Desafío

Si defino

```
function theSpec() {
  return { height: 200, width: 150 }
}
```

e invoco varias veces esta función ¿obtengo siempre el mismo objeto, o cada vez un clon distinto?

Si es siempre el mismo ¿cómo hacer para que devuelva clones?

Si son clones ¿cómo hacer para que devuelva siempre el mismo?

Testeando los límites del *shallow copy*

Armar un objeto `x` tal que si defino `y = {...x}` y hago algún cambio "dentro" de `x` (o sea, hago `x.<cosas> = <nuevoValor>`), se modifica también algo en `y`.

Revolviendo un objeto en forma genérica

Definir una función que dado un objeto, devuelva los keys cuyo value asociado es 0. P.ej.

```
> keysForZero({x:4,y:0,z:1,w:9,f:0})
['y', 'f']
```

A mí me salió con una expresión, o sea

```
function keysForZero(obj) {
    return <expresion>
}
```

usando `Object.entries` y métodos de array.
Un detalle: un array de dos posiciones se puede desarmar con un pattern de la forma `[a,b]`.

Rearmando un objeto en forma genérica

Definir una función que, dado un objeto cuyos valores son todos numéricos, devuelve un objeto con las mismas keys, y cada value el doble del value original. P.ej.

```
> doubledObject({x:4,y:0,z:1,w:9,f:0})
{x:8,y:0,z:2,w:18,f:0}
```

Otra vez, más desafío si sale con una expresión. A mí me salió usando `Object.values` más estas dos cosas:

- si tengo p.ej. `someKey = 'a'`, entonces `{[someKey]: 4}` es el objeto `{a: 4}`.
- ver qué devuelve `Object.assign({}, ...[{a:5},{b:8}])`

De object literals a clases

Empecemos viendo qué pasa si el valor de un atributo es una *función*.

```
let windowSpec = {
    height: 200,
    width: 150,
    area: function() { return this.height * this.width }
}
```

A veeeer

```
> windowSpec.area
[Function: area]
> windowSpec.area()
30000
```

Puedo **ejecutar** la función, y obtener el valor de los atributos con `this.<attrName>`.

Comentario:
Esto pasa con *cualquier* referencia a función, p.ej.

```
> const double = function(n) { return n * 2 }
undefined
> double
[Function: double]
> double(4)
8
```

Demos un paso más: definamos una función *que devuelva* un objeto con atributos "mixtos" (algunos funciones, otros no).

```
function WindowSpecFn(h,w) {
  return {
```

```
    height: h,
    width: w,
    area: function() { return this.height * this.width }
  }
}
```

ya tenemos casi una clase

```
> let spec1 = WindowSpecFn(50,20)
undefined
> spec1
{ height: 50, width: 20, area: [Function: area] }
> spec1.height
50
> spec1.area()
1000
```

En rigor, la definición de clases en JavaScript es un syntax sugar de algo parecido a la definición de `WindowSpecFn`. Antes de ES6 que agregó la sintaxis de `class`, ya se podían definir clases. A mí me salió esto, que funciona …

```
function WindowSpec(h,w) {
  this.height = h
  this.width = w
}

WindowSpec.prototype.area = function() {
  return this.height * this.width
}
```

… sin que yo termine de entender qué es eso del "prototipo".

Última modificación: viernes, 30 de agosto de 2024, 07:45