****

**Trabajo Practico Teorico**

**Haskell y Python**

**MATERIA:** Sintaxis Y Semántica de los Lenguajes.

**PROFESORA:** Roxana María Leituz.

**COMISIÓN:** K2055.

**ALUMNOS:** Aylén Marta Chazarreta, Brisa Nelly Calzado, Dante Silva, Enrique Antonio Marques y Lisandro Corrales Cespedes.

**FECHA DE ENTREGA:** 9/8/2024.

**BNF’s**

* Haskell <https://www.haskell.org/onlinereport/syntax-iso.html>
* Python <https://docs.python.org/3/reference/grammar.html>

**Historia y Descripción de Haskell**

Haskell es un lenguaje de programación funcional puro, de propósito general, compilado, fuertemente tipificado, polimórfico y que utiliza evaluación perezosa.

Durante los años 1980, la comunidad de lenguajes funcionales estaba fragmentada, con muchos lenguajes diferentes y poca estandarización, por lo que se decidió crear un nuevo lenguaje de programación funcional estándar. En 1987 se formó un comité para desarrollar este nuevo lenguaje, que eventualmente se llamaría Haskell, con la idea de consolidar los lenguajes existentes en uno único que sirviera como base para la investigación futura en diseño de lenguajes.  
 Su primera versión nació en el año 1990: Haskell 1.0 con las características básicas y en la actualidad está disponible Haskell 2010 con mejoras y nuevas características basadas en la experiencia de versiones anteriores.

**Historia y Descripción de Python**

Python, desarrollado a finales de los años ochenta por el programador holandés Guido van Rossum, es un lenguaje de programación de propósito general multiparadigma (permite varios estilos de programación tales como programación orientada a objetos, programación imperativa y programación funcional), dinámicamente tipado, que combina características de lenguajes interpretados y compilados.  
 La filosofía de Python se centra en la claridad y legibilidad del código.

Versiones:

* Python 1.0 (1994): Introduce herramientas como lambda, map, filter y reduce.
* Python 2.0 (2000): Introduce nuevas características como el recolector de basura de ciclos , la compresión de listas, las asignaciones aumentadas , los nuevos métodos de cadenas y las cadenas Unicode.
* Python 3.0 (2008): introdujeron la separación entre cadenas Unicode y datos binarios, la función print(), cambios en la sintaxis , tipos de datos, comparadores, etc.
  + Python 3.12 (2023): introdujo mejoras de rendimiento significativas y nuevas características del lenguaje.

**Funciones de Ordenamiento de Python**

* Métodos principales: sort() y sorted()
* Algoritmo utilizado: Timsort
  + El algoritmo Timsort utilizado en Python realiza múltiples ordenamientos de manera eficiente porque puede aprovechar cualquier orden ya presente en el conjunto de datos. Timsort es una mezcla de Insertion Sort y Mergesort. Las partes más pequeñas se ordenan mediante Insertion Sort y luego se fusionan mediante Mergesort.
* Descripción
  + sort(): Este método ordena la lista internamente modificando la lista original. Si alguna comparación falla, la operación entera falla y la lista queda ordenada parcialmente.
  + sorted(): Crea una nueva lista ordenada a partir de un iterable (listas, tuplas, cadenas, etc.), sin modificar el original.
* Sintaxis:
  + lista.sort (key=None, reverse=False)
    - Key: criterio para realizar la comparación. Es opcional, y el valor predeterminado es None que hace que la lista se ordene comparando directamente los elementos, sin obtener valores clave.
    - Reverse: es un valor booleano que sirve para indicar si se desea ordenar la lista de manera descendente (True) o ascendente (False). El valor predeterminado es False.
  + sorted (iterable, key=None, reverse=False)
    - Iterable: todos los elementos que se quieren ordenar.
* Ejemplos:

| **sort** | **sorted** |
| --- | --- |
| palabras = ["manzana", "kiwi", "platano", "fresa"]  palabras.sort(key=len, reverse=True)  print(palabras)  # Imprime ['manzana', 'platano', 'fresa', 'kiwi'] | student\_tuples = [('john', 'A', 15), ('jane', 'B', 12), ('dave', 'B', 10)]  print(sorted(student\_tuples, key=itemgetter(2), reverse=True))  # Imprime [('john', 'A', 15), ('jane', 'B', 12), ('dave', 'B', 10)] |

En resumen:

| **Feature** | **.sort()** | **sorted()** |
| --- | --- | --- |
| Tipo | Es un método | Es una función |
| Comportamiento | Ordena "In-Place" | Retorna una nueva lista ordenada |
| Lista Original | Modificada | Intacta |
| Parámetros | Solo acepta listas | Acepta cualquier elemento iterable |
| Performance | Más rápido para listas largas | Más lento para listas largas |
| Reverso | Con reverse = true | Con reverse = true |
| Múltiples Keys | No acepta | Acepta |

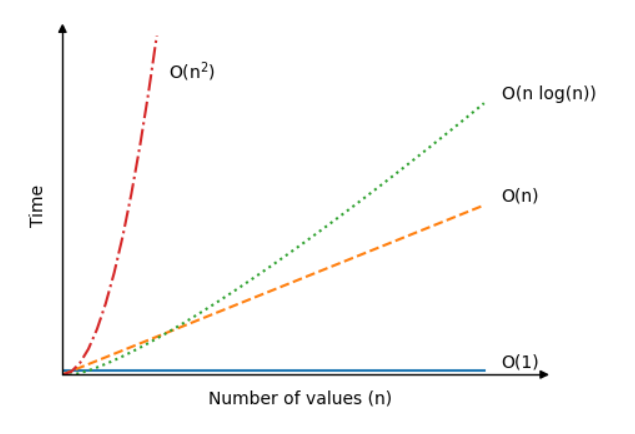
**Funciones de Ordenamiento de Haskell**

* Métodos principales: sort, insert, sortOn
* Descripción:
  + sort: Devuelve una lista con los elementos organizados de menor a mayor, manteniendo los duplicados en el orden en que aparecieron en la entrada. El argumento debe ser finito.
  + insert: Toma un elemento y una lista e inserta el elemento en la lista en la primera posición donde es menor o igual que el siguiente elemento
  + sortOn: Ordena una lista utilizando los resultados de una función clave aplicada a cada elemento. La función clave se evalúa sólo una vez por cada elemento en la lista de entrada, lo que mejora el rendimiento.
* Sintaxis:
  + sort :: [Ord](https://hackage.haskell.org/package/base-4.20.0.1/docs/Data-Ord.html#t:Ord) a => [a] -> [a]
  + insert :: [Ord](https://hackage.haskell.org/package/base-4.20.0.1/docs/Data-Ord.html#t:Ord) a => a -> [a] -> [a]
  + sortOn :: [Ord](https://hackage.haskell.org/package/base-4.20.0.1/docs/Data-Ord.html#t:Ord) b => (a -> b) -> [a] -> [a]
    - Las listas deben contener elementos que sean miembro de clase de tipos Ord, es decir que sean ordenables.

Ejemplos

| **sort** | **insert** | **sortOn** |
| --- | --- | --- |
| sort [1,6,4,3,2,5]  [1,2,3,4,5,6] | insert (-1) [1, 2, 3]  [-1,1,2,3] | sortOn fst [(2, "world"), (4, "!"), (1, "Hello")]  [(1,"Hello"),(2,"world"),(4,"!")] |
| sort "haskell"  "aehklls" | insert 'd' "abcefg"  "abcdefg" | sortOn length ["jim", "creed", "pam", "michael", "dwight", "kevin"]  ["jim","pam","creed","kevin","dwight","michael"] |

**Rendimiento**

Ambos lenguajes utilizan la notación Big-O, la cual es una notación asintótica que se usa para expresar la complejidad de un algoritmo o su rendimiento en función del tamaño de entrada (n).

La función *insert* de Haskell posee una complejidad lineal, O(n), ya que su tiempo de ejecución aumenta linealmente con el tamaño de la entrada. Por otro lado, el resto de funciones mencionadas, tanto de Haskell como de Python poseen una complejidad O(n log(n)) en el peor de los casos, es decir, cuando la lista se encuentra completamente desordenada y una complejidad O(n) en el mejor de los casos, es decir, cuando el desorden es mínimo.

| **Python** | **Haskell** |
| --- | --- |
| sort([5, 20, 14, 15, 16, 89, 57, 6, 7, 8])  Dividir en runs:  [5, 20], [14, 15, 16], [89, 57], [6, 7, 8]  Insertion Sort para ordenar cada run  [5, 20], [14, 15, 16], [57, 89], [6, 7, 8]  MergeSort para fusionar los runs  [5, 14, 15, 16, 20], [57, 89], [6, 7, 8]  [5, 14, 15, 16, 20, 57, 89], [6, 7, 8]  [5, 6, 7, 8, 14, 15, 16, 20, 57, 89] | sort [6, 5, 3, 1, 8, 7, 2, 4]  Dividir lista de a mitades hasta que quedan listas de un único elemento  [6, 5, 3, 1], [8, 7, 2, 4]  [6, 5], [3, 1], [8, 7], [2, 4]  [6], [5], [3], [1], [8], [7], [2], [4]  Ordenar las sublistas individuales  [6], [5] → [5, 6]  [3], [1] → [1, 3]  [8], [7] → [7, 8]  [2], [4] → [2, 4]  Fusionar las Sublistas  [5, 6], [1, 3] → [1, 3, 5, 6]  [7, 8], [2, 4] → [2, 4, 7, 8]  Fusionar las dos listas  [1, 2, 3, 4, 5, 6, 7, 8]  insert 4 [2, 3, 5, 8]  4 <= 2 → False → avanza al siguiente elemento  4 <=3 → False → avanza al siguiente elemento  4 <=5 → True→ inserta el 4 adelante del 5  [2, 3, 4, 5, 8] |

**PPT:**

<https://docs.google.com/presentation/d/1mawvUrOZATDjkZlSlNVnS-i1ttp_Xmusw1vqYibuWug/edit?usp=sharing>

**REFERENCIAS**

ThePower Business School. (s. f.). \*Haskell, el lenguaje de programación\*. ThePower Education. Recuperado de

<https://thepower.education/blog/haskell-lenguaje-de-programacion>

Bambu Mobile. (s. f.). \*¿Para qué se usa Haskell?\*. Bambu Mobile. Recuperado de <https://bambu-mobile.com/para-que-se-usa-haskell/>

Python Software Foundation. (s. f.). \*Sorting HOW TO\*. Python.org.

<https://docs.python.org/3/howto/sorting.html>

Python Software Foundation. (s. f.). \*Cómo ordenar\*. Python.org. <https://docs.python.org/es/3/howto/sorting.html>

Python Software Foundation. (s. f.). \*list.sort\*. Python.org. <https://docs.python.org/es/3/library/stdtypes.html#list.sort>

Ronny. (2007, 11 de diciembre). \*Algoritmos de ordenación Haskell\*. Ronny M.L. <https://ronnyml.wordpress.com/2007/12/11/algoritmos-de-ordenacion-haskell/>

Haskell.org. (s. f.). \*Data.List: sort\*. Hackage. <https://hackage.haskell.org/package/base-4.20.0.1/docs/Data-List.html#v:sort>

ThePower Business School. (s. f.). \*Key differences entre Sort() y Sorted() en Python\*. Naukri.com. <https://www.naukri.com/code360/library/difference-between-sort-and-sorted-in-python>

W3Schools. (s. f.). \*Time Complexity\*. W3Schools. <https://www.w3schools.com/dsa/dsa_timecomplexity_theory.php>

Python Software Foundation. (2023). \*What's new in Python 3.12\*. Python.org. <https://docs.python.org/3.12/whatsnew/3.12.html>

Pérez, J. (s. f.). \*Historia de Python\*. mclibre.org. Recuperado de <https://www.mclibre.org/consultar/python/otros/historia.html#:~:text=La%20versi%C3%B3n%201.0%20se%20public%C3%B3,la%20fundaci%C3%B3n%20Python%20Software%20Foundation>.

Torres Andrés (2021). \*Ordenar listas en Python\*. FreeCodeCamp.org

<https://www.freecodecamp.org/espanol/news/ordenar-listas-en-python-como-ordenar-por-descendente-o-ascendente/>

Wikipedia. (s.f.). \*Timsort\*. Wikipedia, la enciclopedia libre. Recuperado el 8 de agosto de 2024, de <https://pt.wikipedia.org/wiki/Timsort>

Wikipedia. (s.f.). \*Ordenamiento por mezcla\*. Wikipedia, la enciclopedia libre. Recuperado el 8 de agosto de 2024, de <https://es.wikipedia.org/wiki/Ordenamiento_por_mezcla#:~:text=El%20algoritmo%20de%20ordenamiento%20por,O(n%20log%20n)>

.