Sort_Insertion.py:

Introducción:

Ante la consigna de diseñar un algoritmo que cumpla con la funcionalidad de realizar un ordenamiento de valores en una lista mediante un método de inserción, se realizó la programación de dicho algoritmo mediante la utilización del código Python 3.13.9. Dicho algoritmo fue guardado en el archivo Insertion_Sort.py. A continuación, se procede a realizar un informe sobre los detalles de este algoritmo, en el cual se incluye su descripción, las decisiones tomadas, la implementación aplicada y las dificultades encontradas durante el desarrollo del mismo.

Descripción del algoritmo:

En este algoritmo, se implementa el algoritmo Insertion_Sort.py para poder integrarlo en el visualizador de algoritmos. Este algoritmo tiene como objetivo realizar un ordenamiento de un conjunto de valores desordenados.

Este algoritmo de ordenamiento realiza sus funciones tomando un elemento perteneciente a la lista cada vez que se ejecuta y lo inserta en la posición correcta entre los elementos que ya fueron procesados y ordenados. En cada ejecución del algoritmo, se compara el elemento del índice actual con los elementos de los índices anteriores e inserta el valor del elemento de la lista tomado en el índice de lista que le corresponde, mediante un swap de valores.

Decisiones de implementación:

Durante el desarrollo del algoritmo, se decidió utilizar la variable i para indicar el valor del elemento actual que será insertado y ordenado. Se decidió utilizar la variable j como cursor de desplazamiento hacia los primeros índices.

A raíz de que el visualizador necesita realizar solo una ejecución de la función step() para realizar una ejecución del algoritmo, se decidieron realizar cambios en el código del algoritmo. Estos cambios son los siguientes: Setear j=i para la primera ejecución en la que se realice un desplazamiento por el algoritmo; Que cada ejecución realice un único swap entre valores; Realizar un incremento en el valor de i al finalizar un desplazamiento de forma efectiva. Con el fin de mantener una coherencia entre las ejecuciones de la función step(), se utilizan las variables ítems, i, j y n para cumplir con las exigencias y compatibilidades con el visualizador.

I**mplementación aplicada**:

```
items = []
n = 0        # Cantidad de elementos que tendrá la lista y que serán ordenados
i = 0     # índice del elemento que queremos insertar
j = None   # cursor de desplazamiento hacia la izquierda (None = empezar)

def init(vals):
```

```
    global items, n, i, j
    items = list(vals)
    n = len(items)
    i = 1      # insertion sort empieza en el segundo elemento
    j = None

def step():
    global items, n, i, j

    #Todo
    # - Si i >= n: devolver {"done": True}.
    if i >=n:
        return {"a": None, "b": None, "swap": False, "done": True}
    # - Si j es None: empezar desplazamiento para el items[i] (p.ej., j = i) y devolver un
highlight sin swap.
    if j is None:
        j=i
        if j >0:
            return {"a": j-1, "b": j, "swap": False, "done": False}
        else:
            return {"a": 0, "b": 0, "swap": False, "done": False}
    # - Mientras j > 0 y items[j-1] > items[j]: hacer UN swap adyacente (j-1, j) y devolverlo con
swap=True.
    if j>0 and items[j-1] > items[j]:
        items[j-1], items[j] = items[j], items[j-1]
        j-=1
        return {"a": j, "b": j+1, "swap": True, "done": False}
    # - Si ya no hay que desplazar: avanzar i y setear j=None.
    i+=1
    j=None
    return {"a": None, "b": None, "swap": False, "done": False}
```

En el código se compone de la siguiente manera:

- Def init(vals): Función que devuelve las variables globales de seteadas a sus valores iniciales. Estas variables son:
    - n: Cantidad de elementos almacenados en la lista a ordenar.
    - j: Cursos de desplazamiento de hacia la izquierda, el cual permite determinar la posición en la cual va a ser ubicado el elemento a ordenar.
    - i: Valor del elemento de la lista que será ordenado.
    - items: Lista que se utiliza para almacenar las variables anteriormente mencionadas y utilizarlas en otras funciones del algoritmo.

- Def step(): Función de ejecución del algoritmo. En esta función se aplican las condiciones lógicas mediante if y else. Sesta función se ejecuta una vez por cada run del algoritmo. Devuelve un diccionario con las claves “a”, “b”, “swap” y “done”, cuyos valores serán evaluados en el visualizador y varían dependiendo de las condiciones lógicas implementadas.
    - a: Clave de lista que almacena valores int. Se usa para almacenar el índice del cursor de desplazamiento que mueve el elemento hacia la izquierda.
    - B: Clave de lista que almacena valores int. Se usa para almacenar el valor del elemento que será ordenado.
    - Swap: Clave de lista que almacena valores tipo booleano. Se usa para determinar si el valor de las claves a y b fueron intercambiados entre sí. Si toma el valor True, significa que los valores fueron cambiados. Si toma el valor False, significa que los valores no fueron cambiados.
    - Done: Clave de lista que almacena valores tipo booleano. Se usa para determinar si la ejecución el algoritmo debe continuar o finalizar. Si el valor es True, el algoritmo ha finalizado. Si el valor es False, el algoritmo ha terminado.

Dificultades encontradas:

Durante el desarrollo del algoritmo Insertion_Sort, se encontraron distintas dificultades antes de comprobar su pleno funcionamiento. Estas dificultades fueron las siguientes:

- Dificultad para lograr comprender las consignas y requerimientos del contrato establecido.
- Dificultades para lograr las correctas funciones lógicas en la función step() (if, else).
- Dificultades para lograr que las variables globales puedan tomar los valores requeridos a medidas que realizan las ejecuciones.
- Dificultades para que las acciones return de la función step() devuelvan los diccionarios con las funciones correctas.