Fat Arrow

Francisco Arce
www.pacoarce.com

Arrow

- Conocidas en otros lenguajes (C#, Java) como “expresiones lambda”, arrows o flechas son abreviaciones de funciones utilizando el operador =>
- Una forma más ***compacta*** para hacer funciones en JS.
- “this” se maneja diferente.
- Funciones de una sola línea.

Arrow

```
//var x = function(a) { return a; }
var x = a => a;
            alert(x(10));
            //Sin parámetros
            var y = () => alert("Hola, desde ES6");
            y();
            //Más de un parámetro
            var z = (a,b,c) => alert(a+" "+b+" "+c);
            z("Hola","cara","de bola");
            //Desde otra función
            setTimeout(()=>alert("pasaron 5 segundos"),5000);
```

Francisco Arce
www.pacoarce.com

Arrow

```
odds  = evens.map(v => v + 1);

pairs = evens.map(v => ({ even: v, odd: v + 1 }));

nums  = evens.map((v, i) => v + i);
```

Arrow

```
nums.forEach(v => {
  if (v % 5 === 0)
    fives.push(v);
});
```

Arrow

```
this.nums.forEach((v) => {
    if (v % 5 === 0)
        this.fives.push(v);
});
```

Referencias

https://medium.com/@lehiarteaga/ecmascript-6-es6-y-sus-caracter%C3%ADsticas-55a1fc9275b1
http://www.enrique7mc.com/2015/12/novedades-de-es6/
https://www.youtube.com/watch?v=J85lRtO_yjY
https://www.youtube.com/watch?v=6sQDTgOqh-I

ES6: clases

Francisco Arce
www.pacoarce.com

Class

- Ahora podemos hacer clases por medio de la sentencia “class”.
- Podemos utilizar el método “ constructor()” para crear la función constructora.
- Los lenguajes tradicionales basados en clases ofrecen la palabra reservada ***this*** para referencia la instancia actual de la clase.
- En Javascript ***this*** se refiere ***al contexto de la llamada*** y como tal puede ser cambiado a algo más que un objeto.

Francisco Arce
www.pacoarce.com

ES6: instancias

Francisco Arce
www.pacoarce.com

Objeto

- Un objeto es una instancia de la clase, la cual es creada usando el operador ***new***.
- Cuando se usa un punto para acceder al método del objeto, ***this*** se va a referir al objeto a la izquierda al punto.

Francisco Arce
www.pacoarce.com

Objeto

let burger = new Hamburger();

burger.listToppings();

- En este código vemos que cuando ***this*** es usada desde adentro de la clase Hamburger, Se va a referir al objeto ***burger***.

ES6: herencia

Francisco Arce
www.pacoarce.com

Herencia

- Al igual que en otros lenguajes de programación, una clase puede extender otra clase heredando métodos o propiedades de la clase padre.

Herencia

- La función ***super()*** ejecuta el método con el mismo nombre desde el que se está llamando a ***super()***, de esta forma al definir el nuevo constructor llamamos a ***super()*** y le pasamos los mismos parámetros que recibe el constructor, entonces se ejecuta ese constructor y luego código del nuevo.

ES6: Getters y Setters

Francisco Arce
www.pacoarce.com

Getters y Setters

- En algunos lenguajes de programación (como Java) existen los ***getters*** y ***setters***.
- Estos métodos que se usan para controlar variables internas de un objeto (propiedades).
- Para usarlos simplemente se agrega ***get*** o ***set*** delante del nombre del método de la siguiente forma:

Francisco Arce
www.pacoarce.com

Getters y Setters

- Definir un método ***get*** con el nombre que quieras (no puede ser el nombre de la propiedad) y este debería devolver el valor deseado (técnicamente puede hacer cualquier cosa el método), o defines un método ***set*** con otro nombre (tampoco el mismo de la propiedad) y que recibe el nuevo valor y lo asigna a ***this*** .

Getters y Setters

- Aunque esto hace bastante más legible y limpio el código, al tener métodos específicos para obtener o modificar propiedades del objeto, la verdad es que no son necesarios ya que simplemente usando la sintaxis de objetos de toda la vida puedes obtener el valor de una propiedad y modificarlo.

ES6: Métodos estáticos

Francisco Arce
www.pacoarce.com

Métodos estáticos

- Al igual que en otros lenguajes también va a ser posible crear métodos estáticos usando la palabra clave ***static*** antes del nombre del método.

```
class miClase {
  static miMetodo() {
    return 'hola mundo'
  }
}
```

Métodos estáticos

- Luego para poder usarlo simplemente llamas al método desde la clase sin instanciar:

```
let mensaje = miClase.miMetodo(); // 'hola mundo';
```

ES6: Características

Francisco Arce
www.pacoarce.com

Características

- Los nombres de las clases no pueden ser ***eval*** ó ***arguments***;
- No están permitidos nombres de clase repetidos.
- El nombre constructor solo puede ser usado para métodos, no para ***getters***, ***setter*** o un generador de métodos
- Las clases no se pueden llamar antes de definirse.
- Todavía se puede instanciar la clase desde cualquier parte, solo es necesario esperar a que esté definida.

Referencias

https://medium.com/@lehiarteaga/ecmascript-6-es6-y-sus-caracter%C3%ADsticas-55a1fc9275b1
http://www.enrique7mc.com/2015/12/novedades-de-es6/
https://platzi.com/blog/ecmascript-nueva-sintaxis/
http://es6-features.org/#ClassInheritanceFromExpressions

ES6

Class Inheritance, From Expressions

Francisco Arce
www.pacoarce.com

Class Inheritance, From Expressions

- Class Inheritance, From Expressions
- Support for mixin-style inheritance by extending from expressions yielding function objects. [Notice: the generic aggregation function is usually provided by a library like this one, of course]
-

Francisco Arce
www.pacoarce.com

Class Inheritance, From Expressions

```
var aggregation = (baseClass, ...mixins) => {
    let base = class _Combined extends baseClass {
        constructor (...args) {
            super(...args)
            mixins.forEach((mixin) => {
                mixin.prototype.initializer.call(this)
            })
        }
    }
    let copyProps = (target, source) => {
        Object.getOwnPropertyNames(source)
```

Francisco Arce
www.pacoarce.com

Class Inheritance, From Expressions

Base Class Access

Intuitive access to base class constructor and methods.

```
class Shape {
    ...
    toString () {
        return `Shape(${this.id})`
    }
}
class Rectangle extends Shape {
    constructor (id, x, y, width, height) {
```

Francisco Arce
www.pacoarce.com

Class Inheritance, From Expressions

Static Members
Simple support for static class members.

```
class Rectangle extends Shape {
    …
    static defaultRectangle () {
        return new Rectangle("default", 0, 0, 100, 100)
    }
}
class Circle extends Shape {
    …
```

Francisco Arce
www.pacoarce.com

Class Inheritance, From Expressions

Getter/Setter also directly within classes (and not just within object initializers, as it is possible since ECMAScript 5.1).

```
class Rectangle {
    constructor (width, height) {
        this._width  = width
        this._height = height
    }
    set width  (width)  { this._width = width           }
    get width  ()       { return this._width            }
    set height (height) { this._height = height         }
```

Francisco Arce
www.pacoarce.com

ES6

Template Strings

Francisco Arce
www.pacoarce.com

Template Strings

- Son un tipo especial de cadena con formato, similares a la interpolación en otros lenguajes como Ruby, se definen con un par de caracteres back-tick (`) o acentos agudos del francés, a diferencia de las cadenas normales que usan comillas sencillas o dobles.

Template Literals

```
var s1= `esta es una template string`;
// Pueden contener valores
var n = 5;
var s2 = `El valor de n es ${n}`;

// Pueden abarcar múltiples líneas
var s3 = `Esta es una cadena
escrita en dos líneas`;
alert(s2);
```

Francisco Arce
www.pacoarce.com

Template Literals

```
var customer = { name: "Foo" }

var card = { amount: 7, product: "Bar", unitprice: 42 }

var message = `Hello ${customer.name},

want to buy ${card.amount} ${card.product} for

a total of ${card.amount * card.unitprice} bucks?`
```

Referencias

https://medium.com/@lehiarteaga/ecmascript-6-es6-y-sus-caracter%C3%ADsticas-55a1fc9275b1
http://www.enrique7mc.com/2015/12/novedades-de-es6/
http://es6-features.org/#StringInterpolation

ES6

Let y const

Francisco Arce
www.pacoarce.com

Let y Const

- ***let*** indica que una variable sólo va a estar definida en un bloque en particular, al terminar el bloque la variable deja de existir, esto es muy útil para evitar errores lógicos cuando alteramos una variable que no deberíamos.

Let y Const

```
function letTest() {
        if (true) {
                let x = 23;
                 console.log(x);  // 71
         }
        console.log(x);  // no existe x
}
```

Francisco Arce
www.pacoarce.com

Let y Const

- ***const*** por su parte previene que una variable declarada cambie de valor, convirtiéndose efectivamente en una constante.
- Siempre es recomendable usar constantes para valores que sabemos que no van a cambiar, así se evitan modificaciones inesperadas.

Let y Const

```
const a = 7;

a = 5; // error

console.log(a);
```

Referencias

https://medium.com/@lehiarteaga/ecmascript-6-es6-y-sus-caracter%C3%ADsticas-55a1fc9275b1
http://www.enrique7mc.com/2015/12/novedades-de-es6/

ES6

Generadores

Francisco Arce
www.pacoarce.com

Generadores

- Los generadores son un tipo especial de función que regresa una serie de valores con un algoritmo definido por el usuario.
- Una función se convierte en ***generador*** si contiene una o más expresiones ***yield*** y se declara con ***function****.

Francisco Arce
www.pacoarce.com

Generadores

- Para utilizar un ***generador***, asignamos a una variable el resultado de esta función y llamamos al método ***next()*** que devuelve un objeto con propiedad ***value***.
- La expresión ***yield*** se encarga de devolver el valor, pero además guarda el estado interno de la función.

Francisco Arce
www.pacoarce.com

Generadores

```
function* rango (inicio, fin, incremento) {
    while (inicio < fin) {
        yield inicio
        inicio+= incremento
    }
}

for (let i of rango(0, 10, 2)) {
    console.log(i) // 0, 2, 4, 6, 8
}
```

Francisco Arce
www.pacoarce.com

Referencias

https://medium.com/@lehiarteaga/ecmascript-6-es6-y-sus-caracter%C3%ADsticas-55a1fc9275b1
http://www.enrique7mc.com/2015/12/novedades-de-es6/
http://es6-features.org/#GeneratorMethods

Generadores

```
let fibonacci = {
    *[Symbol.iterator]() {
        let pre = 0, cur = 1
        for (;;) {
            [ pre, cur ] = [ cur, pre + cur ]
            yield cur
        }
    }
}
```

Francisco Arce
www.pacoarce.com

Generadores

```
for (let n of fibonacci) {
    if (n > 10)
        break
    console.log(n)
}
```

Francisco Arce
www.pacoarce.com

ES6

Literales octales y binarias

Francisco Arce
www.pacoarce.com

Literales octales y binarias

- Hay ocasiones en que el contexto de nuestros datos requiere que trabajemos con cifras no decimales, por ejemplo en base 2 (binario) o base 8 (octal), ahora es sencillo crear este tipo de literales con los prefijos (0b) y (0o) respectivamente.

Literales octales y binarias

```
var a = 0b111110111; // binario

console.log(a); // 503

var b = 0o767; // octal

console.log(b); // 503
```

Referencias

https://medium.com/@lehiarteaga/ecmascript-6-es6-y-sus-caracter%C3%ADsticas-55a1fc9275b1
http://www.enrique7mc.com/2015/12/novedades-de-es6/
https://www.youtube.com/watch?v=xt9CqO8snb0

ES6

Maps y Sets

Francisco Arce
www.pacoarce.com

Maps y Sets

- Los mapas (Map) son una estructura de datos que almacenan pares de llave (key) y valor (value), los conjuntos (Set) tienen la característica de no aceptar duplicados, y ambos permiten búsquedas eficientes cuando se tiene un gran volumen de información porque no guardan sus elementos ordenados por un índice, como ocurre con los arreglos.

Referencias

https://medium.com/@lehiarteaga/ecmascript-6-es6-y-sus-caracter%C3%ADsticas-55a1fc9275b1
http://www.enrique7mc.com/2015/12/novedades-de-es6/
http://es6-features.org/#SetDataStructure
http://es6-features.org/#WeakLinkDataStructures

Maps y Sets

- Set Data-Structure
- Cleaner data-structure for common algorithms based on sets.
- let s = new Set()
- s.add("hello").add("goodbye").add("hello")
- s.size === 2
- s.has("hello") === true
- for (let key of s.values()) // insertion order
- console.log(key)

Francisco Arce
www.pacoarce.com

Maps y Sets

- Map Data-Structure
- Cleaner data-structure for common algorithms based on maps.
- let m = new Map()
- let s = Symbol()
- m.set("hello", 42)
- m.set(s, 34)
- m.get(s) === 34
- m.size === 2
- for (let [key, val] of m.entries())
- console.log(key + " = " + val)
-

Francisco Arce
www.pacoarce.com

ES6

Promises

Francisco Arce
www.pacoarce.com

Promises

- El flujo de información de Internet tiene características asíncronas, lo que significa que mientras esperamos el resultado de una operación como por ejemplo que carguen los datos de una página web, un programa puede realizar otras operaciones y utilizarlo cuando el resultado esté listo.
- Las “promesas” (***promises***) son objetos que representan esta clase de operaciones y los datos que se obtienen.
- Las promesas nos sirven para administrar las funciones de ***callback***, procesos asíncronos.

Promises

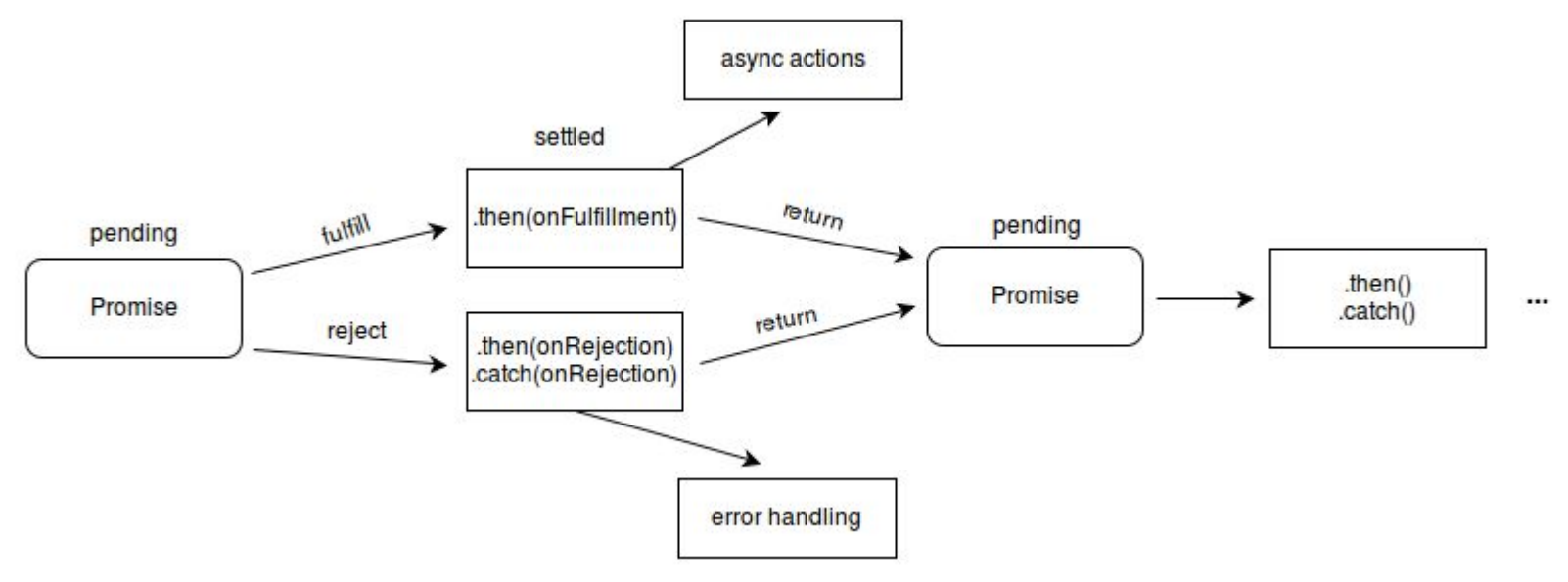
- Una promesa tiene tres estados: pending, fulfilled (success) y rejected (error).
- Las promesas se pueden encadenar.
- Para enviar información dentro de las promesas, se envían dentro de resolve() y de reject(), de dentro de ***return***.
- Con ***throw*** cortamos las promesas.

Francisco Arce
www.pacoarce.com

Promises

- Promises a+
- Accius
- Angular tiene manejador de promesas toPromise
- jQuery tiene deferred
- Polyfill es6 promise (https://github.com/stefanpenner/es6-promise)
- await
- https://polyfill.io/v2/docs/

Promises



Francisco Arce
www.pacoarce.com

Promises

Crear una promesa

- new Promise(function(resolve, reject){})
- new Promise((resolve, reject)=>{})
- La opción ***resolve()*** regresa el valor correcto (*success*).
- La opción ***reject()*** regresa el valor erróneo (*error*).

Francisco Arce
www.pacoarce.com

Promises

Llamar a las promesas

- Tiene un método que es “then”, lee el valor correcto (*success*).
- Tiene un método que es “catch”, lee el valor erróneo (*error*).

ES6

Encadenar promesas

Francisco Arce
www.pacoarce.com

Promises

Más de una promesa

- Encadenar varias promesas
- all[ ] indican si se cuemplen ***todas*** las promesas: El método Promise.all(iterable) devuelve una promesa que termina correctamente cuando todas las promesas en el argumento iterable han sido concluídas con éxito, o bien rechaza la petición por la primera promesa que es rechazada.

Promises

Más de una promesa

- race[ ] indican si se cumple “alguna” de las promesas, la más rápida. El método Promise.race(iterable) retorna una promesa que se cumplirá o no ***tan pronto*** como una de las promesas del argumento iterable se cumpla o se rechace, con el valor o razón de rechazo de ésta.

Promises

```
let miPromesa = new Promise(function(resolve, reject){

});
```

ES6

Crear una promesa con AJAX

Francisco Arce
www.pacoarce.com

Crear una promesa con Ajax

- En esta clase realizaremos un proceso asíncrono con Ajax controlado con una promesa.
- Es necesario ejecutarlo en un servicio de Internet, como Apache o IIS.

Francisco Arce
www.pacoarce.com

Referencias

https://medium.com/@lehiarteaga/ecmascript-6-es6-y-sus-caracter%C3%ADsticas-55a1fc9275b1

http://www.enrique7mc.com/2015/12/novedades-de-es6/

http://es6-features.org/#PromiseCombination

https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/Promise

https://promisesaplus.com/

https://www.youtube.com/watch?v=9im-5iDgH54&t=6329

Promises

```
function msgAfterTimeout (msg, who, timeout) {
    return new Promise((resolve, reject) => {
        setTimeout(() => resolve(`${msg} Hello ${who}!`), timeout)
    })
}
msgAfterTimeout("", "Foo", 100).then((msg) =>
    msgAfterTimeout(msg, "Bar", 200)
).then((msg) => {
    console.log(`fin desdpués de 300ms:${msg}`)
})
```

Francisco Arce
www.pacoarce.com

Promises

- Promise Combination
- Combine one or more promises into new promises without having to take care of ordering of the underlying asynchronous operations yourself.

Promises

- https://script.aculo.us/
- http://backbonejs.org/
- Prototype.js
- Callback hell
-

ES6

Objetos de propagación o ...spread

Francisco Arce
www.pacoarce.com

...Spread

- La sintaxis de ***Spread*** nos permite expandir la expresión para los siguientes casos:
  - Arreglos (arrays)
  - Llamadas a Funciones
  - Múltiple ***destructuring*** de variables

Francisco Arce
www.pacoarce.com

...Spread

```
//ejemplo en funciones

const suma = (a, b) => a + b;

let nums = [3, 5];

suma(...nums); // igual   a suma(nums[0], nums[1])
```

Francisco Arce
www.pacoarce.com

...Spread

```
let cde = ['c', 'd', 'e'];

let abc = ['a', 'b', ...cde, 'f', 'g'];  // ['a', 'b', 'c', 'd', 'e', 'f', 'g'];
```

Francisco Arce
www.pacoarce.com

...Spread

```
let mapABC  = { a: 5, b: 6, c: 3};

let mapABCD = { ...mapABC, d: 7};  // { a: 5, b: 6, c: 3, d: 7 }
```

Francisco Arce
www.pacoarce.com

Objetos de propagación

https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Operadores/Spread_operator

ES6

Valores por default en argumentos

Francisco Arce
www.pacoarce.com

Valores por default en argumentos

- En EcmaScript 6 podemos recibir valor por omisión o default en los parámetros de las funciones. Pueden ser definidos como variables primitivas, funciones, expresiones o arreglos.

Valores por default en argumentos

```
function multiplicar(a, b) {
  b = typeof b !== 'undefined' ?  b : 1;
  return a*b;
}

multiplicar(5); // 5
```

Valores por default en argumentos

```
function multiplicar(a, b = 1) {
  return a*b;
}

multiplicar(5); // 5
```

Valores por default en argumentos

```
function f (x, y = 7, z = 42) {
    return x + y + z
}
f(1) === 50
```

Valores por default en argumentos

```
function cambiaFondo(elemento, color = 'yellow') {
  elemento.style.backgroundColor = color;
}

cambiaFondo(algunDiv);           // color configurado a 'yellow'
cambiaFondo(algunDiv, undefined); // color configurado a 'yellow' también
cambiaFondo(algunDiv, 'blue');    // color configurado a 'blue'
```

Valores por default en argumentos

```
function agregar(valor, arreglo = []) {
  arreglo.push(valor);
  return arreglo;
}

agregar(1); //[1]
agregar(2); //[2], no [1, 2]
```

Valores por default en argumentos

```
function llamarAlgo(cosa = algo()) { return cosa; }

function algo(){
  return "Hola";
}

llamarAlgo();  //hola
```

Valores por default en argumentos

```
function f(x=1, y) {
  return [x, y];
}

f(); // [1, undefined]
```

Valores por default en argumentos

```
function f([x, y] = [1, 2], {z: z} = {z: 3}) {
  return x + y + z;
}
f(); // 6
```

ES6

Argumentos extendidos

Francisco Arce
www.pacoarce.com

Argumentos extendidos

- Extended Parameter Handling
- Rest Parameter
- Aggregation of remaining arguments into single parameter of variadic functions.

```
function f (x, y, ...a) {
    return (x + y) * a.length
}
f(1, 2, "hello", true, 7) === 9
```

Referencias

http://es6-features.org/#DefaultParameterValues

https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Funciones/Parametros_por_defecto

ES6

Mejoras en las expresiones regulares

Francisco Arce
www.pacoarce.com

Mejoras en las expresiones regulares

La bandera “y” mantiene la posición entre coincidencias mediante el parámetro ***lastIndex***, por lo que se le llama “búsqueda pegajosa” o “bandera adhesiva”.

ES6

Mejoras en las expresiones regulares

Francisco Arce
www.pacoarce.com

2. Mejoras en las expresiones regulares

Se anexa la bandera **/y** cambia dos cosas al hacer coincidir una expresión regular (re) con una cadena:

2. Mejoras en las expresiones regulares

Anclado a ***re.lastIndex***: la coincidencia debe comenzar en ***re.lastIndex*** (el índice después de la coincidencia anterior).

Este comportamiento es similar al anclaje ^, pero con ese anclaje, las coincidencias siempre deben comenzar en el índice 0.

2. Mejoras en las expresiones regulares

match() repetidamente: si se encontró una coincidencia, ***re.lastIndex*** se establece en el índice después de la coincidencia.

2. Mejoras en las expresiones regulares

El principal caso de uso para este comportamiento de coincidencia es la simbología, donde desea que cada coincidencia siga inmediatamente a su predecesor.

Un ejemplo de “tokenización” a través de una expresión regular ***adhesiva*** y exec () se da más adelante.

2. Mejoras en las expresiones regulares

Si la bandera /g no está establecido, la coincidencia siempre comienza al principio, pero se salta hacia adelante hasta que se encuentra una coincidencia. ***REGEX.lastIndex*** no se cambia.

lastIndex es una propiedad de lectura / escritura entera de instancias de expresiones regulares que especifica el índice en el que se inicia la próxima coincidencia.

2. Mejoras en las expresiones regulares

```
const REGEX = /a/;

REGEX.lastIndex = 7; // ignored

const match = REGEX.exec('xaxa');

console.log(match.index); // 1

console.log(REGEX.lastIndex); // 7 (unchanged)
```

2. Mejoras en las expresiones regulares

Si se establece la bandera ***/g***, la coincidencia comienza en ***REGEX.lastIndex*** y se salta hacia adelante hasta que se encuentre una coincidencia.

REGEX.lastIndex se establece en la posición después de la coincidencia.

Eso significa que recibirá todas las coincidencias si realiza un ciclo hasta que ***exec()*** devuelva nulo.

2. Mejoras en las expresiones regulares

```
const REGEX = /a/g;
REGEX.lastIndex = 2;
const match = REGEX.exec('xaxa');
console.log(match.index); // 3
console.log(REGEX.lastIndex); // 4 (updated)
// No match at index 4 or later
console.log(REGEX.exec('xaxa')); // null
```

2. Mejoras en las expresiones regulares

Si solo la bandera ***/y*** se establece, la coincidencia se inicia en ***REGEX.lastIndex*** y se ancla a esa posición (no se salta hacia adelante hasta que se encuentre una coincidencia).
REGEX.lastIndex se actualiza de forma similar a cuando se establece la bandera ***/g***.

2. Mejoras en las expresiones regulares

```
const REGEX = /a/y;

// No match at index 2
REGEX.lastIndex = 2;
console.log(REGEX.exec('xaxa')); // null

// Match at index 3
REGEX.lastIndex = 3;
const match = REGEX.exec('xaxa');
console.log(match.index); // 3
console.log(REGEX.lastIndex); // 4
```

Francisco Arce
www.pacoarce.com

2. Mejoras en las expresiones regulares

La configuración de **/ *y*** y **/ *g*** es la misma que la configuración **/ *y***.

2. Mejoras en las expresiones regulares

2.2. RegExp.prototype.test(str)

test() works the same as exec(), but it returns true or false (instead of a match object or null) when matching succeeds or fails:

```
const REGEX = /a/y;
REGEX.lastIndex = 2;
console.log(REGEX.test('xaxa')); // false
REGEX.lastIndex = 3;
console.log(REGEX.test('xaxa')); // true
console.log(REGEX.lastIndex); // 4
```

Francisco Arce
www.pacoarce.com

Referencias

http://es6-features.org/#RegularExpressionStickyMatching

https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Regular_Expressions

https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/RegExp

http://2ality.com/2015/07/regexp-es6.html

https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/RegExp/lastIndex

Francisco Arce
www.pacoarce.com

ES6

Objetos

Francisco Arce
www.pacoarce.com

Objetos

1. Sintaxis corta para los objetos:

Antes:
obj = {x:x, y:y}

Ahora:
obj = { x, y }

Objetos

2. Soporte de nombres calculados en la definición de propiedades:

```
let obj = {
    foo: "bar",
    [ “id" + num() ]: 4
}
```

Objetos

3. Soporte de notación tipo método en la definición de propiedades, en forma regular y como función generadora.

```
obj = {
    foo (a, b) { },
    bar (x, y) { },
    *num (x, y) { }
}
```

Objetos

4. Método assign() para copiar objetos:

```
var o1 = { a: 1 };
var o2 = { b: 2 };
var o3 = { c: 3 };

var obj = Object.assign(o1, o2, o3);
console.log(obj); // { a: 1, b: 2, c: 3 }
console.log(o1);  // { a: 1, b: 2, c: 3 }
```

Objetos

4. Método assign() para copiar objetos:

```
var o1 = { a: 1, b: 1, c: 1 };
var o2 = { b: 2, c: 2 };
var o3 = { c: 3 };

var obj = Object.assign({}, o1, o2, o3);
console.log(obj); // { a: 1, b: 2, c: 3 }
```

Objetos

4. Método assign() para copiar objetos:

```
let empleado1= {nombre:”Pedro”, edad:30};
let empleado2= empleado1;
empleado2.nombre = “Pablo”;

let empleado2= Object.assign({},empleado1,{nombre:”Pablo”});
```

Referencias

http://es6-features.org/#PropertyShorthand

https://www.youtube.com/watch?v=Gh2FaDqZp2g

https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Operators/Object_initializer

Objetos

4. Método create():
5. Método is()

ES6

Symbol Type

Francisco Arce
www.pacoarce.com

Symbol Type

- Los “símbolos” son un nuevo tipo de dato primitivo.
- Los símbolos son únicos e inmutables y son utilizados como identificadores para propiedades de objetos.
- Los símbolos pueden tener una descripción opcional, pero sólo es utilizada para fines del depurador.

Symbol Type

- Cada valor del tipo Symbol tiene asociado un valor del tipo String o Undefined que sirve únicamente como descripción del símbolo.
- El constructor Symbol no debe ser usado con el operador new.
- Tampoco debe ser extendido mediante clases.

Symbol Type

```
var sim1 = Symbol();
var sim2 = Symbol("foo");
var sim3 = Symbol("foo");
```

El código anterior crea tres símbolos nuevos. Hay que destacar que Symbol("foo") no convierte la cadena "foo" en un símbolo, sino que crea un símbolo nuevo que tiene la misma descripción.

Symbol Type

Symbol("foo") === Symbol("foo"); // false

- La siguiente sintaxis con el operador new lanzará un TypeError:

var sym = new Symbol(); // TypeError

Symbol Type

```
var sym = Symbol("foo");
typeof sym;     // "symbol"
var symObj = Object(sym);
typeof symObj;  // "object"
```

Symbol.for() y Symbol.keyFor()

Francisco Arce
www.pacoarce.com

Symbol Type

```
Symbol("foo") !== Symbol("foo")
const foo = Symbol()
const bar = Symbol()
typeof foo === "symbol"
typeof bar === "symbol"
```

Francisco Arce
www.pacoarce.com

Symbol Type

```
let obj = {}
obj[foo] = "foo"
obj[bar] = "bar"
JSON.stringify(obj) // {}
Object.keys(obj) // []
Object.getOwnPropertyNames(obj) // []
Object.getOwnPropertySymbols(obj) // [ foo, bar ]
```

Francisco Arce
www.pacoarce.com

Symbol Type

```
const COLOR_RED    = Symbol('Red');
const COLOR_ORANGE = Symbol('Orange');
const COLOR_YELLOW = Symbol('Yellow');
const COLOR_GREEN  = Symbol('Green');
const COLOR_BLUE   = Symbol('Blue');
const COLOR_VIOLET = Symbol('Violet');
```

Symbol Type

```
function getComplement(color) {
    switch (color) {
        case COLOR_RED:
            return COLOR_GREEN;
        case COLOR_ORANGE:
            return COLOR_BLUE;
        case COLOR_YELLOW:
            return COLOR_VIOLET;
        case COLOR_GREEN:
            return COLOR_RED;
        case COLOR_BLUE:
            return COLOR_ORANGE;
        case COLOR_VIOLET:
            return COLOR_YELLOW;
        default:
```

Francisco Arce
www.pacoarce.com

Referencias

http://es6-features.org/#GlobalSymbols

https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/Symbol

https://www.youtube.com/watch?v=DHrYasp1OTw

http://exploringjs.com/es6/ch_symbols.html

Francisco Arce
www.pacoarce.com

ES6

Nuevos métodos

Francisco Arce
www.pacoarce.com

Nuevos métodos

Asignación de propiedad de objeto:
Nueva función para asignar propiedades enumerables de uno o más objetos fuente a un objeto de destino (método ***assign***).

Nuevos métodos

Búsqueda de un elemento en un arreglo:
Nueva función para encontrar un elemento en un arreglo.

```
[ 1, 3, 4, 2 ].find(x => x > 3) // 4
[ 1, 3, 4, 2 ].findIndex(x => x > 3) // 2
```

Nuevos métodos

Repetición de cadenas
Nueva funcionalidad de repetición de cadenas.

```
" ".repeat(4 * depth)
"foo".repeat(3)
```

Nuevos métodos

Búsqueda de cadenas
Nuevas funciones de cadena específicas para buscar una subcadena.

```
var cadena = "hola cara de bola";
console.log(cadena.startsWith("ola",1));
console.log(cadena.endsWith("ola"));
console.log(cadena.includes("ola"));
console.log(cadena.includes("ola", 1));
console.log(cadena.includes("ola", 2));
```

Francisco Arce
www.pacoarce.com

Nuevos métodos

Comprobación de tipo de número
Nuevas funciones para verificar números que no son números y números finitos.

```
Number.isNaN(42) === false
Number.isNaN(NaN) === true

Number.isFinite(Infinity) === false
Number.isFinite(-Infinity) === false
Number.isFinite(NaN) === false
Number.isFinite(123) === true
```

Francisco Arce
www.pacoarce.com

Nuevos métodos

Número de verificación de seguridad
Comprobar si un número entero está en el rango seguro, es decir, está representado correctamente por JavaScript (donde todos los números, incluidos los números enteros, son técnicamente números en coma flotante).
± 9007199254740991 o ± 9,007,199,254,740,991

Number.isSafeInteger(42) === true
Number.isSafeInteger(9007199254740992) === false

Nuevos métodos

Comparación de números
Disponibilidad de un valor estándar de Epsilon para una comparación más precisa de los números de coma flotante.

```
console.log(0.1 + 0.2 === 0.3); // false
console.log(Math.abs((0.1 + 0.2) - 0.3) < Number.EPSILON);  // true
```

Nuevos métodos

Número de truncamiento
Trunca un número de coma flotante a su parte integral, soltando completamente la parte fraccionaria.

```
console.log(Math.trunc(42.7)) // 42
console.log(Math.trunc( 0.1)) // 0
console.log(Math.trunc(-0.1)) // -0
```

Nuevos métodos

Determinación de signo de número
Determine el signo de un número, incluidos los casos especiales de cero firmado y no número.

```
console.log(Math.sign(7))   // 1
console.log(Math.sign(0))   // 0
console.log(Math.sign(-0))  // -0
console.log(Math.sign(-7))  // -1
console.log(Math.sign(NaN)) // NaN
```

Referencias

http://es6-features.org/#NumberSignDetermination

ES6

Internationalization & Localization

Francisco Arce
www.pacoarce.com

Internationalization & Localization

Collation: Podemos modificar el método sort() sobre los diferentes idiomas:

```
// En alemán  "ä" se ordena junto a "a"
// En Sueco, "ä" se ordena después de a "z"
var lista = [ "ä", "a", "z" ];
var l10nDE = new Intl.Collator("de");
var l10nSV = new Intl.Collator("sv");
l10nDE.compare("ä", "z") === -1;
l10nSV.compare("ä", "z") === +1;
console.log(lista.sort(l10nDE.compare)); // [ "a", "ä", "z" ]
console.log(lista.sort(l10nSV.compare)); // [ "a", "z", "ä" ]
```

Francisco Arce
www.pacoarce.com

Internationalization & Localization

Formateo de números según su zona:

```
var l10nEN = new Intl.NumberFormat("en-US");
var l10nDE = new Intl.NumberFormat("de-DE");
l10nEN.format(1234567.89) === "1,234,567.89";
l10nDE.format(1234567.89) === "1.234.567,89";
```

Internationalization & Localization

Formateo de monedas:

```
var l10nUSD = new Intl.NumberFormat("en-US", { style: "currency", currency: "USD" });
var l10nGBP = new Intl.NumberFormat("en-GB", { style: "currency", currency: "GBP" });
var l10nEUR = new Intl.NumberFormat("de-DE", { style: "currency", currency: "EUR" });
l10nUSD.format(100200300.40) === "$100,200,300.40";
l10nGBP.format(100200300.40) === "£100,200,300.40";
l10nEUR.format(100200300.40) === "100.200.300,40 €";
```

Francisco Arce
www.pacoarce.com

Internationalization & Localization

Formateo de fechas:

```
var l10nEN = new Intl.DateTimeFormat("en-US");
var l10nDE = new Intl.DateTimeFormat("de-DE");
l10nEN.format(new Date("2015-01-02")) === "1/2/2015";
l10nDE.format(new Date("2015-01-02")) === "2.1.2015";
```

Francisco Arce
www.pacoarce.com

Referencias

http://es6-features.org/#DateTimeFormatting

https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/Intl

http://www.ecma-international.org/ecma-402/1.0/#sec-8

ES6

Destructuring

Francisco Arce
www.pacoarce.com

Destructuring

- ***Destructuring*** es un nuevo método para extraer datos rápidamente de un objeto { } o un arreglo [ ] sin tener que escribir mucho código.

```
let foo = ['uno', 'dos', 'tres'];
let [one, two, three] = foo;
console.log(one); // uno
```

Francisco Arce
www.pacoarce.com

Destructuring

```
let modulo = {
   cuadrado(lon) { console.log(lon*lon);},
   circulo(radio) { console.log(radio*Math.PI); },
   texto(text) { console.log(text); },
};
let {cuadrado, texto, circulo} = modulo;
cuadrado(5);
texto('hola');
circulo(10);
```

Francisco Arce
www.pacoarce.com

Destructuring

- ***Destructuring*** también puede ser usado para pasar objetos a una función, permitiéndonos obtener propiedades específicas de un objeto.
- También nos permite asignar valores por default como argumentos.

Destructuring

```
let juana = { nombre: 'Juana', paterno: 'Pérez'};
let juan = { nombre: 'Juan', paterno: 'López', materno: 'Pérez' }
function nombreCompleto({nombre, paterno, materno = 'N/A'}) {
  console.log(`Hola ${nombre} ${paterno} ${materno}`)
}
nombreCompleto(juana) // -> Hola Juana Pérez N/A
nombreCompleto(juan) // -> Hola Juan López Pérex
```

Francisco Arce
www.pacoarce.com

Objetos de propagación

https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Operadores/Spread_operator

Destructuring

```
var params = [ "hello", true, 7 ]
var other = [ 1, 2, ...params ] // [ 1, 2, "hello", true, 7 ]

function f (x, y, ...a) {
    return (x + y) * a.length
}
f(1, 2, ...params) === 9

var str = "foo"
var chars = [ ...str ] // [ "f", "o", "o" ]
```

Francisco Arce
www.pacoarce.com