Registro de actividades

Ejercicio 1: Suma con gráfico

-Lectura del ejercicio: 5 minutos viendo los ejemplos
-Interpretación del ejercicio: 10 minutos
-Pensamiento e indagación de las librerías a utilizar: en este punto estuve buen tiempo pensando y luego investigando cuáles podrían ser las mejores librerías para hacer una animación en tiempo real. Recordé que en un curso de Métodos Computacionales utilicé diferentes clases de la librería Matplotlib para graficar funciones en 2d y 3d, sin embargo descubrí que existía la clase FuncAnimation, y dado que conocía bien la librería Matplotlib decidí abordar el problema con esta función. Ahora bien, al momento de ver cómo iba a registrar cada evento cuando el usuario oprimiera una tecla, pensé que podría implementar de alguna manera la clase *Input* y guardar los registros en una variable. Luego de indagar un poco más sobre librerías descubrí la función Keyboard, la cual permitía registrar de manera sencilla cada evento que el usuario hiciese con las teclas. Vi ejemplos de cómo y en qué contexto se podía implementar la librería, y fue allí cuando me di cuenta que la librería *Keyboard* funcionaba como un led o un buzzer en el sentido que el pulsar una tecla hacía las veces de “escuchar” una acción.
-Desarrollo de las versiones prueba: Dado que actualmente trabajo con sistema operativo Windows, podía trabajar desde Google Colab o podía instalar Python (para trabajar desde Jupyter Notebook). Desde Google Colab comencé a trabajar, instalé librerías y las importé al .ipynb, pero pronto me di cuenta que la animación en tiempo real no se estaba ejecutando correctamente, y que adicionalmente algunos atributos de la librería Keyboard no se estaban ejecutando tampoco, por lo que decidí empezar a trabajar desde Jupyter Notebook con archivos .py para luego ejecutarlos desde el cmd del equipo. Pensé inicialmente en hacer el programa con una función (def animate():) que hiciera la animación y otra función que registrara las pulsaciones y llamara a la función animate para que se fuese actualizando en tiempo real. Hice un par de pruebas, pero no sentía que fuese muy organizado ni elegante que el programa se basara en funciones, por lo que luego pensé el programa como una clase única con varias funciones y con métodos y atributos propios de la clase. En esta tarea pude haberme demorado unas 6 horas mientras aplicaba cambios, los ejecutaba, corregía errores, etc.
-Desarrollo de la versión final: Cuando ya estaba terminando el primer ejercicio, empecé a mirar el segundo y pude ver que había un librería llamada Pyqtgraph con la cual estaba hecha la interfaz de *Abacus*, pensé tal vez en volver a hacer el ejercicio con esta librería pues vi que se podían incorporar botones y otros atributos gráficos, pero consideré que no me alcanzaría el tiempo para desarrollar de nuevo todo el ejercicio. Aquí fue cuando pensé que una alternativa para los botones podía ser que el usuario oprimiese la tecla “r” para reiniciar el contador y la tecla “e” para salir del programa. Pude observar además que la gráfica no estaba tomando en cuenta los últimos 30 segundos de actividad, sino todo el tiempo de ejecución, por lo que en la función *actualizar gráfica* incorporé un condicional con el tiempo. Si este superaba los 30 segundos, las variables *tiempos* y *valores* solo tomarían los últimos 30 valores registrados hasta el momento. Luego de varias pruebas logré que la tecla “r” reiniciara el contador de la suma acumulada, pero no logré que el programa se cerrara al oprimir la tecla “e”. No tengo idea de que pueda ser, pues en la consola no arroja *warnings* o errores de ejecución, puede ser un tema de mi computador.

Ejercicio 2: Ejecución y edición AbacusSoftware

-Lectura del ejercicio: 5 minutos viendo los ejemplos
-Exploración de las carpetas: luego de descargar la carpeta di doble click sobre el archivo test.py, rápidamente pude observar que se abrió el cmd pero no ejecutó nada. Procedí a abrir las carpetas desde Jupyter Notebook y pude ver que, dentro de test.py, las siguientes líneas eran comentarios:
pa.constants.DEBUG = True
abacus.open_stdout()
abacus.close_stdout()
Quité el signo # de las líneas, guardé y ejecuté de nuevo el programa sin éxito alguno.
Consideré que lo más sensato era leer el README.md para ver si habían instrucciones allí. Abrí el requirements.txt e hice los cambios que indicaba el taller. Luego de abrir el .md instalé el AbacusSoftware y luego lo ejecuté desde la terminal de Jupyter, vi que ahora se inicializaba el ejecutable, pero solo aparecía la imagen de Abacus y posterior a ello salían los prompts con los errores que detectaba la terminal. En particular, este llamó mi atención:
AttributeError: module 'pyqtgraph' has no attribute 'GraphicsWindow'. Did you mean: 'GraphicsView'?
Pude ver que GraphicsWindow era un atributo depurado, obsoleto, por lo que hice el reemplazo en todas las líneas del main.py donde aparecía la palabra GraphicsWindow, guardé y ejecuté. No ejecutó el programa. Seguí leyendo el .md y seguí las instrucciones de crear un ambiente virtual para los desarrolladores, luego activé los scripts y cuando estaba instalando los paquetes de requirements.txt me apareció el siguiente aviso:

```
File
"C:\Users\chris\AppData\Local\Temp\pip-build-env-k1zwcks1\overlay\Lib\site-packages\setuptools\__init__.py", line 10, in <module>
    import distutils.core
ModuleNotFoundError: No module named 'distutils'
```

Investigué y el módulo distutils también estaba depurado de las versiones más recientes de Python. Intenté incoporarlo manualmente en el requirements.txt pero no hubo cambios. Tras múltiples intentos de ejecutar tanto el AbacusSoftware como el test.py no tuve éxito, no pude encontrar qué módulos, librerías o clases debía incorporar o modificar.
Esta tarea me tomó 4 horas aproximadamente.

Botones: Pude hallar en el main.py los botones Connect, Start Acquisition, así como el botón Help. Incorporé las siguientes líneas para su incorporación, visualización y ejecución:

```
self.menuHelp = self.menubar.addMenu("Help")
self.menubar.addAction(self.menuHelp.menuAction())
self.menuHelp.addAction(self.actionAbout)
self.help_window = AboutWindow()
```

Esta última línea llama a la clase “AboutWindow” del menuBar.py, desde la cual se setea el mensaje que aparecerá al dar click en el botón.

message = ‘As a physics student, my curriculum allows me to take courses that involve the field of electronics. In 2022, I had the opportunity to take one such course where I learned to program using "Arduino" boards. I was able to incorporate this knowledge into a project at the end of the course. The project involved developing a seismograph with an alert system, incorporating an accelerometer, ultrasonic sensor, OLED displays, LEDs, buzzers, resistors, etc. It was a project in which I developed programming skills in Arduino, as well as skills in integrating electronic devices into educational projects.’