Informe de Trabajo Practico de Introduccion a la Programación
Comisión 02

Correo: gitip2025@gmail.com
Integrantes:

- Delfina Gomez
  DNI: 44.885.456
- Aylen Medina
  DNI: 46.752.206
- Paulina Baez
  DNI: 47.054.037

Fecha de entrega: 27/11/25

20/11/25, 16:54hs (“Delfina Gomez”): Ante las dificultades técnicas con el push de git con Github individual de cada integrantes y que los informes no se vean, se ha creado este correo compartido y este informe en formato Word en caso de no ser posible enviar los informes con Git.

21/11/25 22:20hs(“Delfina Gomez”): "En el presente informe, yo, Delfina Gomez, comento mis encuentros y dificultades en el código. El algoritmo de inserción presenta un comportamiento mucho más lento que otros códigos como el bubble o selection, y es que al ser un código que compara elemento con el de su izquierda, una vez comparado lo coloca donde debería estar (hablando de que funcione de forma creciente). Las dificultades que se ha presentado en el momento de este código fue la forma en la colocación de las banderas True/False, ya que suele ser confuso como y en qué momento entonces se da inicio o terminado dado cuando obtiene determinado valor. Una vez resuelto eso, nos encontramos con la situación en la anteúltima clase presencial de la materia en la fecha 20/11/25 que, por algún motivo, el código no corría como debería, ya que ordenaba el primer o segundo elemento y debías volver a reiniciar para que repita el proceso con la nueva lista que dejó. Sin embargo,sin hacer cambios al código, de repente se lo probó y funcionó completamente, haciendo todos los cambios con un solo click y mostrando su comportamiento".

23/11/25 23:07hs(“Delfina Gomez”): "Informo el primer paso del código de inserción sobre porque inicia con if i>=n. Entendemos a "n" como la variable que almacena la cantidad de elementos de una lista (por n=len(list) en la función def init(vals)), sirve para saber hasta que indice se puede llegar y cuando frenar los desplazamientos. La variable "i"(i=1 en la función def init(vals)) es el índice del elemento que quiero "insertar" correctamente. Funciona de la siguiente manera: El índice 0 ya está "ordenado" por sí mismo, por lo que empiezo a ordenar desde i=1, y cada vez que quiero insertar un elemento, hago i+=1. Entendiendo eso, if i>=n: return {"done":True} condiciona que todos los elementos se hayan insertado, y de hacerlo, hace que el visualizador se detenga.

24/11/25 - 17:01 hs (Aylen Medina)
"En el presente informe, yo, Aylen Medina, comento mis encuentros, dificultades y decisiones de implementación respecto al algoritmo de Bubble Sort (o burbujeo). Este

algoritmo, a diferencia de otros como inserción o selección, requiere comparar pares de elementos adyacentes en cada paso, intercambiando sus posiciones si están desordenados. Al estar adaptado al visualizador, debe ejecutarse paso a paso, lo que hace que su comportamiento sea distinto al de una implementación normal en Python.

Una de las principales dificultades que tuve al principio fue que el código no mostraba ningún movimiento en el visualizador. Esto ocurría porque el índice j —que es el que recorre los pares para compararlos— avanzaba únicamente cuando no había swap. Esto hacía que, si sucedía un intercambio, el código quedará trabado comparando siempre el mismo par, y por eso la animación no se veía. La solución fue hacer que j avance siempre, haya o no intercambio.

También tuve problemas con la bandera swapped, que indica si en la pasada actual hubo algún intercambio. Esta variable es importante porque si en una pasada completa no hubo swaps, significa que toda la lista está ordenada y el algoritmo puede terminar antes. En versiones anteriores no se reiniciaba bien, lo cual hacía que el algoritmo terminará antes de tiempo o siguiera cuando ya no había nada por ordenar.

Otra complicación fue la variable hice_swap, que es la que la interfaz necesita para saber si debe animar el intercambio. Al principio no estaba inicializada antes del if, generando errores internos que no aparecían en pantalla pero que impedían el funcionamiento. Esto se resolvió definiendo siempre al inicio del paso.

Con respecto al control del flujo, al principio había usado cadenas de texto ("comparar" / "nueva_pasada"), pero eso era muy fácil de romper. Cambiar a un estado booleano (True/False) simplificó mucho el comportamiento del algoritmo y redujo errores.

Finalmente, otro punto que me costó al principio fue comprender el formato del diccionario que debe devolver la función step(). Es obligatorio devolver las claves "a", "b", "swap" y "done" porque es el formato exacto que la interfaz del visualizador (UI) necesita para iluminar las columnas, mostrar intercambios y saber cuándo detener el algoritmo.

El día 20/11/25, en la clase presencial, probé nuevamente el código y ese mismo día logré corregirlo por completo. Ese fue el momento en el que verifiqué que el avance de j, el reinicio de swapped, el estado del algoritmo y la estructura del diccionario estaban funcionando correctamente. A partir de esas correcciones, el Bubble Sort empezó a animarse bien en el visualizador y el algoritmo finalmente quedó funcionando como corresponde."

24/11/25 - 16:53 hs (Aylen Medina) En este paso explicó el funcionamiento principal del código de burbuja que hice. El algoritmo recorre la lista comparando siempre dos elementos vecinos (a y b), y si el de la izquierda es mayor, los intercambia. Uso i para saber cuántas pasadas se hicieron y j para moverme dentro de cada pasada. La función init(vals) solo prepara todo para empezar desde cero (lista, índices y banderas). En step() se compara, se hace el swap si corresponde, y cuando j llega al final de la pasada se revisa si hubo intercambios: si no hubo, la lista ya está ordenada; si sí hubo, comienza otra pasada

moviendo i y reiniciando j. Cada vuelta devuelve un diccionario con los índices y si hubo o no intercambio para que el visualizador muestre el avance paso a paso.

26/11/25 - 13:15 hs (Aylen Medina) Yo, Aylen Medina, informó las dificultades que tuve al utilizar Git durante el trabajo práctico. Como no tengo experiencia previa con esta herramienta, muchas de las indicaciones del propio programa me resultaron confusas y no entendía por qué mis cambios no aparecían o por qué no podía modificar ciertos archivos.

Una de las mayores complicaciones fue que, por error, estuve trabajando en un repositorio que no era el mío, lo que me impedía hacer cambios o subir mis archivos. Esto me generó bastante frustración, ya que no entendía por qué el repositorio no mostraba mis modificaciones.

Finalmente, con la ayuda de mi compañera Delfina, pude entender mejor qué había pasado y cómo debía usar GitHub correctamente. Gracias a su explicación, ingresé a mi propio repositorio y pude subir mis archivos editando directamente desde la plataforma. Esto me permitió ver mis cambios de manera clara y segura.

Aunque el proceso fue estresante, logré comprender cómo identificar el repositorio correcto, cómo subir mis aportes y cómo organizarme mejor dentro de GitHub.

25/11/2025 - 20:30 hs (Paulina Baez): En el presente informe, yo Paulina Báez observo mis dificultades y problemas con el código y git. El código de selection (o selección), tiene una restricción con la cantidad de elementos que tiene que seleccionar y ordena de acuerdo a su exploración, y puede llegar a ser más lento que sus contrapartes (Bubble o Insertion). No tuve muchas dificultades a entorno del algoritmo de selección, sin embargo, como poder subir el archivo en el Git, y cómo realizar los respectivos comentarios que informa mi proceso de observación y dificultad, me fue un poco más complicado. Ya que no entendía su manejo, por mi falta de experiencia previa a esta plataforma.

25/11/2025 - 20:35 hs (Paulina Baez): Por otra parte, explicaré cómo está constituido el código de Selección. El algoritmo busca los elementos menores de la lista, al seleccionar el elemento mínimo de la lista, lo intercambia, ordenándose para volver a buscar otro mínimo y rebobinar de nuevo el proceso hasta ordenarlo de menor a mayor.
El algoritmo se conforma por una lista nueva y variables principales . Una función init(vals) que guarda una copia de las variables principales, y una función step (): conformada por elementos de la variable y con las condiciones de buscar y swap.
En la primera fase, se busca encontrar el elemento más pequeño en la variable que compone lo no ordenado (i=0), comparando el índice menor actual con el buscador, donde devuelve el elemento menor encontrado, avanzado con otro elemento hasta finalizar su búsqueda.
En la segunda fase, se intercambia si es necesario (swap_hecho = (min_idx != i)), al encontrar el elemento menor de la lista.
Y la última fase, al terminar el ordenamiento de los elementos, regresa en posición de búsqueda hasta repetir el mismo ciclo.

26/11/25 23:30hs (Delfina Gomez): Explico parte del código en el caso de que el elemento i no se haya movido: if j is none: // j=i // return {"a":j,"b":j,"swap":False, "done":False}// J no ha

arrancado el desplazamiento, por lo que lo inicio en i (j=i). Esto solo sirve para mostrar que seleccioné el elemento. El algoritmo "marca" el número que va a insertar.

26/11/25 23:40hs (Delfina Gomez): En otra parte del código de algoritmo de ordenamiento por inserción, se muestra las condiciones para desplazar a la izquierda con la siguiente linea: if j > 0 and items[j - 1] > items[j]: La condición explícita es que mientras no esté al inicio de la lista (j>0) y el elemento de la izquierda sea más grande (items[j-1] que el de la derecha (items[j]), tengo que intercambiarlos. Esto mueve el número hacia su posición correcta.

27/11/2025 10:51 hs (Paulina Baez): El algoritmo de selección, tiene aparte una restricción busca paso por paso dos elementos de la lista que quiere hacer el intercambio.Por lo tanto, al simular con grandes cantidades de elementos de la lista puede que tarde en hacer el intercambio o falle la búsqueda.
También indicó que el algoritmo de selección tiene un retorno como bucle, ya que al poner (return {done: False}) esto implicaría que la búsqueda nunca termina de finalizar, pero si estaría completamente ordenado hasta que se sume un nuevo elemento de la lista.