

Clear

```
*** There is some bug lurking there. See failed test cases ***
```

```
Here are the tests that ran:
```

```
Test case FAILED for Difference of two empty sets. Result: undefined. Expected: 0
```

```
Test case FAILED for Difference of two sets. Result: undefined. Expected: 5
```

Summary

- A set is a data structure which has the following key properties:
    - Elements are unordered
    - No Duplicates
- The add and remove functions are used to add and remove items from the set.
- The union of two sets returns a set containing all elements in either this or the other set
- The intersection of two sets returns a set containing all elements in both this and the other set.