Mutabilidad

Objetivos:

- Conocer el concepto de mutabilidad.
- Duplicar objetos antes de operar con ellos.

Motivación

En este capítulo estudiaremos como trabajar con un array cuando necesitamos transformar sus datos pero además necesitamos guardar los originales.

Este tipo de situaciones se dan en el trabajo diario, por ejemplo si tenemos un arreglo de datos con precios y queremos hacer distintas simulaciones, una aumentando los precios de los primeros productos, luego otra aumentando los últimos y finalmente una aumentando todos.

Para resolver esto tendremos que conocer el concepto **mutabilidad**.

Introducción a mutabilidad

En lugar de partir explicando el problema con el caso del arreglo de precios y tener que resolver con ciclos reduciremos el problema a la situación más sencilla.

```
a = ["hola", "yo", "soy", "un", "arreglo"]
b = a
b[0] = "nos vemos"
puts a[0] #nos vemos
```

Cambiamos parte del arreglo guardado en la variable `b` pero sin intención también cambiamos el arreglo dentro de la variable `a`, y esto sucede porque ambos son el mismo arreglo.

En este capítulo estudiaremos este fenomeno y como prevenirlo.

Mutabilidad

La mutabilidad quiere decir que un objeto puede cambiar después de que fue creado.

```
a = ["hola", "yo", "soy", "un", "arreglo"] # aquí lo creamos
b = a
b[0] = "nos vemos" # aquí lo modificamos
puts a[0] #nos vemos
```

Asignación vs modificación

Asignar un nuevo objeto es distinto a crear un objeto

```
a = 8
a = 10 # Aquí estamos asignando el objeto 10

a = [0,1,2,3,4,5]
a[0] = 100 # Aquí estamos modificando el array
```

Mutabilidad y string

Los strings en ruby también son mutables.

```
a = "hola"
a[0] = "H"
puts a => "Hola"
```

En cambios los Integers y los Floats no son mutables, o sea no hay forma de cambiar el valor al número 2 sin que sea otro número.

En resumen hasta ahora

- Existen objetos que pueden cambiar se llaman mutables.
- Existen objetos que no pueden cambiar se llaman inmutables.
- La mutabilidad es peligrosa si no tenemos cuidado porque podemos cambiar una variable por error.

¿Cómo podemos saber si son el mismo objeto o no?

Copiando un arreglo para evitar cambios indeseados

Para esto ocuparemos el método `.dup`

```
a = [1, 2, 3, 4]
b = a.dup
a[0] = 100
print a #[100, 2, 3, 4]
print b [1, 2, 3, 4]
```