SETS, MAPS, MODULES

SETS

MAPS

OTRAS HERRAMIENTAS

MODULOS

SETS

QUÉ SON

- Los sets son **colecciones de valores**, como los arrays.
- **No** están **ordenados ni indexados**
- **No** permiten valores **duplicados**

```
Let data_set = new Set()
```

SETS

EJEMPLO

- **add:** argumento individual. Devuelve el set
- **delete:** solo borra un único valor.

devuelve Boolean

- El set distingue entre 1 y "1" (===)
- **has:** comprueba si existe valor
- **Set más rápido que array**
- **Set -> array** […data_set]
- **No index:** sí recuerda orden inserción

```
let t = new Set(s);            // A new set that copies the elements of s.
let unique = new Set("Mississippi"); // 4 elements: "M", "i", "s", and "p"
unique.size       // => 4
let s = new Set();  // Start empty
s.size            // => 0
s.add(1);         // Add a number
s.size            // => 1; now the set has one member
s.add(1);         // Add the same number again
s.size            // => 1; the size does not change
s.add(true);      // Add another value; note that it is fine to mix types
s.size            // => 2
s.add([1,2,3]);   // Add an array value
s.size            // => 3; the array was added, not its elements
s.delete(1)       // => true: successfully deleted element 1
s.size            // => 2: the size is back down to 2
s.clear();        // Remove everything from the set
s.size            // => 0
```

MAPS

QUÉ SON

- **Set de claves asociadas a un valor**

```
let m = new Map();  // Create a new, empty map
let n = new Map([   // A new map initialized with string keys mapped to numbers
   ["one", 1],
   ["two", 2]
]);
```

- **set:** sirve par… …y existe, sustituye valor. Permite encadenar varios sets.
- **get:** obtiene valor dada una key
- **has:** verifica si existe una key
- **delete:** borra una key y su valor
- **clear:** borra todo el mapa

MAPS

EJEMPLO

```
let m = new Map();   // Start with an empty map
m.size               // => 0: empty maps have no keys
m.set("one", 1);     // Map the key "one" to the value 1
m.set("two", 2);     // And the key "two" to the value 2.
m.size               // => 2: the map now has two keys

m.get("two")      // => 2: return the value associated with key "two"
m.get("three")    // => undefined: this key is not in the set
m.set("one", true);  // Change the value associated with an
existing key
m.size            // => 2: the size doesn't change
m.has("one")      // => true: the map has a key "one"
m.has(true)       // => false: the map does not have a key true
m.delete("one")   // => true: the key existed and deletion succeeded
m.size            // => 1
m.delete("three") // => false: failed to delete a nonexistent key
m.clear();        // Remove all keys and values from the map
```

MAPS

OTROS

- **Cualquier valor** de javascript puede ser key
- Los **mapas son iterables** (resultado es array con dos elementos, clave y valor)
- La **iteración** va en **orden de inserción**
- **.keys():** itera sobre las claves
- **.values():** itera sobre los valores

OTRAS HERRAMIENTAS

EXPRESIONES REGULARES

- **Herramienta muy utilizada**
- Buscar en el texto expresiones "concretas"
- Ejemplo: imagenes_de_david_1.png, imagenes_de_david_2.png…
- Patrones alfanuméricos, inicios, finales, agrupaciones…

```
let pattern = /s$/;  // both expressions are the same
let pattern = new RegExp("s$");
pattern.test("suerte"); // false
pattern.test("suertes); // true
```

OTRAS HERRAMIENTAS

DATETIME

- **Tiempos y fechas**
- **Timestamp:** valor de la cantidad de segundos transcurridos desde el 1 de Enero de 1970 UTC -> estandariza el tiempo

```
let startTime = Date.now();
reticulateSplines(); // Do some time-consuming operation
let endTime = Date.now();
```

OTRAS HERRAMIENTAS

DATETIME

- **Ejemplo**

```
let d = new Date(2020, 0, 1, 17, 10, 30); // 5:10:30pm on New Year's Day 2020
d.toString()   // => "Wed Jan 01 2020 17:10:30 GMT-0800 (Pacific Standard
Time)"
d.toUTCString()        // => "Thu, 02 Jan 2020 01:10:30 GMT"
d.toLocaleDateString()  // => "1/1/2020": 'en-US' locale
d.toLocaleTimeString()  // => "5:10:30 PM": 'en-US' locale
d.toISOString()        // => "2020-01-02T01:10:30.000Z"
```

OTRAS HERRAMIENTAS

JSON

- JavaScript Object Notation

```
{
        “hola”: [1, 2, 3],
        “suerte”: “yepe”
}
```

OTRAS HERRAMIENTAS

JSON

- Stringify (streaming)

```
let o = {s: "", n: 0, a: [true, false, null]};

let s = JSON.stringify(o);  // s == '{"s":"","n":0,"a":[true,false,null]}'

let copy = JSON.parse(s);   // copy == {s: "", n: 0, a: [true, false, null]}
```

MÓDULOS

INTRODUCCIÓN

- **Programas grandes** de manera cómoda
- **Divide y vencerás**
- **Evitar duplicación**
- **Mejor orden**
- **Colaboración entre desarrolladores**

MÓDULOS

NODE

- **Exports**

```
exports.mean = data => data.reduce(sum)/data.length;
exports.stddev = function(d) {
   let m = exports.mean(d);
   return Math.sqrt(d.map(x => x - m).map(square).reduce(sum)/(d.length-1));
};
module.exports = class BitSet extends AbstractWritableSet {
   // implementation omitted
};
```

También se puede exportar todo del tirón al final del módulo:

```
// Now export only the public ones
module.exports = { mean, stddev };
```

MÓDULOS

NODE

- **Imports externos**

```
// These modules are built in to Node
const fs = require("fs");       // The built-in filesystem module
const http = require("http");   // The built-in HTTP module

// The Express HTTP server framework is a third-party module.
// It is not part of Node but has been installed
const express = require("express");
```

MÓDULOS

NODE

- **Imports internos al programa**

```
// Import the entire stats object, with all of its functions
const stats = require('./stats.js');

// We've got more functions than we need, but they're neatly
// organized into a convenient "stats" namespace.
let average = stats.mean(data);

// Alternatively, we can use idiomatic destructuring
assignment to import
// exactly the functions we want directly into the local namespace:
const { stddev } = require('./stats.js');

// This is nice and succinct, though we lose a bit of context
// without the 'stats' prefix as a namspace for the stddev() function.
let sd = stddev(data);
```

MÓDULOS

ES6

- **Exports**

```
export const PI = Math.PI;

export function degreesToRadians(d) { return d * PI / 180; }

export class Circle {
    constructor(r) { this.r = r; }
    area() { return PI * this.r * this.r; }
}
```

```
export { Circle, degreesToRadians, PI };
```

MÓDULOS

ES6

- **Imports with export statements**

```
import BitSet from './bitset.js';
import { mean, stddev } from "./stats.js";
import * as stats from "./stats.js";
```

- **Import full file**

```
import "./analytics.js";
```

BIBLIOGRAFÍA

1. JavaScript: The Definitive Guide, 7th Edition
2. Eloquent JavaScript, 3th edition, Marijn Haverbeke
3. Clean Code: A Handbook of Agile Software Craftsmanship (Robert C. Martin Series)
4. Modern c++ Programming with Test-Driven Development, Jeff Langr
5. Refactoring: Improve the design of existing Code, Martin Fowler
6. Game programming patterns, Robert Nystrom