ITERATORS AND GENERATORS AND OTHERS

ITERADORES Y GENERADORES

EXCEPCIONES

MÓDULOS

ITERADORES Y GENERADORES

INTRODUCCIÓN

- Multiples variables iteradoras: arrays, strings, mapas, objetos
- Varias maneras de iterar: spread, for/of…
- Asignación desestructurada

```
> a
[ 5, 4, 3 ]
> console.log(...a)
5 4 3
```

```
> let [b, c, d] = a;
undefined
> b
5
> c
4
> d
3
```

ITERADORES Y GENERADORES

ITERADORES

- Formado por tres partes
- El objeto **iterable**
- El objeto **iterador**
- El objeto **resultado** de iteración (**value y done**)

```
> let suerte_iterable = [1, 2, 3];
undefined
> let suerte_iteratorr = suerte_iterable[Symbol.iterator]();
undefined
> for (let result=suerte_iteratorr.next(); !result.done; result=suerte_iteratorr.next()){console.log(result.value)}
1
2
3
```

ITERADORES Y GENERADORES

Creación de clase iterable

- Una clase no iterable queremos que lo sea
- Se convierte con el método Symbol.iterator

```
/*
 * A Range object represents a range of numbers {x: from <= x <= to}
 * Range defines a has() method for testing whether a given number is a member
 * of the range. Range is iterable and iterates all integers within the range.
 */
class Range {
  constructor (from, to) {
    this.from = from;
    this.to = to;
  }

  // Make a Range act like a Set of numbers
  has(x) { return typeof x === "number" && this.from <= x && x <= this.to; }

  // Return string representation of the range using set notation
  toString() { return `{ x | ${this.from} ≤ x ≤ ${this.to} }`; }
```

ITERADORES Y GENERADORES

Creación de clase iterable

```
  // Note that the name of this method is a special symbol, not a string.
  [Symbol.iterator]() {
    // Each iterator instance must iterate the range independently of
    // others. So we need a state variable to track our location in the
    // iteration. We start at the first integer >= from.
    let next = Math.ceil(this.from);  // This is the next value we return
    let last = this.to;           // We won't return anything > this
    return {                    // This is the iterator object
      // This next() method is what makes this an iterator object.
      // It must return an iterator result object.
      next() {
        return (next <= last)   // If we haven't returned last value yet
          ? { value: next++ } // return next value and increment it
          : { done: true };   // otherwise indicate that we're done.
      },

      // As a convenience, we make the iterator itself iterable.
      [Symbol.iterator]() { return this; }
    };
  }
}

for(let x of new Range(1,10)) console.log(x); // Logs numbers 1 to 10
[...new Range(-2,2)]                  // => [-2, -1, 0, 1, 2]
```

ITERADORES Y GENERADORES

Hay otras maneras (busca por internet cómo)

ITERADORES Y GENERADORES

Generadores

- Es un iterador que computa nuevos elementos en base a un cálculo.
- Se crea a partir de una función generadora
- Se debe marcar el punto de devolución con la key **yield**
- La función se define con un *

ITERADORES Y GENERADORES

Ejemplo

```
2   function* power_generator(base)
3   {
4       let initial_value = 1;
5       while(true)
6       {
7           initial_value *= base;
8           yield initial_value;
9       }
10  }
11
12
13  let generator = power_generator(5);
14  console.log(generator.next());
15  console.log(generator.next());
16  console.log(generator.next());
17  console.log(generator.next());
18
```

EXCEPCIONES

DEFINICIÓN

- Una herramienta para controlar nuestro programa frente a errores
- Ejemplo 4xx-5xx para servers
- Ejemplo: documentamos que no aceptamos imágenes de mayor tamaño que 300x300, porque nos podría saturar la memoria en disco. El cliente se lee la documentación, pero o no se entera o se salta este paso. ***¿Podemos dejar que el cliente nos sature nuestro Sistema?***

EXCEPCIONES

EJEMPLO SIMPLE

```
2   function analyze_image(image)
3   {
4
5       let dimensions = [ image.length, image[0].length ];
6       if (dimensions[0] > 2 || dimensions[1] > 2)
7       {
8           throw ("image size too big!");
9       }
10  }
11
12
13  console.log(analyze_image([[1, 2], [1, 3]]));
14  console.log(analyze_image([[1, 2, 4], [1, 3, 5]]));
15
16
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
    internalBinding('errors').triggerUncaughtException(
                              ^
image size too big!
(Use `node --trace-uncaught ...` to show where the exception was thrown)
PS C:\Users\David\Dropbox\albaniles_digitales\core_javascript>
```

EXCEPCIONES

TRY/CATCH

```
16
17  function analyze_image_try(image)
18  {
19      try
20      {
21          let dimensions = [ image.length, image[0].length ];
22          if (dimensions[0] > 2 || dimensions[1] > 2)
23          {
24              throw ("image size too big!");
25          }
26      }
27      catch (err)
28      {
29          console.log("image is to big, please reduce size to 300x300");
30      }
31  }
32
33
34  console.log(analyze_image_try([[1, 2], [1, 3]]));
35  console.log(analyze_image_try([[1, 2, 4], [1, 3, 5]]));
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS C:\Users\David\Dropbox\albaniles_digitales\core_javascript> node '.\9 ARRAY METHOD
ple2.js'
undefined
image is to big, please reduce size to 300x300
undefined
PS C:\Users\David\Dropbox\albaniles_digitales\core_javascript>
```

EXCEPCIONES

TRY/CATCH/FINALLY

```
36
37  function analyze_image_try_finally(image)
38  {
39      let analysis_data = 1;
40      let extra_analysis_data = 1;
41      try
42      {
43          let dimensions = [ image.length, image[0].length ];
44          if (dimensions[0] > 2 || dimensions[1] > 2)
45          {
46              throw ("image size too big!");
47          }
48          extra_analysis_data = 2;
49      }
50      catch (err)
51      {
52          console.log("image is to big, please reduce size to 300x300");
53          extra_analysis_data = 0;
54      }
55      finally
56      {
57          analysis_data = 100;
58      }
59      return [analysis_data, extra_analysis_data];
60  }
61
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS C:\Users\David\Dropbox\albaniles_digitales\core_javascript> node '.\9 ARRAY METHODS
[ 100, 2 ]
image is to big, please reduce size to 300x300
[ 100, 0 ]
```

EXCEPCIONES

¡NOTA!

EXCEPCIONES COMPLICADAS -> SMELL CODE

EXCEPCIONES

SISTEMA DE EXCEPCIONES

```
image is to big, please reduce size to 300x300
undefined


node:internal/process/esm_loader:94
    internalBinding('errors').triggerUncaughtException(
                              ^
not controlled error
(Use `node --trace-uncaught ...` to show where the exception was thrown)
PS C:\Users\David\Dropbox\albaniles_digitales\core_javascript>
```

```
class ImageSizeError extends Error
{
    constructor(msg)
    {
      super(msg);
      this.name = 'CustomError';
    }
}

function analyze_image_try_custom_error(image)
{
    try
    {
        let dimensions = [ image.length, image[0].length ];
        if (dimensions[0] > 2 || dimensions[1] > 2)
        {
            throw (ImageSizeError);
        }
        throw ("not controlled error");
    }
    catch (err)
    {
        if ( err.name == "ImageSizeError" )
        {
            console.log("image is to big, please reduce size to 300x300");
        } else
        {
            throw err;
        }
    }
}

console.log(analyze_image_try_custom_error([[1, 2, 4], [1, 3, 5]]));
console.log(analyze_image_try_custom_error([[1, 2], [1, 3]]));
```

MODULOS

LINT

- Se utiliza para arreglar estéticamente Código
- Automático
- Herramienta típica: ESLint
- (en vscode existe extension)
- Npm install eslint
- Fichero configurable .eslintrc

MODULOS

PRETTIER

- Parecida al linter

MODULOS

NPM

- Comando que se utiliza para instalar librerías
- npm install <package>
- npm install --save-dev <package>
- npm install -g <package>
- npm uninstall -g <package>

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