Objetos en Javascript

Objectos literales

Compañeros de ruta de todo JavaScriptero.

```
let windowSpec = {
    height: 200,
    width: 150
}
```

Puedo pedir un atributo, puedo cambiar valores. En principio son abiertos, puedo agregar lo que quiera. Si pido un atributo que no tiene definido, obtengo `undefined`. También puedo pensar en un objeto como un mapa (... que es, creo, como lo piensa JavaScript ...), por lo tanto pedirle los keys y values.

```
> windowSpec.height
200
> windowSpec.height = 100
100
> windowSpec.color = 'blue'
'blue'
> windowSpec
{ height: 100, width: 150, color: 'blue' }
> windowSpec.preferredPhilosopher
undefined
> Object.keys(windowSpec)
['height', 'width', 'color']
> Object.values(windowSpec)
[ 100, 150, 'blue' ]
```

También está la notación `<objeto>[<atributo>]` que permite obtener un atributo sin fijar el nombre

```
let attrNamePrefix = 'widt'
windowSpec[attrName + 'h']
```

Referencias

Si hago

```
let windowSpec = { height: 200, width: 150 }
let otherSpec = windowSpec
const thirdSpec = windowSpec
```

los tres identificadores hacen referencia *al mismo objeto*. Probar qué pasa con cualquiera de los tres si después se hace

```
windowSpec.height = 100
```

(*pregunta*: ¿qué pasa si hago `windowSpec = 100`, también cambian todos?)

Distinta es la cosa si hacemos

```
let otherSpec = {...windowSpec}
```

porque se está generando un *clon* de `windowSpec`. Parece que los "tres-puntos" tienen una variedad de usos. Este es un syntax sugar para `Object.assign()`, o sea, un *shallow copy*. Ver la diferencia entre shallow copy y deep copy.

Identidad e igualdad

La diferencia entre referencias-al-mismo-objeto y clones, se puede testear con los operadores `===` y `==`. El primero sólo da `true` para referencias-al-mismo-objeto, el segundo también da `true` para clones.
Probar `windowSpec === otherSpec` y `windowSpec == otherSpec` con las dos definiciones de `otherSpec` que dimos.

Para ir cerrando

Los "tres-puntos" permiten mergear varios objetos, y también agregar/modificar valores.

```
let point = {x:8, y:12, z:-4}
let otherSpec = {...windowSpec, ...point, width:75, borderWidth:4}
```

Terminamos esta parte mostrando una variante sintácticamente parecida, pero con un efecto muy distinto. Es esto

```
let otherSpec = { windowSpec }
```

que es simplemente una abreviatura para `{ windowSpec: windowSpec }`.

(*pregunta*: si con esta definición cambio `windowSpec.height` ¿cambia algo en `otherSpec`?)

Comentario:
Lo de las referencias compartidas y los "tres-puntos" corre también para *arrays*.
De hecho ... **los arrays son objetos**, probar `Object.keys(['a', 'b', 'c'])`, notar la similitud entre `windowSpec['width']` y `someArray[1]`.

Última modificación: viernes, 30 de agosto de 2024, 07:41