Date : 25 - 06 - 2020
Morning Session : 9am – 11 PM
By ~ Sundeep Charan Ramkumar Today

Topics: ES6 Day 34 (Iterators and Generators)

Iterators:

Symbol => A unique method to define values. Even if the inside value is the same, the output will be false. A symbol can be created with the help of Symbol().

```
const data = { id: 12, numbers: [1, 2, 3] };
const idSymbol = Symbol("id");

data[idSymbol] = 323;
console.log(data);
```

```
                                                    app.js:11
▼ {id: 12, numbers: Array(3), Symbol(id): 323} i
    id: 12
  ▶ numbers: (3) [1, 2, 3]
    Symbol(id): 323
  ▶ __proto__: Object
```

Or

Assigning dynamic values of a variable inside object as a field name

```
const idSymbol = Symbol("id");
const data = { id: 12, numbers: [1, 2, 3], [idSymbol]: 42 };

console.log(data);
```

Iterator is something which consists of a method and that method gives an object of whether it is going to end or it is not going to end.

```
const someNumbers = [2, 4, 6, 8, 10];
const numberIterator = someNumbers[Symbol.iterator]();
console.log(numberIterator);
for (let number of someNumbers) {
  console.log(number);
}
```

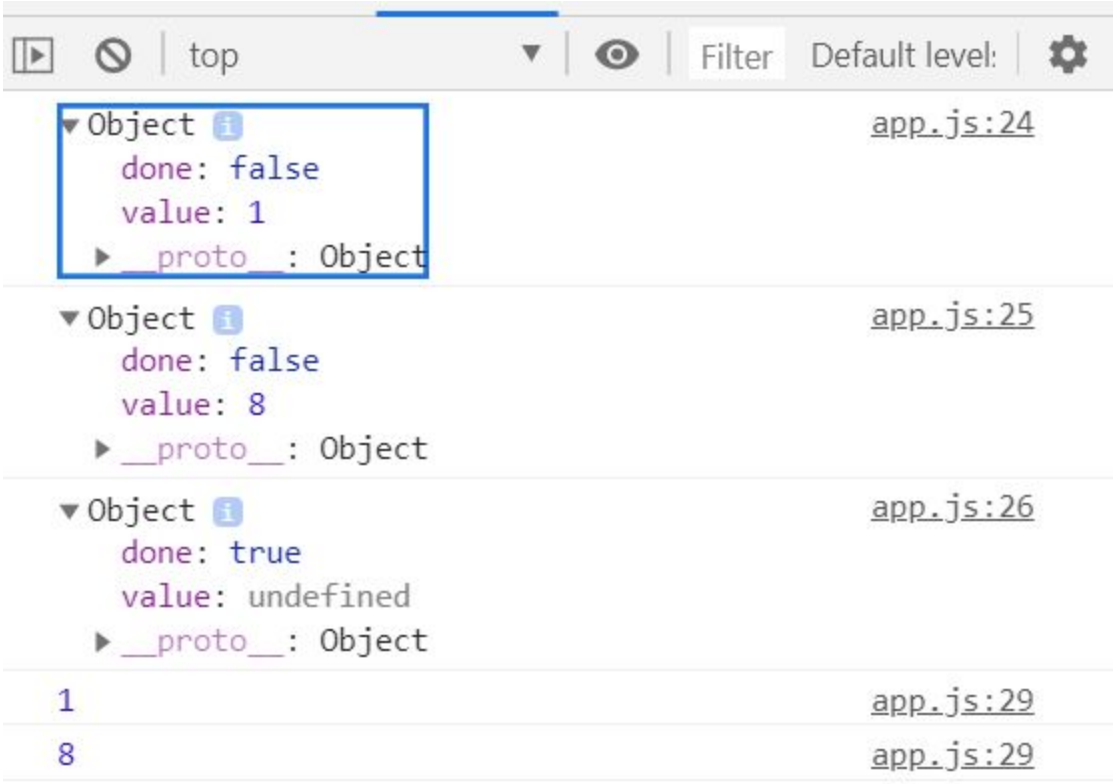
[Symbol.Iterator] : is not a property name

```
// 1, 8, 27, 64, 125, 256, ......
const generateCubeNumbers = (limit = 10) => {
  return {
    [Symbol.iterator]: function () {
      let counter = 1;
      return {
        next: function () {
          let obj = null;
          while (counter <= limit) {
            obj = { value: counter * counter * counter, done: false };
            counter++;
            return obj;
          }
          return { value: undefined, done: true };
        }
      };
    }
  };
};

const cubeNumberIteratorObject = generateCubeNumbers(2);
const cubeNumberIterator = cubeNumberIteratorObject[Symbol.iterator]();

console.log(cubeNumberIterator.next());
console.log(cubeNumberIterator.next());
console.log(cubeNumberIterator.next());

for (let cubeNumber of cubeNumberIteratorObject) {
  console.log(cubeNumber);
}
```



used the name Symbol.iterator. Symbols offer names that are unique and cannot clash with other property names. Also, Symbol.iterator will **return an object called an** iterator. This iterator will have a method called next which will return an object with keys value and done.

Generators: you can pause and resume while executing that particular code.

function* is a new “keyword” for *generator functions* (there are also *generator methods*). yield is an operator with which a generator can pause itself. Additionally, generators can also receive input and send output via yield.

```
function* generatorFunction() {
    yield 6;
    yield 7;
    yield 8;
    return 10;
  }

  function someFunc() {
    return "Good afternoon";
  }

  const generator = generatorFunction();
  console.log(generator.next());
  console.log(generator.next());
  console.log(generator.next());
  console.log(generator.next());

  for (let number of generator) {
    console.log(number);
  }

  const message = someFunc();
  console.log(message);
```

```
▶ {value: 6, done: false}                app.js:44
▶ {value: 7, done: false}                app.js:45
▶ {value: 8, done: false}                app.js:46
▶ {value: 10, done: true}                app.js:47
```

```
// Cube number fetch using generators
function* cubeNumberGenerator(limit = 10) {
  let counter = 1;
  while (counter <= limit) {
    yield counter * counter * counter;
    counter++;
  }
  return undefined;
}

const cubeIterator = cubeNumberGenerator(5);
console.log(cubeIterator.next());
console.log(cubeIterator.next());
console.log(cubeIterator.next());
console.log(cubeIterator.next());
console.log(cubeIterator.next());
console.log(cubeIterator.next());
```

```
▶ {value: 1, done: false}          app.js:96
▶ {value: 8, done: false}          app.js:97
▶ {value: 27, done: false}         app.js:98
▶ {value: 64, done: false}         app.js:99
▶ {value: 125, done: false}        app.js:100
▶ {value: undefined, done: true}   app.js:101
```