Trabajo práctico N° 2

Ruby y Swift

Migueles, Peralta, Perez Erro y Suarez Pujol

Ruby:

Ruby es un lenguaje de programación interpretado, dinámico, orientado a objetos, y de propósito general. Fue creado por el programador japonés Yukihiro "Matz" Matsumoto en la década de 1990. Matsumoto comenzó a desarrollar Ruby en 1993 mezclando partes de sus lenguajes favoritos existentes en ese momento, como Perl y Python. Su filosofía principal era diseñar un lenguaje que optimizara la productividad y la satisfacción del programador, priorizando que la programación sea lo más natural y expresiva posible

"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

Ruby fue lanzado oficialmente en 1995. Desde su lanzamiento, ha ganado popularidad, especialmente en el desarrollo web, gracias a la creación del framework Ruby on Rails en 2004, que facilitó la construcción rápida de aplicaciones web robustas y escalables

Dato a color: Aunque Ruby no es tan popular como algunos otros lenguajes de programación en la actualidad, sigue siendo utilizado por varias compañías modernas, especialmente aquellas que adoptaron Ruby on Rails para el desarrollo web. Algunas de las compañías más conocidas que usan Ruby son: Airbnb, GitHub, Twitch y SoundCloud

Swift:

Swift es un lenguaje de programación compilado, estático, multi-paradigma, y de propósito general. Fue creado por Apple y anunciado públicamente en la conferencia WWDC (Worldwide Developers Conference) en junio de 2014.

Su desarrollo comenzó en 2010, liderado por Chris Lattner. El objetivo era modernizar la programación en las plataformas de Apple, ofreciendo un lenguaje más seguro, rápido y fácil de usar en comparación con Objective-C, que había sido el principal lenguaje para el desarrollo de aplicaciones en iOS y macOS durante muchos años.

"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

Desde su lanzamiento, Swift ha ganado rápidamente popularidad y ha sido adoptado como el lenguaje principal para el desarrollo de aplicaciones en el ecosistema de Apple. En 2015, Apple hizo de Swift un proyecto de código abierto, permitiendo que la comunidad global contribuyera a su evolución.

Entre las compañías que utilizan Swift, además de Apple, se encuentran: LinkedIn, Twitter, WhatsApp y Spotify

1. Funciones de ordenamiento:

Ruby

- sort:

Descripción: Devuelve un nuevo array con los elementos ordenados en forma ascendente.

Sintaxis: array.sort

Ejemplo:

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

- sort! :

Descripción: Ordena el array en el lugar, modificando el array original.

Sintaxis: array.sort!

Ejemplo:

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

- sort_by:

Descripción: Ordena los elementos del array según el resultado de una función o bloque que se pasa como argumento. Devuelve un nuevo array ordenado.

Sintaxis: array.sort_by { |element| ... }

Ejemplo:

```ruby
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. Esto significa que pueden cambiar el orden relativo de elementos con claves iguales, mientras que sort_by lo mantiene.

```ruby
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"},
```

Tanto la “b” como la “d” tienen el mismo valor. Si ordenamos según el valor, usando sort_by, en la salida aparecerá primero b, y después d, porque mantiene el orden.

Sin embargo, si ordenamos el mismo ejemplo utilizando sort, aparece primero la d:

```ruby
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"},
```

Swift

- sorted():

Descripción: Devuelve un nuevo array con los elementos ordenados en orden ascendente.

Sintaxis: array.sorted()

Ejemplo:

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

- sort():

Descripción: Ordena el array en el lugar, modificando el array original.

Sintaxis: array.sort()

Ejemplo:

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

- sorted(by:)

Descripción: Devuelve un nuevo array ordenado según el criterio especificado por un cierre que compara dos elementos.

Sintaxis: array.sorted { $0 < $1 }

Ejemplo:

```swift
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. Esto significa que, cuando se ordenan elementos con claves iguales, su orden relativo en la secuencia original se mantiene.

```swift
// Definir un array de diccionarios
var items: [[String: Any]] = [
    ["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 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"],
Sorted with sort(): [["value": 1, "label": "b"], ["value": 1, "label": "d"], ["val
```

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

Comparación de Rendimiento de Funciones de Ordenamiento en Ruby y Swift

A la hora de evaluar que método conviene elegir, se tienen en cuenta varios factores, como el algoritmo de ordenación, la complejidad en O, el consumo de memoria, la facilidad de uso, su flexibilidad y su comportamiento con datos duplicados (de este último ya hablamos un poco)

Rendimiento en Ruby:

Utiliza Timsort: un algoritmo que está diseñado para aprovechar las características del ordenamiento parcial. Funciona dividiendo la lista en "runs" (secuencias ordenadas) y luego fusionándolas.

sort! modifica la colección en el lugar, lo cual puede ser ventajoso para colecciones grandes si se quiere evitar la creación de una nueva colección.

sort_by es útil cuando se necesita ordenar según una clave específica.

Rendimiento en Swift:

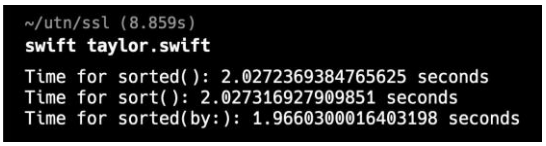
Utiliza Quicksort modificado: un algoritmo basado en la estrategia de "divide y vencerás". Selecciona un "pivote" y reorganiza los elementos alrededor de este pivote, de modo que los elementos menores que el pivote estén a la izquierda y los mayores estén a la derecha. Luego, se aplica el mismo proceso recursivamente a las sublistas.

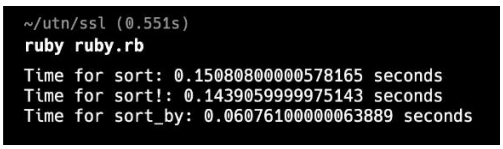
sort() modifica la colección original en el lugar, lo que puede ser más eficiente en términos de memoria si se desea evitar la creación de una nueva colección.

sorted(by:) permite especificar un criterio de ordenación personalizado.

Tanto los métodos de ordenamiento de Ruby como los de Swift, tienen un Promedio O(n log n), el cual se refiere a cómo crece el tiempo de ejecución del algoritmo con respecto al tamaño de la entrada. La notación O(n log n) para la complejidad promedio indica que el algoritmo es eficiente y escalable para entradas grandes, con un crecimiento del tiempo de ejecución que es manejable en comparación con algoritmos menos eficientes.

Por último, medimos el tiempo que le lleva a cada método ordenar 1.000.000 de números aleatorios





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

Link Kahoot!: https://create.kahoot.it/details/8c0bdd88-cc39-4d5f-812b-39abc0e50847