Operaciones típicas en un hash

Objetivos

- Eliminar un elemento dentro de un hash.
- Unir dos hashes en uno solo.
- Invertir un hash.
- Transformar las claves del hash en un arreglo.
- Transformar los valores del hash en un arreglo.

Introducción

En este capítulo aprenderemos varias funcionalidades de los hashes que nos harán la vida más fácil. Todos los métodos que aprenderemos los podemos consultar desde la documentación.

Eliminar un elemento dentro de un hash.

Podemos eliminar una llave con su valor del hash ocupando el método `.delete`

```
a = {k1: 5, k2: 7}
a.delete(:k1)
print a
```

- Al igual que cuando accedemos al valor cuando borremos la clave debemos ser consientes de si es un símbolo o un string.
- El método delete además devuelve el valor borrado en caso de que necesitemos trabajar con él.

Uniendo hashes

Otra operación muy común es unir dos hashes, para eso ocuparemos el método merge.

```
a = {k1: 5, k2: 10}
b = {k3: 15, k4: 20}
a.merge(b)
```

Colisiones

Cuando dos hashes tienen la misma llave el segundo sobreescribe al primero

```
a = {k1: 5, k2: 10}
b = {k2: 15, k3: 20}
a.merge(b)
```

Las operaciones `a.merge(b)` y `b.merge(a)` entregan resultados distintos

Invirtiendo un hash

En algunas ocasiones puede ser útil invertir un hash para obtener una llave a partir de un valor

Por ejemplo supongamos que tenemos un hash con colores y sus valores hexadecimales Y queremos obtener el color a partir del código.

```
colors = {'red' => '#cc0000', 'green' => '#00cc000', 'blue' => '#0000cc' }
colors.invert['#cc0000'] #red
```

Obtener todas las claves de un hash

Lo podemos hacer de forma muy sencilla con `.keys`

```
colors = {'red' => '#cc0000', 'green' => '#00cc000', 'blue' => '#0000cc' }
colors.keys
```

Obtener todas los valores de un hash

Existe un método específico para esto llamado `.values`

```
colors = {'red' => '#cc0000', 'green' => '#00cc000', 'blue' => '#0000cc' }
colors.values
```

```
ventas = {
  zona_norte: [1000,500,300]
  zona_centro: [500, 500, 400]
}
```

Reconstruir un hash a partir de los keys y values

Existe un método llamado `.zip` que permite unir dos arreglos.

```
[1,2,3].zip([4,5,6])
```

Ejercicio práctico

Se tiene una base de datos de colores


```
{
  "aliceblue": "#f0f8ff",
  "antiquewhite": "#faebd7",
  "aqua": "#00ffff",
  "aquamarine": "#7fffd4",
  "azure": "#f0ffff",
  "darkorchid": "#9932cc",
  "darkred": "#8b0000",
  "darksalmon": "#e9967a",
  "navajowhite": "#ffdead",
  "navy": "#000080",
  "orchid": "#da70d6",
  "palegoldenrod": "#eee8aa",
  "peachpuff": "#ffdab9",
  "peru": "#cd853f",
  "pink": "#ffc0cb",
  "purple": "#800080",
  "rebeccapurple": "#663399",
  "red": "#ff0000",
  "saddlebrown": "#8b4513",
  "seashell": "#fff5ee",
  "sienna": "#a0522d",
  "silver": "#c0c0c0",
  "skyblue": "#87ceeb",
  "slateblue": "#6a5acd",
  "teal": "#008080",
  "thistle": "#d8bfd8",
  "tomato": "#ff6347",
  "turquoise": "#40e0d0",
  "violet": "#ee82ee",
  "wheat": "#f5deb3",
  "white": "#ffffff",
  "whitesmoke": "#f5f5f5",
  "yellow": "#ffff00",
  "yellowgreen": "#9acd32"
}
```


Se pide crear un programa llamado busqueda_colores.rb donde el usuario ingrese un color en hexadecimal y le muestra en pantalla el nombre del color, en caso de no encontrarlo aparecerá el texto no-no

Uso: busqueda_colores.rb #6a5acd #40e0d0

> slateblue

Solución iterando

Tenemos dos soluciones posibles, una sería iterar el arreglo hasta encontrar la clave, la otra sería invertir el arreglo como aprendimos y aprovechar las bondades del hash.

```
colors = {
  "aliceblue": "#f0f8ff",
  "antiquewhite": "#faebd7",
  "aqua": "#00ffff",
  "aquamarine": "#7fffd4",
  "azure": "#f0ffff",
  "darkorchid": "#9932cc",
  "darkred": "#8b0000",
  "darksalmon": "#e9967a",
  "navajowhite": "#ffdead",
  "navy": "#000080",
  "orchid": "#da70d6",
  "palegoldenrod": "#eee8aa",
  "peachpuff": "#ffdab9",
  "peru": "#cd853f",
  "pink": "#ffc0cb",
  "purple": "#800080",
  "rebeccapurple": "#663399",
  "red": "#ff0000",
  "saddlebrown": "#8b4513",
  "seashell": "#fff5ee",
  "sienna": "#a0522d",
  "silver": "#c0c0c0",
  "skyblue": "#87ceeb",
  "slateblue": "#6a5acd",
  "teal": "#008080",
  "thistle": "#d8bfd8",
  "tomato": "#ff6347",
  "turquoise": "#40e0d0",
  "violet": "#ee82ee",
  "wheat": "#f5deb3",
  "white": "#ffffff",
  "whitesmoke": "#f5f5f5",
  "yellow": "#ffff00",
  "yellowgreen": "#9acd32"
}

#search = ARGV[0]

search = '#6a5acd'
colors.each do |k,v|
  if v == search
    puts k
  end
end
```

Si bien esta solución muestra en pantalla el valor correcto no maneja la situación de cuando no lo encuentra. Podríamos estar tentados a poner esto:

```
#search = ARGV[0]
search = '#6a5acd'
colors.each do |k,v|
  if v == search
    puts k
  else
    puts "no-no"
  end
end
```

El problema de este acercamiento es que vamos a mostrar muchas veces el texto de no encontrado y solo debemos mostrarlo una vez. Para evitar esto ocuparemos una variable para guardar cuando encontremos el valor y preguntaremos por ella al final.

```
search = '#6a5acd'
founded = false
colors.each do |k,v|
  if v == search
    founded = true
    puts k
  end
end

puts "no-no" unless founded
```

Solución invirtiendo el arreglo

```
match = colors.invert[search]
if match
  puts match
else
  puts "no-no"
end
```

Solución elegante

```
match = colors.invert[search]
puts match ? match : "no-no"
```

Discutamos las soluciones

¿Por qué estudiamos varias soluciones a un problema si con una basta?

Esta pregunta es muy importante, un programador no busca memorizar la respuesta a los problemas, de hecho muy rara vez nos toca resolver exactamente el mismo problemas dos veces, pero el acercamiento a resolver un problema si puede ser reutilizado en situaciones similares.

Lo importante es conocer diversas estrategias que nos permitan después enfrentar problemas nuevos.

En esta ocasión aprendimos que invertir un hash puede ser mucho más fácil que iterarlo.

Ejercicio

Modificar el programa anterior para que el usuario pueda buscar múltiples colores.

Uso: busqueda_colores.rb #6a5acd #9acd32 #ffff00

> slateblue yellowgreen yellow

Tip: El usuario puede ingresar cuantos colores desee.

Discutamos primero la forma incorrecta de resolverlo

```
search1 = ARGV[0]
search2 = ARGV[1]
search3 = ARGV[2]

match = colors.invert[search1]
puts match ? match : "no-no"

match = colors.invert[search2]
puts match ? match : "no-no"

match = colors.invert[search3]
puts match ? match : "no-no"
```

Es una pésima solución porque si el usuario ingresa mas valores no funcionará. Otra forma de darse cuenta de que es mal código es porque estamos repitiendo el mismo código varias veces.

```
## Solución buena

# Simplemente tenemos que iterar el arreglo.
# ARGV = ['#6a5acd', '#6a5acd']

ARGV.each do |search|
  match = colors.invert[search]
  puts match ? match : "no-no"
end
```