Ordenamiento

Clase #22

IIC1103 – Introducción a la Programación

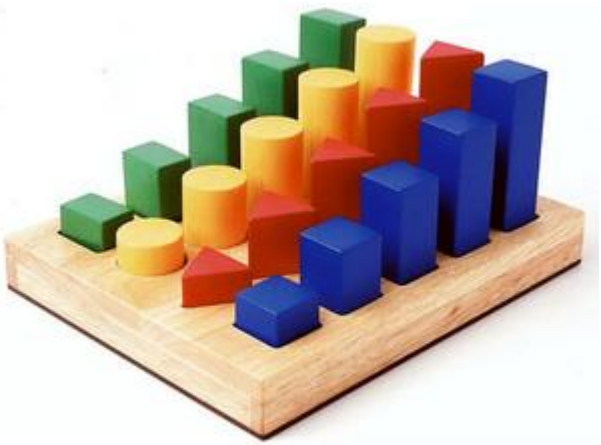
Marcos Sepúlveda (marcos@ing.puc.cl)

Veremos hoy …

- Método `sort()` de listas
- Ordenamiento por selección
- Ordenamiento por inserción
- Ordenamiento con método `sorted()`

¿Qué necesitamos ordenar?

- Registro de números de teléfono
- Notas de un curso
- Puntaje de equipos que participan en un campeonato



Ordenando Listas – usando `sort()`

- Si queremos ordenar una lista **`L`**, por ejemplo:
  - **`L = [3, 1, 70, -1, 3, 7, 5]`**
- Usando el método **`sort()`** podemos simplemente escribir:
  - **`L.sort()`**
- Y la lista nos queda ordenada (ascendentemente) :
  - **`L → [-1, 1, 3, 3, 5, 7, 70]`**
- ¿Se puede ordenar en forma descendente con **`sort()`**?
  - **`L.sort()`**
  - **`L.reverse()`**

  ó
  - **`L.sort(reverse=True)`**

Ordenando listas de enteros – usando `sort()`

```
import random

def Lista_Aleatoria(n, desde, hasta):
    L = []
    for i in range(n):
        L.append(random.randint(desde, hasta))
    return L

L = Lista_Aleatoria(10, 1, 100)
print("Lista desordenada:", *L)
L.sort()
print("Lista ordenada   :", *L, "\n---")

# ordenar en orden decreciente
L = Lista_Aleatoria(10, 1, 100)
print("Lista desordenada:", *L)
L.sort()
L.reverse()
print("Lista decreciente:", *L, "\n---")

# ó
L = Lista_Aleatoria(10, 1, 100)
print("Lista desordenada:", *L)
L.sort(reverse=True)
print("Lista decreciente:", *L, "\n---")
```

```
Lista desordenada: 57 22 91 36 95 71 71 57 77 62
Lista ordenada   : 22 36 57 57 62 71 71 77 91 95
---
```

```
Lista desordenada: 95 96 71 36 11 42 55 63 23 95
Lista decreciente: 96 95 95 71 63 55 42 36 23 11
---
```

```
Lista desordenada: 52 32 10 72 84 45 66 34 14 34
Lista decreciente: 84 72 66 52 45 34 34 32 14 10
---
```

Ordenando listas de objetos – usando `sort()`

```
class Persona:
    def __init__(self, nombre, edad):
        self.nombre = nombre
        self.edad = edad

    def __lt__(self, otro):
        # por edad
        return self.edad < otro.edad

    def __str__(self):
        return "(" + self.nombre + "," + str(self.edad) + ")"


L = [Persona("juan perez", 34), Persona("pedro lopez", 46),
     Persona("matias donoso", 18), Persona("andres jara", 21),
     Persona("jose tapia", 19), Persona("julio soto", 28)]

print("Lista desordenada:", *L, sep='\n')
print("---")

L.sort()

print("Lista ordenada:", *L, sep='\n')
print("---")
```

```
>>>
Lista desordenada:
(juan perez,34)
(pedro lopez,46)
(matias donoso,18)
(andres jara,21)
(jose tapia,19)
(julio soto,28)
---
```

```
Lista ordenada:
(matias donoso,18)
(jose tapia,19)
(andres jara,21)
(julio soto,28)
(juan perez,34)
(pedro lopez,46)
---
```

Ordenando Listas – algoritmos clásicos

- Pero si no existiera el método **`sort()`**, ¿qué haríamos?

- Idea: **Selección**
  - Buscamos el menor elemento de la lista y lo intercambiamos con el primer elemento.
  - Después buscamos el segundo menor elemento y lo intercambiamos con el segundo elemento de la lista.
  - Después buscamos el tercer menor elemento y lo intercambiamos con el tercer elemento de la lista.
  - … Y así hasta que llegamos al penúltimo elemento de la lista.

  - `L → [3, 1, 70, -1, 3, 7, 5]`
  - `L → [-1, 1, 70, 3, 3, 7, 5]`
  - `L → [-1, 1, 70, 3, 3, 7, 5]`
  - `L → [-1, 1, 3, 70, 3, 7, 5]`
  - `L → [-1, 1, 3, 3, 70, 7, 5]`
  - `L → [-1, 1, 3, 3, 5, 7, 70]`
  - `L → [-1, 1, 3, 3, 5, 7, 70]`

Algoritmos clásicos de ordenamiento – **Selección**

```
def Orden_Seleccion(lista):
    for i in range(0,len(lista)-1):
        posicion = i
        for j in range(posicion+1,len(lista)):
            if lista[j] < lista[posicion]:
                posicion = j
        auxiliar        = lista[posicion]
        lista[posicion] = lista[i]
        lista[i]        = auxiliar
    return lista

L = [ 3, 1, 70, -1, 3, 7, 5]
print("L:", L)

L_desordenado = L.copy()
L_ordenado = Orden_Seleccion(L)

print("L_desordenado:", L_desordenado)
print("L_ordenado:   ", L_ordenado)

print("L:", L)
```

```
>>>
L: [3, 1, 70, -1, 3, 7, 5]
L_desordenado: [3, 1, 70, -1, 3, 7, 5]
L_ordenado: [-1, 1, 3, 3, 5, 7, 70]
L: [-1, 1, 3, 3, 5, 7, 70]
```

Ordenando Listas – algoritmos clásicos

- Pero si no existiera el método **`sort()`**, ¿qué haríamos?

- Otra idea: **Inserción**
  - Asumimos que los primeros **i-1** elementos están ordenados.
  - Se compara el elemento **i**-ésimo con los elementos ya ordenados. Se busca dónde insertarlo: mientras los elementos sean mayores, se desplazan hacia la derecha, deteniéndose cuando se encuentra un elemento menor o cuando ya no se encuentran elementos (el elemento a ordenar es el menor). Se inserta el elemento en dicha posición.

  - **`L →[3, 1, 70, -1, 3, 7, 5]`**
  - **`L →[1, 3, 70, -1, 3, 7, 5]`**
  - **`L →[1, 3, 70, -1, 3, 7, 5]`**
  - **`L →[-1, 1, 3, 70, 3, 7, 5]`**
  - **`L →[-1, 1, 3, 3, 70, 7, 5]`**
  - **`L →[-1, 1, 3, 3, 7, 70, 5]`**
  - **`L →[-1, 1, 3, 3, 5, 7, 70]`**

Algoritmos clásicos de ordenamiento – **Inserción**

```
def Orden_Insercion(lista):
    for i in range(1,len(lista)):
        auxiliar = lista[i]
        j = i
        while j>0 and auxiliar < lista[j-1]:
            lista[j] = lista[j-1]
            j -= 1
        lista[j] = auxiliar
    return lista

L = [ 3, 1, 70, -1, 3, 7, 5]
print("L:", L)

L_desordenado = L.copy()
L_ordenado = Orden_Insercion(L)

print("L_desordenado:", L_desordenado)
print("L_ordenado:", L_ordenado)

print("L:", L)
```

```
>>>
L: [3, 1, 70, -1, 3, 7, 5]
L_desordenado: [3, 1, 70, -1, 3, 7, 5]
L_ordenado: [-1, 1, 3, 3, 5, 7, 70]
L: [-1, 1, 3, 3, 5, 7, 70]
```

Selección versus Inserción

Selección

- Buscamos el menor elemento de la lista y lo intercambiamos con el primer elemento.
- Después buscamos el segundo menor elemento y lo intercambiamos con el segundo elemento de la lista.
- … Y así hasta que llegamos al penúltimo elemento de la lista.

```
[3, 1, 70, -1, 3, 7, 5]
[-1, 1, 70, 3, 3, 7, 5]
[-1, 1, 70, 3, 3, 7, 5]
[-1, 1, 3, 70, 3, 7, 5]
[-1, 1, 3, 3, 70, 7, 5]
[-1, 1, 3, 3, 5, 7, 70]
[-1, 1, 3, 3, 5, 7, 70]
```

Inserción

- Asumimos que los primeros **i-1** elementos están ordenados.
- Se compara el elemento **i**-ésimo con los elementos ya ordenados. Se busca dónde insertarlo: mientras los elementos sean mayores, se desplazan hacia la derecha, deteniéndose cuando se encuentra un elemento menor o cuando ya no se encuentran elementos (el elemento a ordenar es el menor). Se inserta el elemento en dicha posición.

```
[3, 1, 70, -1, 3, 7, 5]
[1, 3, 70, -1, 3, 7, 5]
[1, 3, 70, -1, 3, 7, 5]
[-1, 1, 3, 70, 3, 7, 5]
[-1, 1, 3, 3, 70, 7, 5]
[-1, 1, 3, 3, 7, 70, 5]
[-1, 1, 3, 3, 5, 7, 70]
```

Selección versus Inserción

Selección

```
def Orden_Seleccion(lista):
 for i in range(0,len(lista)-1):
  posicion = i
  for j in range(posicion+1,len(lista)):
   if lista[j] < lista[posicion]:
    posicion = j
   auxiliar         = lista[posicion]
   lista[posicion] = lista[i]
   lista[i]         = auxiliar
 return lista
```

```
[3, 1, 70, -1, 3, 7, 5]
[-1, 1, 70, 3, 3, 7, 5]
[-1, 1, 70, 3, 3, 7, 5]
[-1, 1, 3, 70, 3, 7, 5]
[-1, 1, 3, 3, 70, 7, 5]
[-1, 1, 3, 3, 5, 7, 70]
[-1, 1, 3, 3, 5, 7, 70]
```

Inserción

```
def Orden_Insercion(lista):
 for i in range(1,len(lista)):
  auxiliar = lista[i]
  j = i
  while j>0 and auxiliar < lista[j-1]:
   lista[j] = lista[j-1]
   j -= 1
  lista[j] = auxiliar
 return lista
```

```
[3, 1, 70, -1, 3, 7, 5]
[1, 3, 70, -1, 3, 7, 5]
[1, 3, 70, -1, 3, 7, 5]
[-1, 1, 3, 70, 3, 7, 5]
[-1, 1, 3, 3, 70, 7, 5]
[-1, 1, 3, 3, 7, 70, 5]
[-1, 1, 3, 3, 5, 7, 70]
```

Ordenamiento – links de interés

- Animaciones de distintos algoritmos de ordenamiento
  - http://www.sorting-algorithms.com
  - https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html

- Algoritmos de ordenamiento en Python
  - http://danishmujeeb.com/blog/2014/01/basic-sorting-algorithms-implemented-in-python

Criterio de ordenamiento

- El criterio de ordenamiento está definido de manera implícita en la forma en que se comparan dos valores:

```
def Orden_Seleccion(lista):
    for i in range(0,len(lista)-1):
        posicion = i
        for j in range(posicion+1,len(lista)):
            if lista[j] < lista[posicion]:
                posicion = j
        auxiliar         = lista[posicion]
        lista[posicion] = lista[i]
        lista[i]         = auxiliar
    return lista

def Orden_Insercion(lista):
    for i in range(1,len(lista)):
        auxiliar = lista[i]
        j = i
        while j>0 and auxiliar < lista[j-1]:
            lista[j] = lista[j-1]
            j -= 1
        lista[j] = auxiliar
    return lista
```

- Si cambiamos sólo ese pedazo del código, podemos cambiar el cómo se ordena

Ejemplo – cambiar criterio de ordenamiento

- Suponga que quiere ordenar una lista cuyas componentes son los datos de personas correspondiente a su nombre y edad, que viene en una **lista** del tipo **[nombre, edad]**.
- ¿Cómo podemos ordenar la lista para que los datos queden ordenados de **menor a mayor edad**?

- Si L fuese:

```
L=[['juana',15],['jose',20],['patricio',11],['francisco',9],['andrea',17]]
```

- Queremos obtener la lista:

```
L=[['francisco',9],['patricio',11],['juana',15],['andrea',17],['jose',20]]
```

- El método **`sort()`** en este caso no arroja el resultado deseado.
- Pero, podemos modificar el criterio de ordenamiento de cualquiera de nuestros algoritmos.

Ejemplo – cambiar criterio de ordenamiento

```
def EsMenorPorNombre(item1, item2):
    return item1[0] < item2[0]

def EsMenorPorEdad(item1, item2):
    return item1[1] < item2[1]

def EsMenor(item1, item2):
    return EsMenorPorEdad(item1, item2)

def Orden_Insercion(lista):
    for i in range(1,len(lista)):
        auxiliar = lista[i]
        j = i
        while j>0 and EsMenor(auxiliar, lista[j-1]):
            lista[j] = lista[j-1]
            j -= 1
        lista[j] = auxiliar
    return lista
```

```
>>>
L_desordenado: [('juana', 15), ('jose', 20), ('patricio', 11), ('francisco', 9), ('andrea', 17)]
L_ordenado: [('francisco', 9), ('patricio', 11), ('juana', 15), ('andrea', 17), ('jose', 20)]
```

Ordenamiento usando `sorted()`

- Un método de ordenamiento similar a **`sort()`** es **`sorted()`** que a diferencia del primero devuelve una lista ordenada sin modificar la lista que se le pasa como parámetro.

- Una variante que presenta esta función es que permite especificar que se quiere ordenar una lista de listas, respecto de una componente de la lista. En nuestro ejemplo, **“edad”**, que ocupa la posición **`[1]`** de la lista.

- Para ello debemos:
  - Definir una función auxiliar que devuelva esa componente.
  - Definir en la invocación de **`sorted()`** que ordenaremos usando dicha función.

Ordenamiento usando `sorted()`

```
def Edad(item):
    return item[1]

L=[['juana',15],['jose',20],['patricio',11],['francisco',9],['andrea',17]]

L_ordenado = sorted(L,key=Edad)

print("L:", L)
print("L_ordenado:", L_ordenado)
```

```
>>>
L: [['juana', 15], ['jose', 20], ['patricio', 11], ['francisco', 9], ['andrea', 17]]
L_ordenado: [['francisco', 9], ['patricio', 11], ['juana', 15], ['andrea', 17], ['jose', 20]]
```

Ordenamiento usando `sort()`

- También se puede usar **`key`** con el método **`sort()`**. En este caso, la lista sí se modifica.

```
def Edad(item):
    return item[1]

L=[['juana',15],['jose',20],['patricio',11],['francisco',9],['andrea',17]]
print("L:", L)

L.sort(key=Edad)
print("L:", L)
```

```
>>>
L: [['juana', 15], ['jose', 20], ['patricio', 11], ['francisco', 9], ['andrea', 17]]
L: [['francisco', 9], ['patricio', 11], ['juana', 15], ['andrea', 17], ['jose', 20]]
```

¿Cómo está implementado el sort de Python?

- Se usa Timsort
  - http://en.wikipedia.org/wiki/Timsort