Practica 04-02b El objeto Set

Nombre		Curso	
Apellidos		Fecha	

El objeto Set le permite almacenar valores únicos de cualquier tipo, ya sea valores primitivos o referencias a objetos.

Descripción

Los objetos Set son colecciones de valores. Puede iterar a través de los elementos de un conjunto en orden de inserción. Un valor en un Set **solo puede ocurrir una vez**; es único en la colección del Set.

Igualdad de valores

Debido a que cada valor en el Set tiene que ser único, se verificará la igualdad de valores. En una versión anterior de la especificación ECMAScript, esto no se basaba en el mismo algoritmo que el utilizado en el operador ===. Específicamente, para los Set, +0 (que es estrictamente igual a -0) y -0 eran valores diferentes. Sin embargo, esto se cambió en la especificación ECMAScript 2015. Consulte *"Igualdad de clave para -0 y 0"* en la tabla de compatibilidad del navegador para obtener más detalles.
NaN y undefined también se pueden almacenar en un Set. Todos los valores de NaN se equiparan (es decir, NaN se considera lo mismo que NaN, aunque NaN !== NaN).

Rendimiento

El método has de Set verifica si un valor está en un objeto Set, utilizando un enfoque que, en promedio, es más rápido que probar la mayoría de los elementos que se han agregado previamente al objeto Set. En particular, es, en promedio, más rápido que el método Array.prototype.includes cuando un objeto Array tiene un length igual al size de un objeto Set.

Constructor

Set() (en-US)

Crea un nuevo objeto Set.

Propiedades de instancia

Set.prototype.size

Devuelve el número de valores en el objeto Set.

Métodos de instancia

- **Set.prototype.add(*value*)** Añade value al objeto Set. Devuelve el objeto Set con el valor añadido.

- **Set.prototype.clear()** Elimina todos los elementos del objeto Set.

- **Set.prototype.delete(*value*)** Elimina el elemento asociado a value y devuelve un booleano que afirma si un elemento se eliminó correctamente o no.

- **Set.prototype.has(*value*)** Devuelve un booleano que afirma si un elemento está presente con el valor dado en el objeto Set o no.

Métodos de iteración

- **Set.prototype[@@iterator]()**
  Devuelve un nuevo objeto iterador que genera los **values** de cada elemento del objeto Set en el orden de inserción.

- **Set.prototype.values()**
  Devuelve un nuevo objeto iterador que genera los **values** de cada elemento del objeto Set en el orden de inserción.

- **Set.prototype.keys()**
  Un alias para Set.prototype.values().

- **Set.prototype.entries()**
  Devuelve un nuevo objeto iterador que contiene **un arreglo de [value, value]** para cada elemento del objeto Set, en orden de inserción.
  Esto es similar al objeto Map, de modo que la *clave* de cada entrada es la misma que su *valor* para un Set.

- **Set.prototype.forEach(*callbackFn*[, *thisArg*])**
  Llama a callbackFn una vez por cada valor presente en el objeto Set, en orden de inserción. Si se proporciona un parámetro thisArg, se utilizará como valor this para cada invocación de callbackFn.

Ejemplos Utilizando el objeto Set

```
// EJ1 CREAR SET
const mySet1 = new Set();

mySet1.add(1); // Set [ 1 ]
mySet1.add(5); // Set [ 1, 5 ]
mySet1.add(5); // Set [ 1, 5 ]
mySet1.add("algún texto"); // Set [ 1, 5, 'algún texto' ]
const o = { a: 1, b: 2 };
mySet1.add(o);

mySet1.add({ a: 1, b: 2 }); // o está haciendo referencia a un objeto diferente,
// por lo que está bien

// EJ1 HAS
mySet1.has(1); // true
mySet1.has(3); // false, ya que 3 no se ha agregado al conjunto
mySet1.has(5); // true
```


```
mySet1.has(Math.sqrt(25)); // true
mySet1.has("Algún Texto".toLowerCase()); // true
mySet1.has(o); // true

// EJ3 SIZE Y DELETE
mySet1.size; // 5

mySet1.delete(5); // elimina 5 del conjunto
mySet1.has(5); // false, 5 ha sido eliminado

mySet1.size; // 4, ya que acabamos de eliminar un valor
```

Iterando Set

```
// ITERAR SOBRE LOS ELEMENTOS EN SET

var mySet1 = new Set( 1, "algún texto", {"a": 1, "b": 2}, {"a": 1, "b": 2})

// Ej4 for of
//   1, "algún texto", {"a": 1, "b": 2}, {"a": 1, "b": 2}
for (let item of mySet1) console.log(item)
for (let item of mySet1.values()) console.log(item)

// Ej4 for of
// imprime en consola las claves de los elementos
//   1, "algún texto", {"a": 1, "b": 2}, {"a": 1, "b": 2}
for (let item of mySet1.keys()) console.log(item)

// Ej4 for of
// imprime en consola los elementos en el orden:
//   1, "algún texto", {"a": 1, "b": 2}, {"a": 1, "b": 2}


// imprime en consola los elementos en el orden:
//   1, "algún texto", {"a": 1, "b": 2}, {"a": 1, "b": 2}
// (key y value son los mismos aquí)
for (let [key, value] of mySet1.entries()) console.log(key)

// convertir el objeto Set en un objeto Array, con Array.from
const myArr = Array.from(mySet1) // [1, "algún texto", {"a": 1, "b": 2}, {"a": 1, "b": 2}]

// lo siguiente también funcionará si se ejecuta en un documento HTML
mySet1.add(document.body)
mySet1.has(document.querySelector('body')) // true

// conversión entre Set y Array
const mySet2 = new Set([1, 2, 3, 4])
mySet2.size                 // 4
[...mySet2]                 // [1, 2, 3, 4]

// la intersección se puede simular a través de
const intersection = new Set([...mySet1].filter(x => mySet2.has(x)))

// la diferencia se puede simular mediante
const difference = new Set([...mySet1].filter(x => !mySet2.has(x)))

// iterar entradas de Set con forEach()
```


```
mySet2.forEach(function(value) {    console.log(value)  })

// 1
// 2
// 3
// 4
```

Implementación de operaciones básicas de conjuntos

```
function isSuperset(set, subset) {
  for (let elem of subset) {
    if (!set.has(elem)) {
      return false;
    }
  }
  return true;
}

function union(setA, setB) {
  let _union = new Set(setA);
  for (let elem of setB) {
    _union.add(elem);
  }
  return _union;
}

function intersection(setA, setB) {
  let _intersection = new Set();
  for (let elem of setB) {
    if (setA.has(elem)) {
      _intersection.add(elem);
    }
  }
  return _intersection;
}

function symmetricDifference(setA, setB) {
  let _difference = new Set(setA);
  for (let elem of setB) {
    if (_difference.has(elem)) {
      _difference.delete(elem);
    } else {
      _difference.add(elem);
    }
  }
  return _difference;
}

function difference(setA, setB) {
  let _difference = new Set(setA);
  for (let elem of setB) {
    _difference.delete(elem);
  }
  return _difference;
}

// Ejemplos
```


```
const setA = new Set([1, 2, 3, 4]);
const setB = new Set([2, 3]);
const setC = new Set([3, 4, 5, 6]);

isSuperset(setA, setB); // devuelve true
union(setA, setC); // devuelve Set {1, 2, 3, 4, 5, 6}
intersection(setA, setC); // devuelve Set {3, 4}
symmetricDifference(setA, setC); // devuelve Set {1, 2, 5, 6}
difference(setA, setC); // devuelve Set {1, 2}
```

Relación con objetos Array

```
// Use el constructor Set regular para transformar una matriz en un conjunto

let myArray = ["value1", "value2", "value3"];
let mySet = new Set(myArray);

mySet.has("value1"); // devuelve true

// Utilice el operador de dispersión [...obj]  para transformar un conjunto en una matriz.
console.log([...mySet]); // Le mostrará exactamente el mismo Array que myArray
```

Eliminar elementos duplicados del Array

```
// Úselo para eliminar elementos duplicados del Array
// Array => SET =>  Array  con [... set]

const numbers = [2, 3, 4, 4, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 5, 32, 3, 4, 5];

console.log([...new Set(numbers)]);

// [2, 3, 4, 5, 6, 7, 32]
```

Relación con Strings

```
// si pasamos un string lo toma como un array diferenciando entre mayúsculas, minúsculas
let text = "India";

const mySet = new Set(text); // Set(5) {'I', 'n', 'd', 'i', 'a'}
mySet.size; // 5

// mayúsculas, minúsculas y omisión duplicada
new Set("Firefox"); // Set(7) { "F", "i", "r", "e", "f", "o", "x" }
new Set("firefox"); // Set(6) { "f", "i", "r", "e", "o", "x" }
```

Ejercicios

Ejercicio 1: Crear un Set de Personajes Crea un conjunto (`Set`) llamado `personajesSet` e agrega al menos cinco personajes de "El Señor de los Anillos" como elementos.

Ejercicio 2: Imprimir Personajes Itera sobre el conjunto `personajesSet` e imprime cada personaje en la consola.

Ejercicio 3: Verificar la Existencia de un Personaje Escribe una función que tome el conjunto `personajesSet` y un nombre de personaje como argumentos, y devuelva `true` si el personaje está presente en el conjunto, y `false` en caso contrario.

Ejercicio 4: Obtener el Número de Personajes Escribe una función que tome el conjunto `personajesSet` como argumento y devuelva el número total de personajes en el conjunto.

Ejercicio 5: Agregar Personajes Únicos Escribe una función que tome el conjunto `personajesSet` y un nuevo personaje como argumentos, y añada el nuevo personaje al conjunto solo si no está presente. Devuelve el conjunto actualizado.

Ejercicio 6: Eliminar Personaje Escribe una función que tome el conjunto `personajesSet` y un personaje como argumentos, y elimine el personaje del conjunto si está presente. Imprime un mensaje indicando si el personaje fue eliminado o si no estaba en el conjunto.

Ejercicio 7: Combinar dos Sets de Personajes Crea dos conjuntos de personajes, `set1` y `set2`, y luego escribe una función que combine ambos conjuntos en uno solo. Asegúrate de que no haya elementos duplicados en el nuevo conjunto.