Front-end Advanced

ES6 02 - Iterators and Collections

Table of Contents

Section 1

Symbols

Symbols

- Problem with Object key/value

```
function createToken() {
  return  {
     key: 'SECRET'
  }
}

var token = createToken();
var a = token.key;// easy to access
// easier to modify
token.key = 'NEW';

doTask(token.key); // not working any more
```

Symbols

- Symbol for the rescue

```
// with Symbol
function createToken() {
  var s = Symbol();

   return {
      [s]: 'SECRET'
   }

}

var token = createToken();
var s = Symbol();

token[s]; // undefined
```

Symbols

- What is Symbol ?.
  - Symbol is new primitive in ES6.
  - JS ensure that Symbol will be unique in a same JS runtime
  - Which mean 2 Symbol (created with Symbol()) will always be different

```
var s1 = Symbol();
var s2 = Symbol();

s1 === s2; // false
s1 == s2; // false
```

Symbols

- Where can I find Symbol?
  - JS use **Symbol.iterator** to hide implementation details (its also the main reason why we learn Symbol)

```
var array = [1, 2, 3];

console.log(array); // [1, 2, 4]

console.log(array[Symbol.iterator]); // ƒ values() { [native
code] }
```

Section 2

Iterators

Iterators

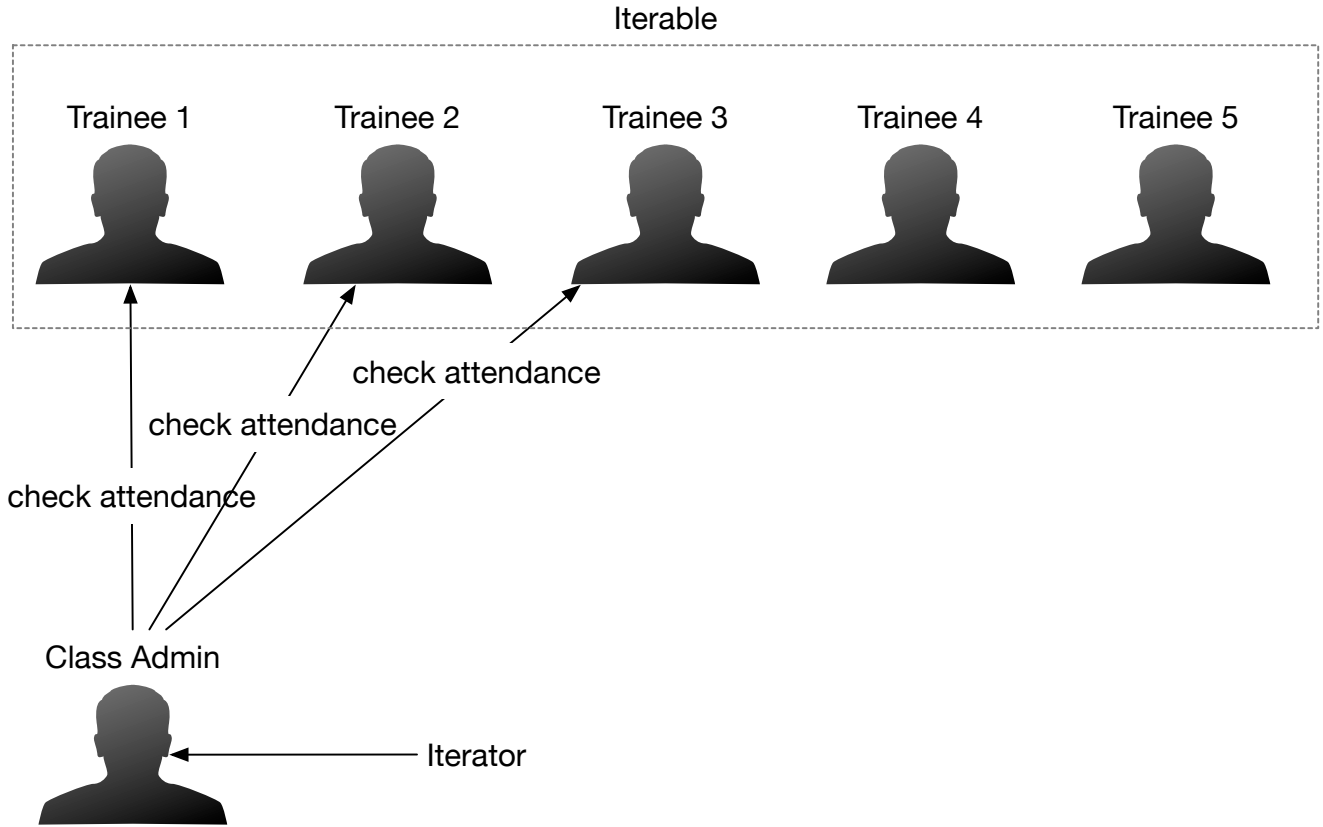
- Ever wonder how **for-of** and **spread** work on array ?

```
var array = [1, 2, 3, 4];

for (var e of array) {
  console.log(e); // 1, 2, 3, 4
}

console.log(...array); // 1 2 3 4
```

Iterators

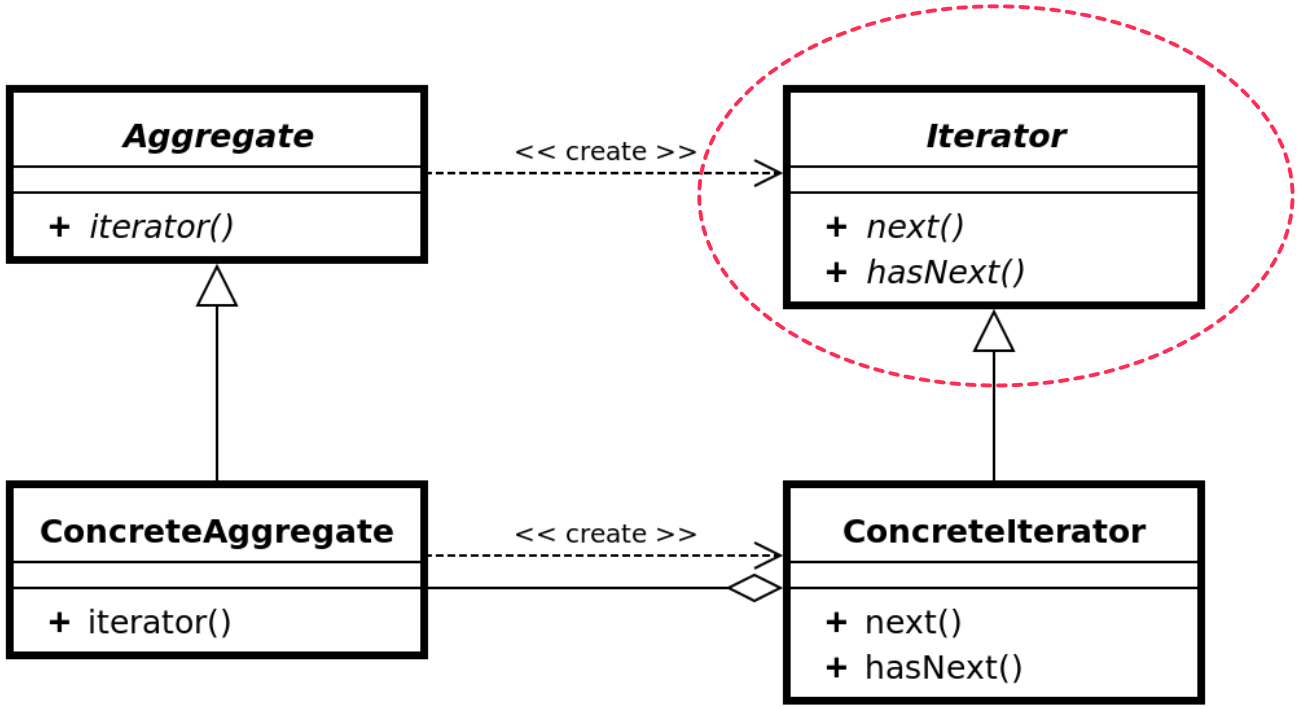
- **Internally**, JS use Iterator pattern to iterate through each element in an array
- Think about Iterator like a Class admin that take attendance by checking each trainee.

Iterators - Explained



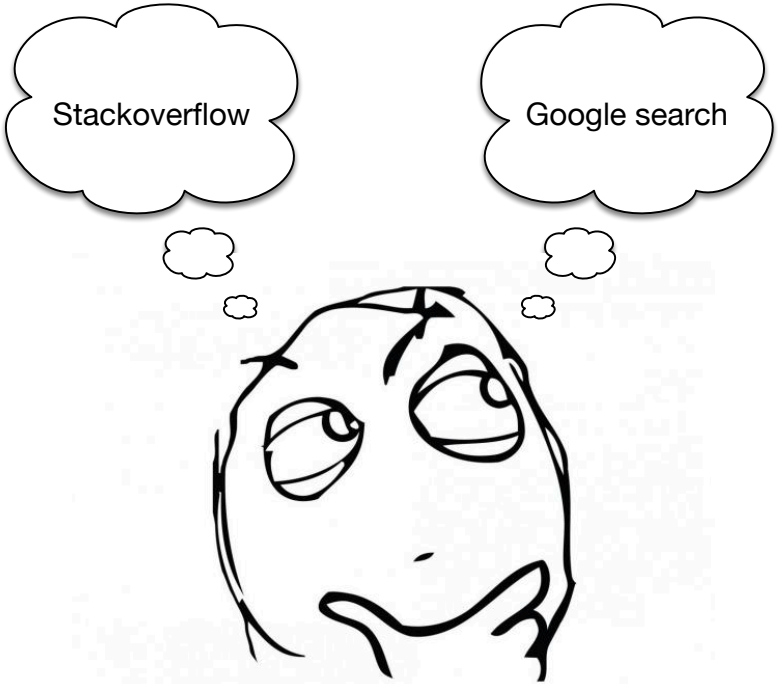
Iterators

- In Design Pattern, Iterator has the following Interface:

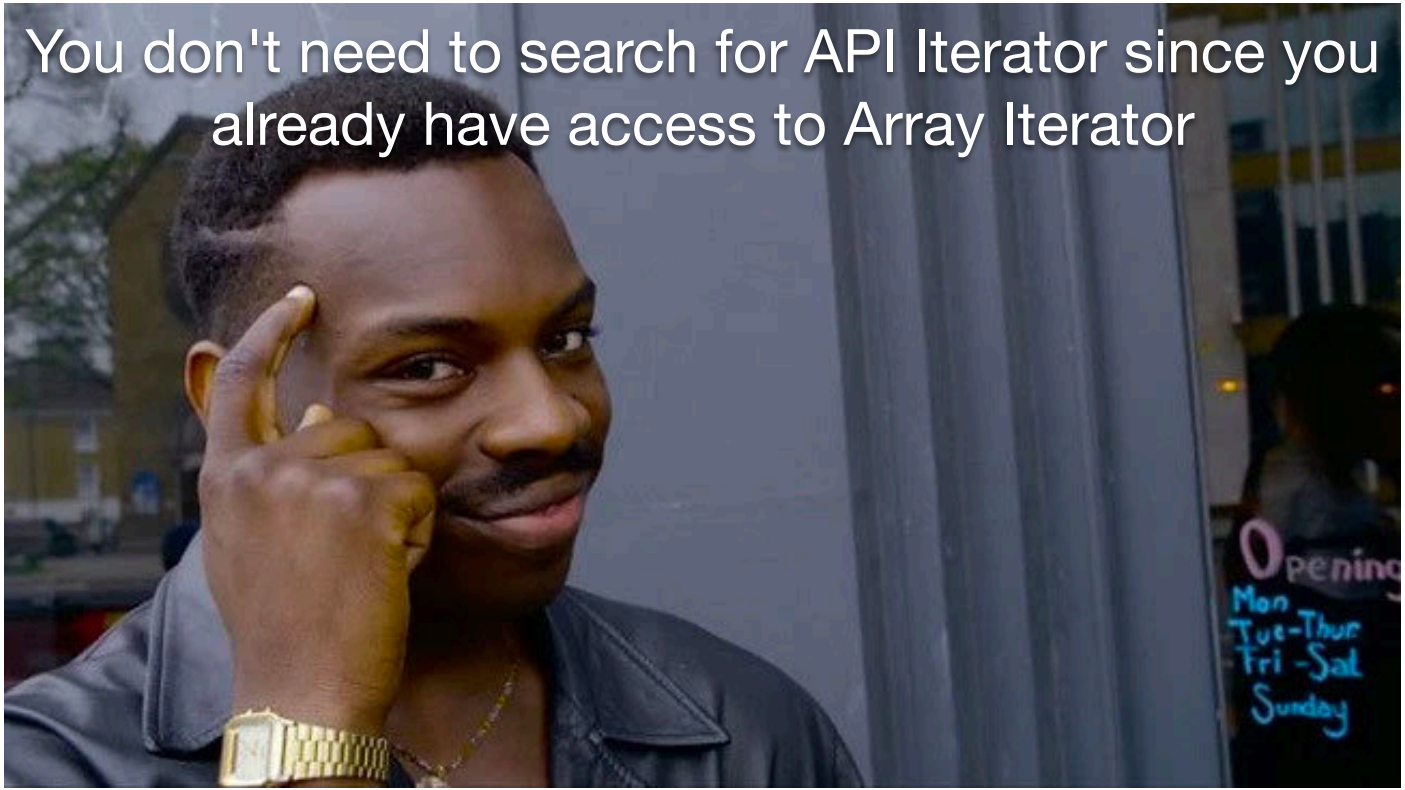


Iterators

- How can we know the API for Iterator in JS ?

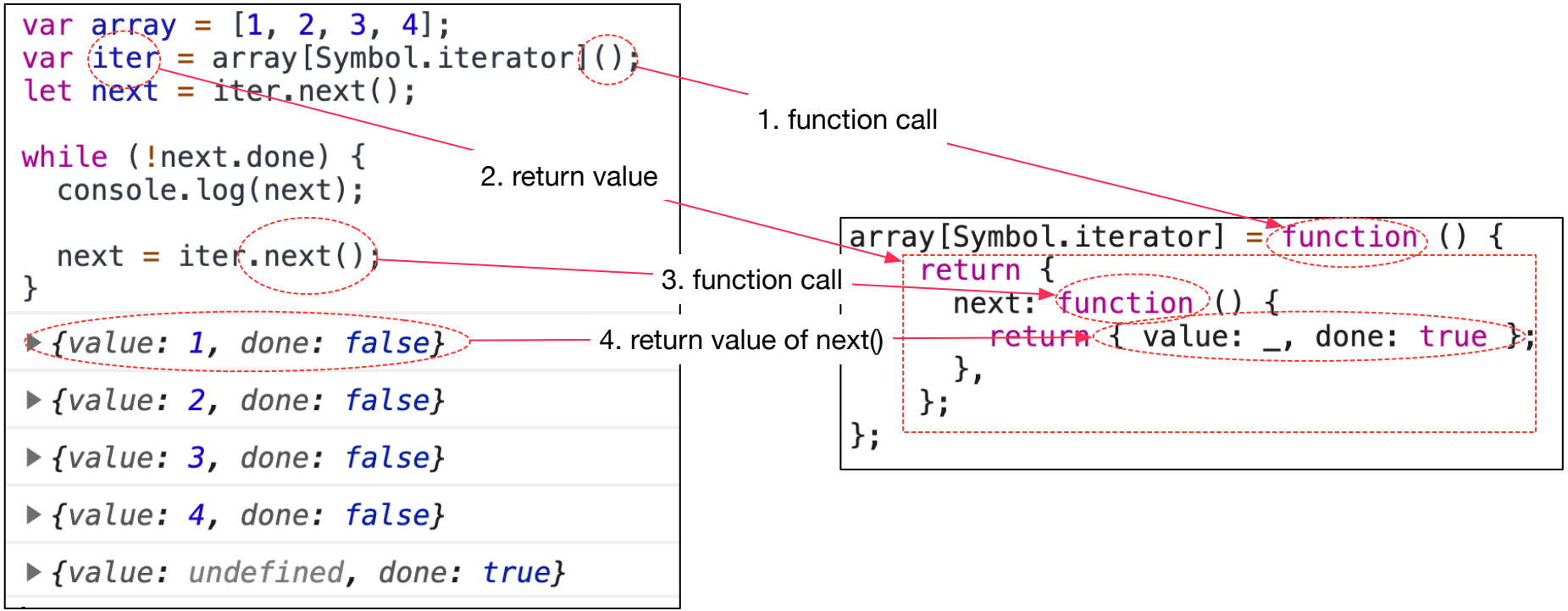


Iterators


You don't need to search for API Iterator since you already have access to Array Iterator
Opening
Mon
Tue-Thur
Fri-Sat
Sunday

Iterators

- Take a look:



Iterators - Usage

Before

```
var obj = {
  name: 'AnhNV',
  clazz: 'React',
  age: 20
}

for (var k of obj) {
  console.log(k);
}
```

```
▶ Uncaught TypeError: obj is not iterable
    at <anonymous>:7:15
```

After

```
var obj = {
  name: 'AnhNV',
  clazz: 'React',
  age: 20,
};

obj[Symbol.iterator] = function () {
  var self = this; // refer to obj when called
  var keys = Object.keys(this); // same as Object.keys(obj);
  var i = 0;

  return {
    next: function () {
      if (i >= keys.length) {
        return { value: undefined, done: true }
      }

      return { value: self[keys[i++]], done: false }
    },
  };
};

for (var k of obj) {
  console.log(k);
}

// better clone object
var o = { ...obj };
console.log(o); // {name: "AnhNV", clazz: "React", age: 20 }
```

Section 3

Generators

Generators

- Iterators is nice but its syntax is scary

```
array[Symbol.iterator] = function () {
    return {
      next: function () {
        return { value: _, done: true };
      },
    };
};
```

Generators

- Generators to the rescue

Before

```
obj[Symbol.iterator] = function () {
  var self = this; // refer to obj when called
  var keys = Object.keys(this); // same as Object.keys(obj);
  var i = 0;

  return {
    next: function () {
      if (i >= keys.length) {
        return { value: undefined, done: true }
      }

      return { value: self[keys[i++]], done: false }
    },
  };
};
```

After

```
obj[Symbol.iterator] = function *() {
  var self = this; // refer to obj when called
  var keys = Object.keys(self); // same as Object.keys(obj);
  // return the keys as array string ['name', 'clazz', 'age']

  for (var i = 0; i < keys.length; i += 1) {
    yield self[keys[i]];
  }

  return;
}
```

Generators

- Syntax:

1. * keyword (must-have)
2. **yield** keyword


```
// generator
function *generator () {
  console.log('yield 1');
  yield 1;

  console.log('yield 2');
  yield 2;

  console.log('yield 2');
  yield 3;

  console.log('return');
  return;
}
```

Generators – Execution flow

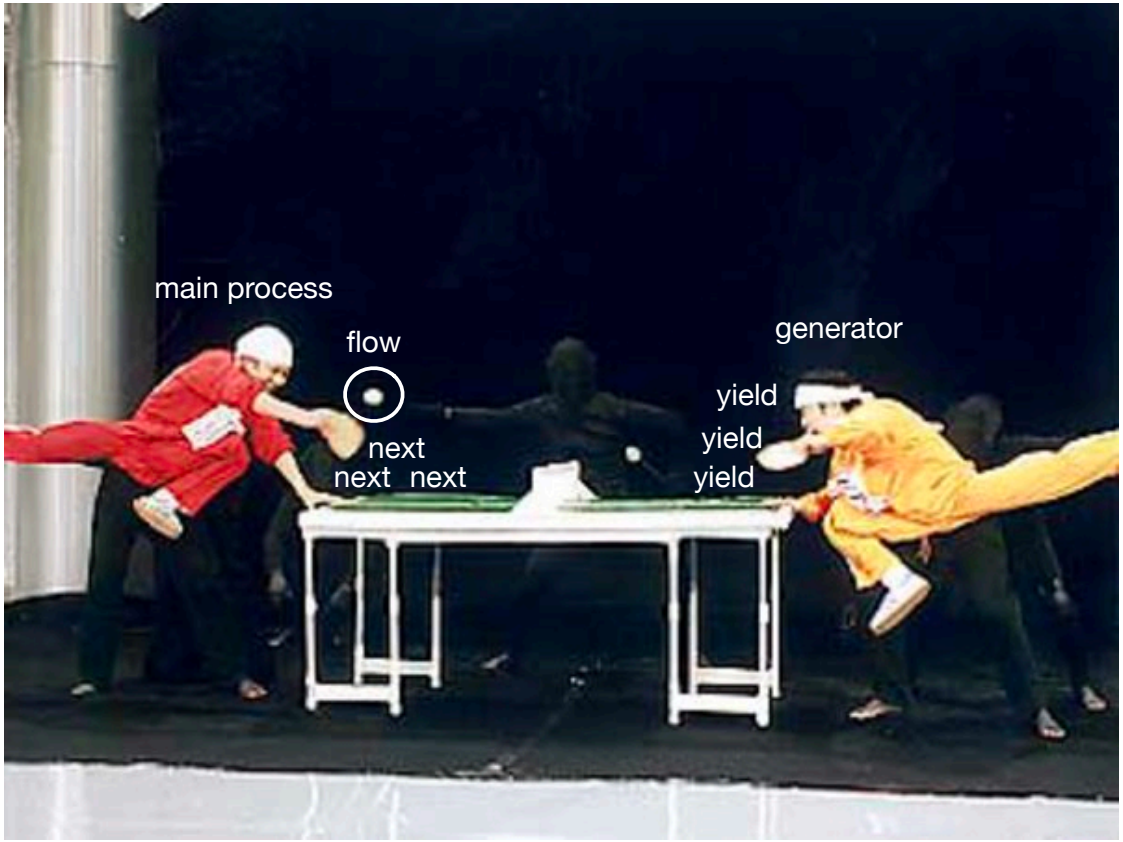
The body of Generator function is not running yet

```
// main process
var iter = generator();

console.log('main: next 1');
iter.next();

console.log('main: next 2');
iter.next();

console.log('main: next 3');
iter.next();

console.log('main: next 4');
iter.next();
```

```
// generator
function *generator () {
  console.log('yield 1');
  yield 1;

  console.log('yield 2');
  yield 2;

  console.log('yield 2');
  yield 3;

  console.log('return');
  return;
}
```

END

Generators – Execution flow



Section 4

Map

Map

- Problem with Object:

```
var k1 = { k: 1 };
var k2 = { k: 2 };

var o = { };

o[k1] = 'Value 1';
o[k2] = 'Value 2'; // overwrite k1

console.log(o[k1]); // Value 2
console.log(o); // {[object Object]: "Value 2"}
```

Map

- Upgraded version of object:

```
var m = new Map();

var k1 = { k: 1 };
var k2 = { k: 2 };

m.set(k1, 'Value 1');
m.set(k2, 'Value 2');

m.get(k1); // Value 1
console.log(m);
```

```
▼Map(2) {{…} => "Value 1", {…} => "Value 2"}
  ▼[[Entries]]
    ▶0: {Object => "Value 1"}
    ▶1: {Object => "Value 2"}
    size: (...)
  ▶__proto__: Map
```

Map

- Simple API:

```
var m = new Map();

var k1 = { k: 1 };
var k2 = { k: 2 };

m.set(k1, 'Value 1'); // set value
m.set(k2, 'Value 2');

m.get(k1); // get value by key

m.has(k1); // check key is in map

m.entries(); // MapIterator: pair of [key, value]
m.keys(); // MapIterator: return array of all keys
m.values(); // MapIterator: return array of all values
```

How to work with Iterator ?

Section 5

Set

Set

- **Problem**: The Back-end API returns a list of user. But that list contain duplicate. You have to remove all duplicates and maintain a list of unique items.

Set

- Set objects are collections of values. A value in the Set **may only occur once**; it is unique in the Set's collection.

```
var s = new Set([1, 2, 4, 1, 1]);
console.log(s); // Set(3) {1, 2, 4}
```

```
▶ Set(3) {1, 2, 4}
```

Set - API

```
var s = new Set([1, 2, 3, 4, 1, 2]); // construct a set with array
console.log(s); // Set(3) {1, 2, 3, 4}

s.add(1); // add new value, duplicate value won't be added
s.add(5); // OK

s.delete(4); // remove 4

s.size; // number of element in set

s.values(); // SetIterator: yield each value in set
```

Summary

- Understand **Symbols** and how to use Symbol to access Iterator function
- Understand **Iterator** object which is used to iterate a collection
- Understand **Generator** function which help developer to create Iterator easier
- Understand **Map/Set** the two new Data Structure in JS

Thank you!