Universidad Central de Venezuela

Facultad de Ingeniería

Ciclo Básico

Departamento de Investigaciones Operacionales y Computación

Cátedra de Programación

Metodos de Ordenamiento

Profesor:

Carlos González

Bachiller:

Ricardo González

C.I: V- 27.469.275

Caracas, junio de 2023

¿Qué es un método de Ordenamiento?

Los métodos de ordenamiento son algoritmos que realizan la operación de arreglar los registros de una tabla en algún orden secuencial de acuerdo a un criterio de ordenamiento. El ordenamiento se efectúa con base en el valor de algún campo en un grupo de datos. El ordenamiento puede estar dado de forma iterativa o recursiva según la naturaleza y forma de ejecución del mismo.

Es decir, estos algoritmos se utilizan en la programación para ordenar datos de manera ascendente y descendente en una lista. Estos métodos de ordenamiento visto desde las líneas de un programa se basan en iteraciones de ordenación de datos hasta lograr el objetivo correspondiente, para así detener las iteraciones.
Antes de analizar el código descrito para el método de bubble sort, es necesario aclarar otros conceptos como:

a. **Import:** Es una palabra clave en python y se utiliza para "invocar" un paquete en python específico como las matemáticas, para poder utilizar las fórmulas matemáticas que usan en el mundo real de acuerdo a los comandos descritos en python en vez de establecer fórmulas manuales y evitando las confusiones en la escritura del código.

```
>>> Import math
 >>> a = math.cos(80)
>>>  print (a)
```

Para este caso se utiliza la fórmula de math.cos para hallar el coseno de 80 y luego imprimir este resultado.

b. **Random:** es un paquete dentro de python al igual que math. Es utilizado para generar números aleatorios con decimales, enteros, desordenar una lista u obtener un valor aleatorio en una secuencia.
c. **Sample:** Es una función dentro del paquete random que ofrece una muestra aleatoria en una secuencia. Tal cual como su nombre en inglés Sample es muestra en español, y podemos establecer aleatoriamente la cantidad de muestras de una secuencia.

```
>>>  import random
>>> números = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> muestra = random.sample(números, 2)
>>> print(muestra)
```

d. **Def y return:** Para el primer caso, **def** se suele utilizar para definir una función específica en las líneas de un programa. Cuando se refiere a específica, es una personalizada. **return** es otra palabra clave en Python que se utiliza para devolver un valor desde una función. Cuando se utiliza return, la ejecución de la función se detiene y el valor especificado se devuelve a la parte del programa que llamó a la función. Es decir, definimos una función que pasa por un proceso en las líneas de código y nos devuelve un resultado que puede ser utilizado después.

```
>>> def suma(a, b):
>>>  resultado = a + b
>>> return resultado
```

1. **Primer método Bubble Sort:**

El método bubble sort radica en la revisión de los elementos de una lista y partiendo de ella se intercambian los elementos de esa lista de menor a mayor o al revés dependiendo de cual sea el caso. Se debe someter a la lista a un ciclo **for** hasta que no sean requeridos más cambios.

```
from random import sample
# Importamos un Método de la biblioteca random para generar listas
aleatorias

lista = list(range(100)) # Creamos la lista base con números del 1 al
100

# Creamos una lista aleatoria con sample
#(8 elementos aleatorios de la lista base)
vectorbs = sample(lista,8)
```

Para esta primera parte del código, se pone en la mesa el paquete de funciones **random**, pero aquí con la función **from,** solo se extrae la metodo **sample** para extraer una muestra de 8 números en una lista con un rango del número de 1 al 100.

```
def bubblesort(vectorbs):
    """Esta función ordena el vector que le pases como argumento con el
Método de Bubble Sort"""

    # Imprimimos la lista obtenida al principio (Desordenada)
    print("El vector a ordenar es:",vectorbs)
    n = 0 # Establecemos un contador del largo del vector

    for _ in vectorbs:
        n += 1 #Contamos la cantidad de caracteres dentro del vector
```

Para la segunda parte del código se define la función bubblesort para la lista de 8 números en el vector de la primera parte, imprime el vector a ordenar y establece un contador que estará ligado al ciclo for.

```
for i in range(n-1):
    # Le damos un rango n para que complete el proceso.
        for j in range(0, n-i-1):
            # Revisa la matriz de 0 hasta n-i-1
            if vectorbs[j] > vectorbs[j+1] :
                vectorbs[j], vectorbs[j+1] = vectorbs[j+1], vectorbs[j]
            # Se intercambian si el elemento encontrado es mayor
            # Luego pasa al siguiente
    print ("El vector ordenado es: ",vectorbs)

bubblesort(vectorbs)
```

En la última parte del código hace que el bucle de **'i'** se repita n-1 veces, como ya **'n'** está definido anteriormente en el código. Luego se tiene anidado otro ciclo for definiendo **'j'** como los elementos dentro del vector a ordenar, donde ***j*** es el primer elemento y ***j+1*** es el elemento que le sigue. Posteriormente se establece una operación donde se ordenan gracias al operador lógico **if.**