

SWIFT  
Y RUBY

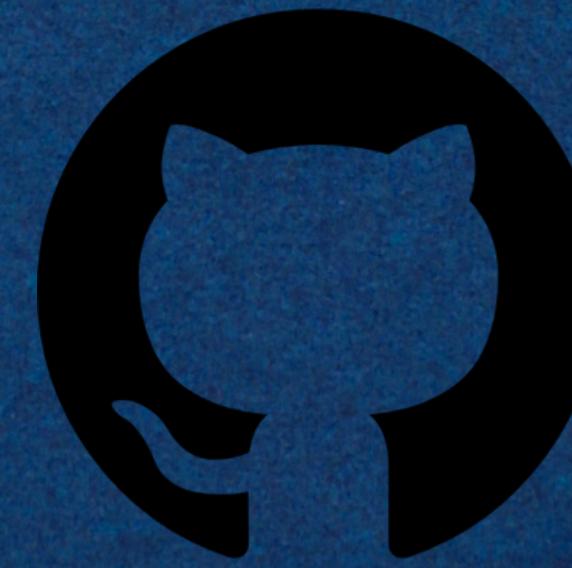
# RUBY

*"A menudo la gente, especialmente los ingenieros en computación, se centran en las máquinas. Ellos piensan, 'Haciendo esto, la máquina funcionará más rápido. Haciendo esto, la máquina funcionará de manera más eficiente. Haciendo esto...' Están centrados en las máquinas, pero en realidad necesitamos centrarnos en las personas, en cómo hacen programas o cómo manejan las aplicaciones en los ordenadores. Nosotros somos los jefes. Ellos son los esclavos."*

*O'Reilly Media en 2003*



*Yukihiko Matsumoto*



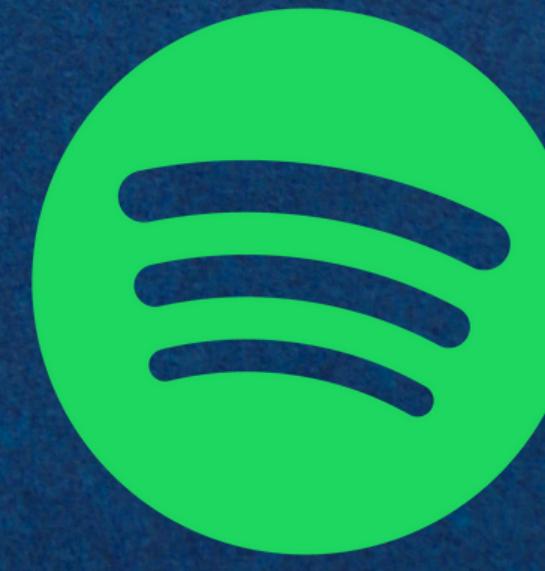
# SWIFT

*"Swift está diseñado para facilitar la escritura y el mantenimiento de programas correctos para el desarrollador. Incluye características como la inferencia de tipos, los opcionales y las clausuras que reducen la probabilidad de errores en el código y hacen que sea más fácil expresar ideas complejas de manera clara y concisa."*

*The Next Web en 2014*



Chris Lattner



# FUNCIONES DE ORDENAMIENTO RUBY

## SORT

Devuelve un nuevo array con los elementos ordenados

```
numbers = [3, 1, 4, 1, 5, 9]
sorted_numbers = numbers.sort
# sorted_numbers => [1, 1, 3, 4, 5, 9]
```

## SORT!

Ordena el array en el lugar, modificando el original.

```
numbers = [3, 1, 4, 1, 5, 9]
numbers.sort!
# numbers => [1, 1, 3, 4, 5, 9]
```

## SORT\_BY

Ordena según criterio. Devuelve un nuevo array ordenado

```
words = ["apple", "banana", "cherry"]
sorted_by_length = words.sort_by { |word| word.length }
# sorted_by_length => ["apple", "banana", "cherry"]
```

En Ruby, el método sort y sort! no garantizan estabilidad, mientras que sort\_by lo hace

Tanto la “b” como la “d” tienen el mismo valor:

```
items = [
  { value: 3, label: 'a' },
  { value: 1, label: 'b' },
  { value: 4, label: 'c' },
  { value: 1, label: 'd' },
  { value: 5, label: 'e' },
  { value: 9, label: 'f' }
]

# Ordenar usando sort
sorted_items = items.sort { |a, b| a[:value] <=> b[:value] }
puts sorted_items.inspect

# Salida: [{:value=>1, :label=>"d"}, {:value=>1, :label=>"b"},
```

usando sort, en la salida aparece primero la “d”,

```
items = [
  { value: 3, label: 'a' },
  { value: 1, label: 'b' },
  { value: 4, label: 'c' },
  { value: 1, label: 'd' },
  { value: 5, label: 'e' },
  { value: 9, label: 'f' }
]

# Ordenar usando sort_by
sorted_items_by = items.sort_by { |item| item[:value] }
puts sorted_items_by.inspect

# Salida: [{:value=>1, :label=>"b"}, {:value=>1, :label=>"d"},
```

usando sort\_by, en la salida aparece primero la “b”,

# FUNCIONES DE ORDENAMIENTO SWIFT

## SORTED()

Devuelve un nuevo array con los elementos ordenados

```
let numbers = [3, 1, 4, 1, 5, 9]
let sortedNumbers = numbers.sorted()
// sortedNumbers => [1, 1, 3, 4, 5, 9]
```

## SORT()

Ordena el array en el lugar, modificando el original.

```
var numbers = [3, 1, 4, 1, 5, 9]
numbers.sort()
// numbers => [1, 1, 3, 4, 5, 9]
```

## SORTED(BY:)

Ordena según criterio. Devuelve un nuevo array ordenado

```
let words = ["apple", "banana", "cherry"]
let sortedByLength = words.sorted { $0.count < $1.count }
// sortedByLength => ["apple", "banana", "cherry"]
```

## En Swift, cualquier función de ordenamiento es estable desde Swift 5.0

```
// Ordenar usando sorted(by:)  
let sortedItemsBy = items.sorted { (a, b) -> Bool in  
    let valueA = a["value"] as! Int  
    let valueB = b["value"] as! Int  
    return valueA < valueB  
}  
print("Sorted with sorted(by): \(sortedItemsBy)")
```

```
// Ordenar usando sort() (en el lugar)  
var itemsToSort = items  
itemsToSort.sort { (a, b) -> Bool in  
    let valueA = a["value"] as! Int  
    let valueB = b["value"] as! Int  
    return valueA < valueB  
}  
print("Sorted with sort(): \(itemsToSort)")
```

```
Sorted with sorted(by): [[{"value": 1, "label": "b"}, {"value": 1, "label": "d"}], [{"value": 1, "label": "b"}, {"value": 1, "label": "d"}]]  
Sorted with sort(): [[{"value": 1, "label": "b"}, {"value": 1, "label": "d"}], [{"value": 1, "label": "b"}, {"value": 1, "label": "d"}]]
```

En este ejemplo, ambas salidas colocaron primero la “b”, respetando el orden

# COMPARACIÓN DE RENDIMIENTO

## RUBY

- Timsort
- `sort!` modifica la colección en el lugar
- `sort_by` - clave específica
- Promedio  $O(n \log n)$

## SWIFT

- Quicksort
- `sort()` modifica la colección en el lugar
- `sorted(by:)` - clave específica
- Promedio  $O(n \log n)$

# TIEMPO DE RESPUESTA

Ordenar un millón de números aleatorios:

```
~/utn/ssl (8.859s)
swift taylor.swift

Time for sorted(): 2.0272369384765625 seconds
Time for sort(): 2.027316927909851 seconds
Time for sorted(by:): 1.9660300016403198 seconds
```

```
~/utn/ssl (0.551s)
ruby ruby.rb

Time for sort: 0.15080800000578165 seconds
Time for sort!: 0.1439059999975143 seconds
Time for sort_by: 0.06076100000063889 seconds
```

# CONCLUSIÓN

Si la estabilidad es crítica, tanto Swift (cualquiera) como Ruby (`sort_by`) son adecuados. Para rendimiento general y simplicidad en el ecosistema Apple, Swift es preferido. Para flexibilidad y scripting, especialmente en desarrollo web, o mayor celeridad en el tiempo de respuesta, Ruby es una excelente opción.

MUCHAS  
**GRACIAS**