ITBA

Trabajo Práctico Final Certificación Profesional Python

Alumno: Gerardo Damian Gimeno
DNI: 24.296.945
Email: gerardogimeno0772@gmail.com
Tel: +54 911 3797 7791

CONTENTS

1 Consigna del Trabajo Práctico.

Se pide implementar un programa que permita leer datos de una API de finanzas, guardarlos en una base de datos y graficarlos. Para ello, se deberá contar con las siguiente:

Entregables:

- Informe de funcionalidad y diseño.
- Repositorio Github.

Funcionalidades:

- **Menú de actualización de datos:** El programa debe solicitar al usuario el valor de un ticker, una fecha de inicio y una fecha de fin. Debe luego pedir los valores a la API y guardar estos datos en una base de datos SQL.

- **Visualización de datos:** El programa debe permitir dos visualizaciones de datos:

    a. Resumen de Tickers cargados: Debe imprimir un resumen de los datos guardados en la base de datos.

    b. Gráfico de Ticker: El programa debe permitir graficar los datos guardados para un ticker específico.

1.1 Repositorio GitHub.

1.2 Informe de funcionalidad y diseño.

El trabajo practico referente a la obtención de información de Tickers desde una API del site Polygon, fue ideado basándonos en los conocimientos aprendidos en el curso de Python del ITBA, como así también en material disponible en las pagina propia de Polygon, como así también en stackoverflow y Python.org.

Teniendo en consideración lo dicho anteriormente, para la creación del script se trabajó en base a definición de funciones que pudiesen ser mucho más fácil de leer por los profesores y a su vez, permitir aislar y realizar troubleshooting del script con mayor facilidad.

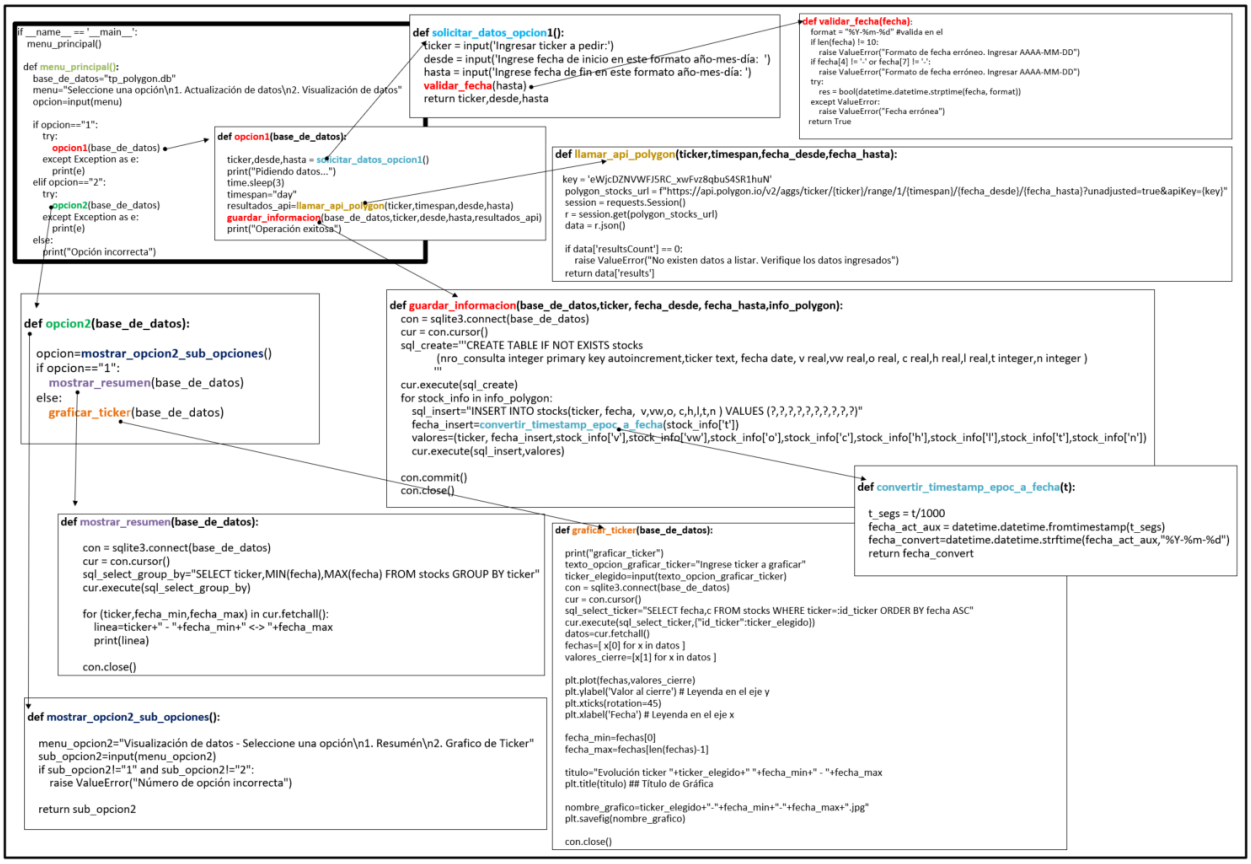
Para poder ordenar lógicamente la secuencia en que se lee cada función, se propuso realizar el siguiente diagrama lógico, con la finalidad de ofrecer una mayor claridad de las diferentes etapas del programa.

El esquema propuesto es el siguiente. El mismo puede ser exportado en Microsoft Visio para una mejor lectura:

Diagrama funcional.vsdx

1.2.1 Esquema lógico de encadenamiento de las funciones.

El diagrama que se expone tiene como función ejemplificar como se concatenan las funciones creadas para cada etapa del programa. Por ejemplo, en el cuadro del menú principal, se muestra como la función “menú_principal” llama a otras dos subfunciones mediante un if y un try.



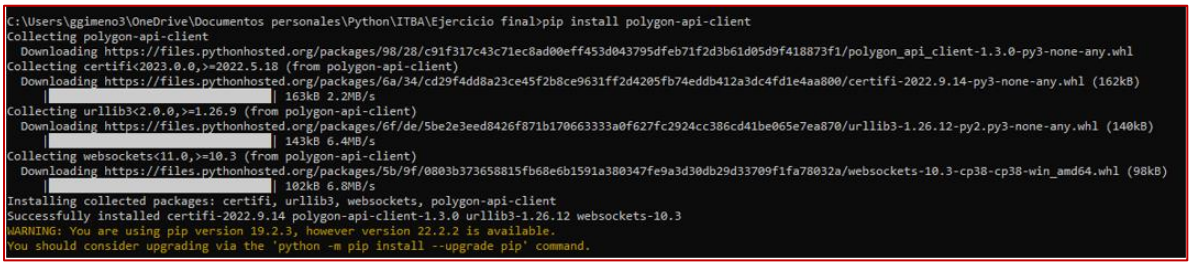
1.2.2 Importación de Librerías y funcionalidades.

A continuación, se detallan las librerías descargadas y como quedaron instaladas en el directorio raíz de mi maquina windows.

```
1  import requests
2  import json
3  import time
4  import datetime
5  import sqlite3
6  import matplotlib.pyplot as plt
```

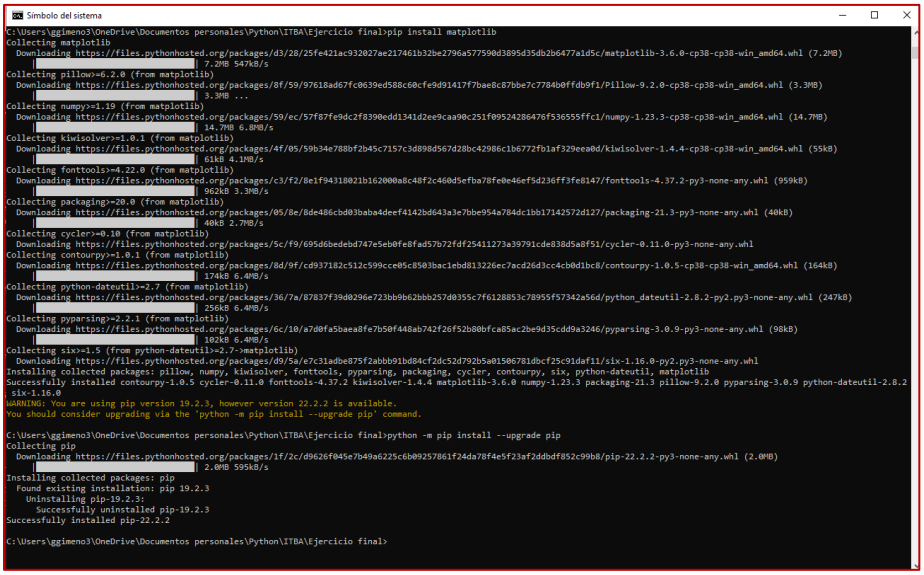
- **from polygon import RESTClient**:

  Antes de poder utilizar esta funcionalidad, realizamos la instalación de la API client de Polygon.



- **import matplotlib.pyplot as plt:**

  Importación de librería grafica conforme se muestra a continuación y actualización de PIP para el OS.



- **import datetime**

Se encontró muy conveniente el uso de este script, para poder convertir el formato de milisegundos del parámetro “v’ (Respuesta del get en formato json) de la fecha, a una notación Year-month-day. El % es para indicar a Python que reemplace en el string, los valos Y (year), m (month) y d(day), que se guardaron en memoria luego de la última solicitud.

La razón por la cual se configuro el parámetro `t_segs = t/1000` es que la librería solo acepta el parámetro en segundo y no en milisegundos. Por ello al valor t_segs se lo divide por 1000, permitiendo a la función convertir los segundos al formato indicado. Como dato adicional, este script nos permitirá incluir en la lista a agregar en sqllite3 y en formato year-month-day, los valores dados por el parámetro “v” que arroja polygon pudiendo indicar en forma correcta el día en que se tomaron los valores del ticker solicitado. Datetime, tiene un rango de fechas que va desde 1970-01-01 hasta la fecha.

```
40  def convertir_timestamp_epoc_a_fecha(t):
41
42      t_segs = t/1000
43      fecha_act_aux = datetime.datetime.fromtimestamp(t_segs)
44      fecha_convert=datetime.datetime.strftime(fecha_act_aux,"%Y-%m-%d")
45      return fecha_convert
```

- **import time**

  utilizamos este módulo para poder agregar un delay en segundos, entre consultas. time.sleep(3)

```
def opcion1(base_de_datos):

    ticker,desde,hasta = solicitar_datos_opcion1()
    print("Pidiendo datos...")
    time.sleep(3)
```

- **import sqlite3**

  Importamos sqlite3 para poder utilizar como base de almacenamiento de datos y extracción de info de ticker a graficar.

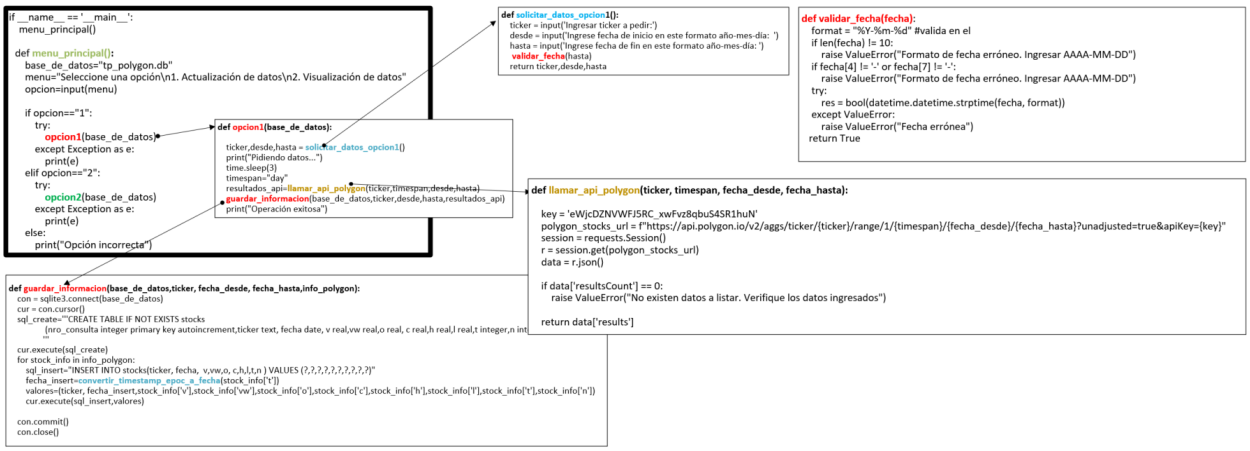
- **import json**

  Usamos el módulo import json para convertir un diccionario Python en un string de formato json. También nos permite leer un archivo json en forma directa.

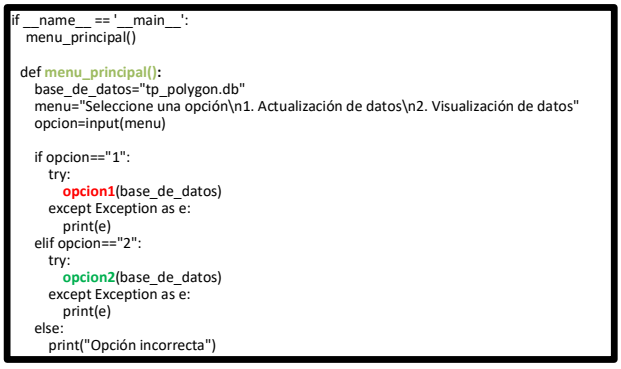
1.3 Función Principal del programa – Solicitud de datos y guardado en sqlite3.

Durante la fase de armado del script, se presentaron muchos problemas a la hora de poder seguir la estructura del programa y encontrar los errores que se estaban presentando. Por esta razon, se decidio realizar el programa en base a funciones que fuesen relacionandose una con otra.

La funcion del menu_principal(), permite identificar la funcion de mismo nombre e iniciar las opciones definidas. Cabe mencionar, que en el programa completo, primero importamos las librerias y módulos, luego definimos las funciones y como ultimo se corre el comando menu_principal().



1.3.1 Función “menu_principal()”.



- Se define el nombre “tp_polygon.db” para los datos que se guardaran en sqllite3. Tambien se definen las opciones para la obtencion de datos mendiante menues.

- Dentro de la misma, creamos una condicion para las dos opciones, 1 para Actualizacion de Datos y 2 para Visualizacion de los mismos.

- La opcion 1 dentro del if, contempla el uso de try and except. Con ellos hacemos es un manejo controlado de los errores al ingresar los datos. Dentro del except, “Exception” me permite detectar los errores que se pueden generar durante la ejecución del programa.

- Lo mismo sucede con la **opcion2**, el try y el except nos pertime el manejo de control de errores y cuando la excepción se cumple, se imprime el error arrojado por la aplicación. “Else”nos dice que si, escribimos algo distinto a 1 o 2, se imprima la leyenda “opción incorrecta”

1.3.2 Funcion “validar_fecha(fecha)”

```python
def validar_fecha(fecha):
    if len(fecha) != 10:
        raise ValueError("Formato de fecha erróneo. Ingresar AAAA-MM-DD")

    if fecha[4] != '-' or fecha[7] != '-':
        raise ValueError("Formato de fecha erróneo. Ingresar AAAA-MM-DD")

    try:
        format = "%Y-%m-%d"
        res = bool(datetime.datetime.strptime(fecha, format))
    except ValueError:
        raise ValueError("Fecha errónea")
    return True
```

Breve resumen de funcionamiento:

def validar_fecha(fecha):
 if len(fecha) != 10: *#la fecha no puede tener ni mas ni menos de 10 caracteres, por lo tanto cualquier valor mayor o menor dispara la excepcion, indicando que es erroneo el formato.*
  raise ValueError("Formato de fecha erróneo. Ingresar AAAA-MM-DD") *#lanza exepcion y para al programa*

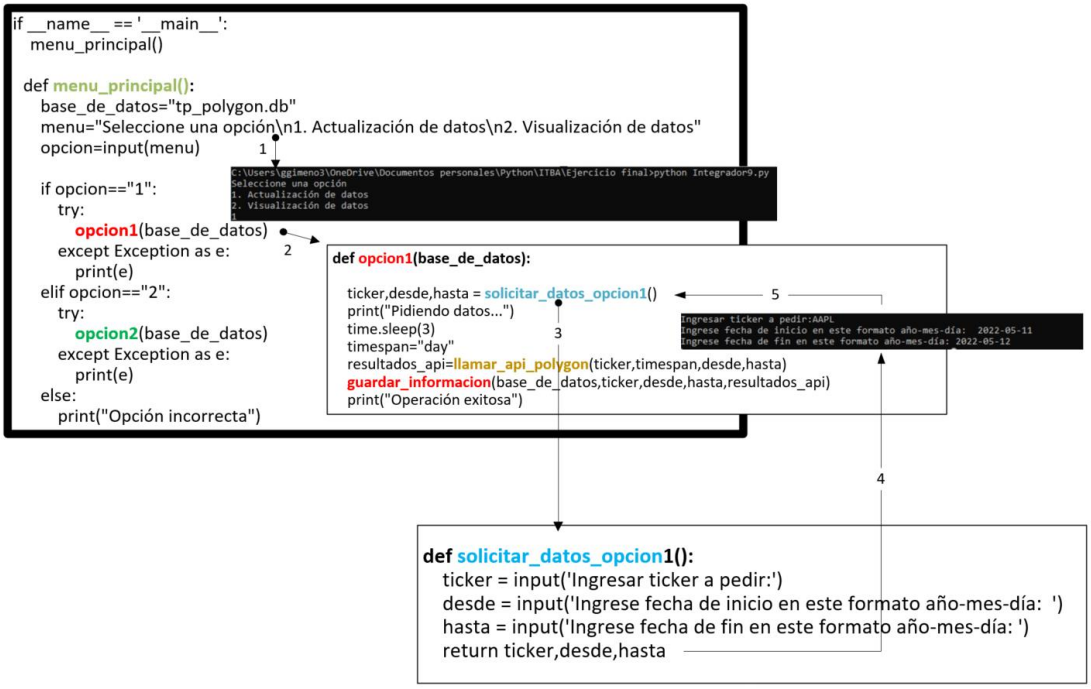
 if fecha[4] != '-' or fecha[7] != '-': *#para que se respete los guiones, no encontre otra opcion mas sencilla. Con el if verifico si en las posiciones 4 y 7 de la fecha ingresada hay un guion medio.*
  raise ValueError("Formato de fecha erróneo. Ingresar AAAA-MM-DD") *#Si no hay guion, se anza exepcion con el comando raise.*

 try:
  format = “%y-%m-%d” *#valida que el formato de la fecha, sea el mismo que necesitamos ingresar para que la API nos devuelva los valores de los tickers. ES UN STRING*
  res = bool(datetime.datetime.strptime(fecha, format)) *# a partir de un string (Format) datetime me devuelve un objeto en formato año, mes y dia.*
 except ValueError:
  raise ValueError("Fecha errónea")
#Si el formato es incorrecto, entonces lo indicara como fecha erronea.
 return True

1.3.3 Funcion “opcion1()” y “solicitar_datos_opcion1().

La funcion “opcion1” es invocada por la funcion “menu_principal”. Dentro de “opcion1”, se ejecutan sub-funciones para solicitar datos como nombre de Ticker, fecha inicio y fin de la cotizacion de la misma. Luego la funcion “opcion1”, continua ejecutadose imprimiendo un mensaje que informa al usuario que se estan pidiendo los datos. Time.sleep(3) me permite agregar un delay de 3 segundos, para emular un tiempo de consulta.

Respecto a timespan=day, lo fije en dias, ya que no pude encontrar la forma de que ese parametro sea variable tal como aparece en la pagina. Cualquier consulta sera realizada en base a dias.



1.3.4 Consulta a la API de Polygon y guardado de informacion en SQLITE3.

La funcion “llamar_api_polygon”, se ejecuta dentro de la funcion “opcion1”. Con los valores provienietes de los datos solicitados anteriormente por la f “solicitar_datos_opcion1”, se vuelcan dentro del get que se envia a la URL que queda integrada con los valores almacenados en memoria de la funcion nombrada anteriormente.

Siguiendo la logica del flujo de datos, el diagrama siguiente, muestra la contiuacion de como las funciones realizan las operaciones necesarias para compilar info, guardar en memoria y dejarla disponible para la siguiente funcion a ser llamada.

A continuacion, se detalla comando por comando cuales son las acciones que cada uno de ellos impone en el programa:

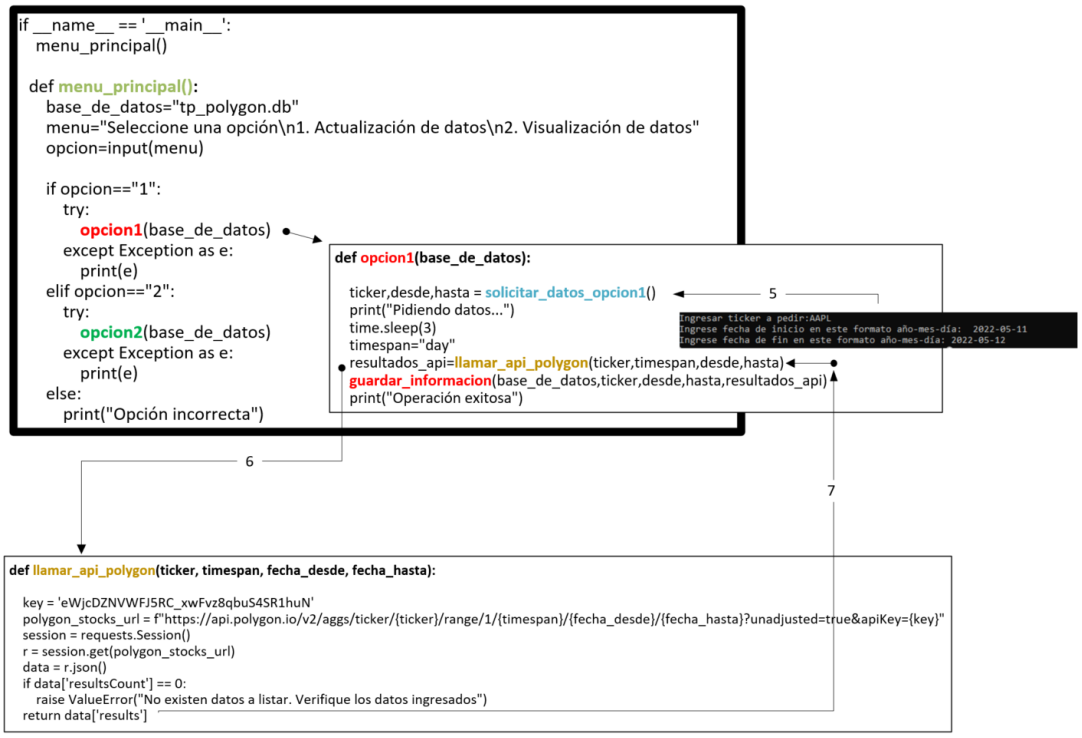
- **KEY**: Previamente se solicita una key para ser usada durante el requets de la informacion de los tickers. Esta clave es unica.

- **polygon_stocks_url:** esta variable nos arroja la URL que hara el request con nombre de ticker, fecha-inicio, fecha-fin. La construccion de la misma responde a la siguiente funcion:
  ```
  polygon_stocks_url = f"https://api.polygon.io/v2/aggs/ticker/{ticker}/range/1/{timespan}/{fecha_desde}/{fecha_hasta}?unadjusted=true&apiKey={key}"
  ```
  Los valores que estan entre corchetes, se autocomplementaran, con los valores guardado como variables en memoria, por la funcion “solicitar_datos_opcion1”

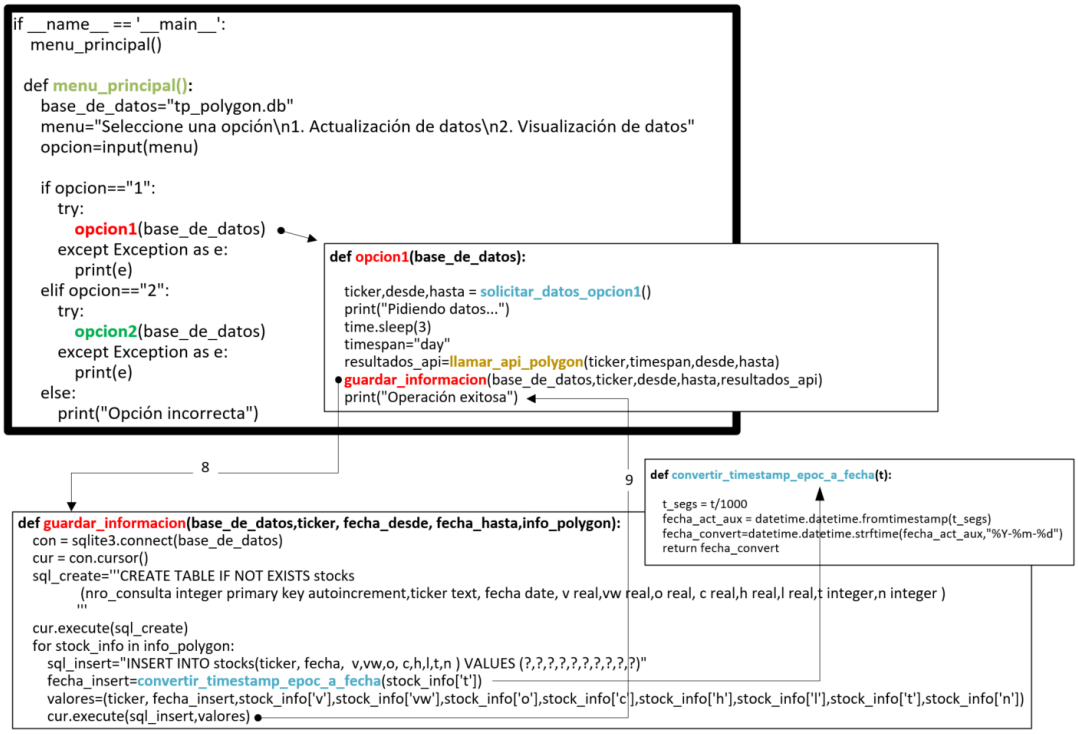
- **session = requests.Session()**
  r = session.get(polygon_stocks_url)
  data = r.json()

  Estas tres sentencias me permiten generar un get a la URL definida en el item anterior guardar los datos en una variable R con formato json.

- Respecto al if dentro de la funcion “llamar_api_polygon”, si el diccionario DATA en la clave ResultCount es igual a cero, con **raise** lanza una exepcion y devuelve el valor indicado. En este punto se frena la ejecucion y vuelve a ejecutar la funcion.



1.3.5 Funcion, guardar informacion enviada por la API.



Cuando el get que devuelve la informacion de la API de Polygon es correcta, la logica secuencial de la funcion "opcion1" continua y llama a una nueva función que permite almacenar los datos en una .db con los valores organizados de tal manera, que el primer parámetro que uno vea en la tabla sql, sea el nombre del ticker, la fecha de cierre y todos los valores asociados a la key Result, la cual llamaremos stock_info.

Cuando la tabla se haya creado con los valores definidos y con utilizando el script de timestamp para convertir el valor t de tiempo de milisegundos a segundos, para luego ser leído y traducido a una fecha en formato Y-M-D. Esta función, también permite saber la fecha exacta que proporciona el Json, pudiendo saber hasta aquellos días que son feriados o fines de semana.

A continuación, se agrega un detalle de cada sentencia o grupo de sentencias del script:

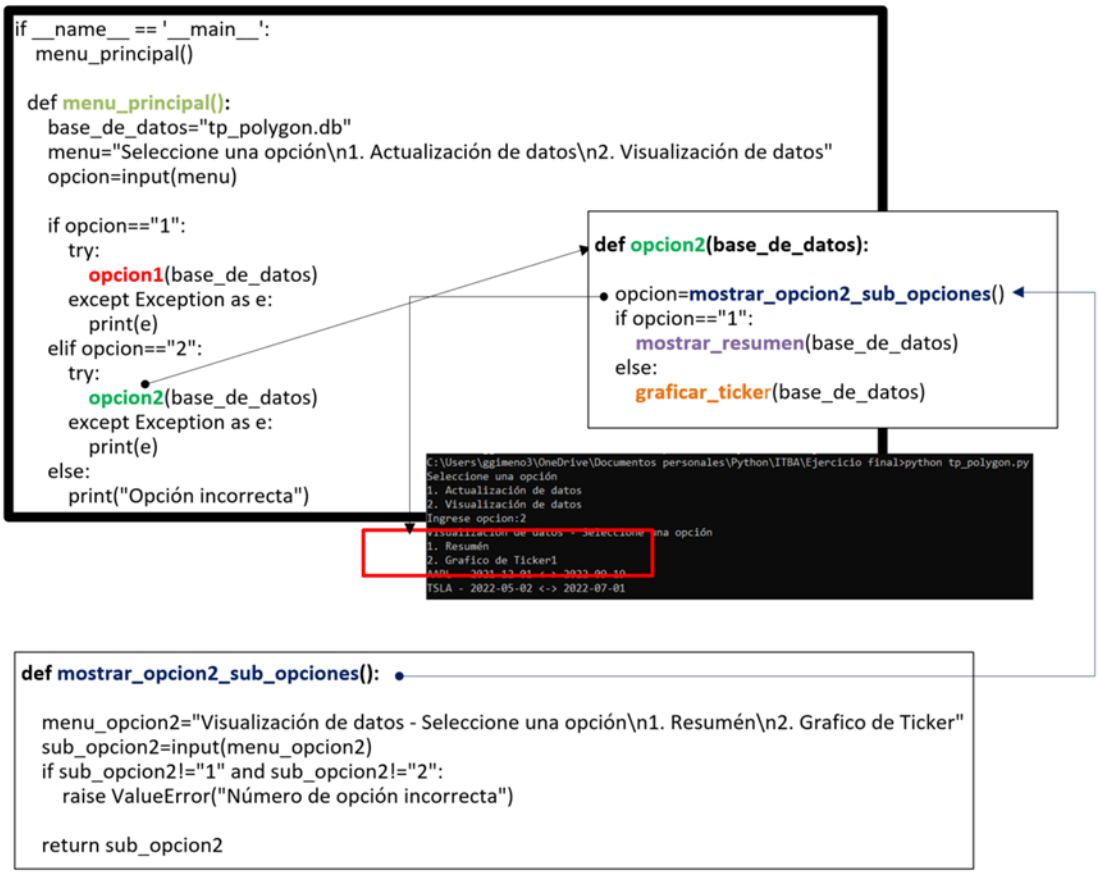
- Se ejecuta la conexión a la base y se crea un cursor. Luego creamos los campos de la base de datos indicando que tipo de valor es cada una. (Esto lo copie de un ejemplo, ya que no sabía que había que definirlo de esa manera y fue un prueba y error hasta que funciono). Para simplificar, cada campo representa los valores de la cadena de caracteres que devuelve el Json.

- Como segundo bloque de operadores, recorremos con un for los valores almacenados en info_polygon y obtenemos en forma ordenada el almacenamiento en la db de los valores que luego utilizaremos para nuestra grafica. Tanto Ticker nombre, v, t (tiempo) y c(cierre al día de la acción, son los valores que utilizaremos para la gráfica.

- Cuando se completa la creación y almacenado de los valores del ticket, el programa devuelve el mensaje "Operación exitosa"

1.4 Función opcion2 – Menues, Visualización y Grafica de tickers guardados.

1.4.1 Menues dentro de función mostrar_opcion_2_sub_opciones.



- La función opción2 dentro de la función del menú_principal, nos permite genera el menú correspondiente donde se visualiza las opciones Resumen y Grafica de Ticker.

- El if nos permite validar que, si las opciones son distintas de 1 o 2, que imprima opción incorrecta. Si la opción es 1, entonces el script continúa corriendo dentro de la función opcion2, encontrándose con un if que nos permite llamar a la opción “mostrar resumen” que veremos en el próximo punto.

1.4.2 Visualización de tickers guardados (Mostrar resumen).

- La función específica “mostrar_resumen” dentro de la función “opcion2”, nos permite consultar a la base de datos cuales son los ticker creados, por nombre, fecha inicio de consulta y fecha fin.

1.4.3 Grafica de Tickers guardados.

```
if __name__ == '__main__':
  menu_principal()

def menu_principal():
  base_de_datos="tp_polygon.db"
  menu="Seleccione una opción\n1. Actualización de datos\n2. Visualización de datos"
  opcion=input(menu)

  if opcion=="1":
    try:
      opcion1(base_de_datos)
    except Exception as e:
      print(e)
  elif opcion=="2":
    try:
      opcion2(base_de_datos)
    except Exception as e:
      print(e)
  else:
    print("Opción incorrecta")
```

```
def opcion2(base_de_datos):

  opcion=mostrar_opcion2_sub_opciones()
  if opcion=="1":
    mostrar_resumen(base_de_datos)
  else:
    graficar_ticker(base_de_datos)
```

```
def graficar_ticker(base_de_datos):

  print("graficar_ticker")
  texto_opcion_graficar_ticker = "Ingrese el acrónimo del ticker a graficar. Ejemplo AAPL\nNombre del Ticker: "
  ticker_elegido = input(texto_opcion_graficar_ticker)
  con = sqlite3.connect(dase_de_datos)
  cur = con.cursor()
  sql_select_ticker = "SELECT fecha,c FROM stocks WHERE ticker=:id_ticker ORDER BY fecha ASC"
  cur.execute(sql_select_ticker, {"id_ticker": ticker_elegido})
  datos = cur.fetchall()

  fechas = [x[0] for x in datos]
  valores_cierre = [x[1] for x in datos]

  plt.figure(figsize=(15, 10))
  plt.plot(fechas, valores_cierre)
  plt.ylabel('Valor al cierre')  # Leyenda en el eje y
  plt.xticks(rotation=45)
  plt.xlabel('Fecha')  # Leyenda en el eje x

  fecha_min = fechas[0]
  fecha_max = fechas[len(fechas) - 1]
  titulo = "Evolución ticker " + ticker_elegido + " desde " + fecha_min + " al " + fecha_max
  plt.title(titulo)  ## Título de Gráfica

  # plt.show() #Descomentar para mostrar en pantalla
  nombre_grafico = ticker_elegido + "-" + fecha_min + "-" + fecha_max + ".jpg"

  plt.savefig(nombre_grafico)
  con.close()
```

```
C:\Users\ggimeno3\OneDrive\Documentos personales\Python\ITBA\Ejercicio final>python  tp_polygon.py
Seleccione una opción
1. Actualización de datos
2. Visualización de datos
Ingrese opcion:2
Visualización de datos - Seleccione una opción
1. Resumén
2. Grafico de Ticker
Ingrese 1 o 2: 1
AAPL - 2022-07-01 <-> 2022-09-01

C:\Users\ggimeno3\OneDrive\Documentos personales\Python\ITBA\Ejercicio final>python  tp_polygon.py
Seleccione una opción
1. Actualización de datos
2. Visualización de datos
Ingrese opcion:2
Visualización de datos - Seleccione una opción
1. Resumén
2. Grafico de Ticker
Ingrese 1 o 2: 2
graficar_ticker
Ingrese el acrónimo del ticker a graficar. Ejemplo AAPL
Nombre del Ticker: AAPL

C:\Users\ggimeno3\OneDrive\Documentos personales\Python\ITBA\Ejercicio final>
```

- En el resumen arriba mostrado, se pueden ver como se solicita la gráfica de un determinado ticker. Como había comentado más arriba, no he podio logra que un ticker para una misma empresa, tenga fechas diferencias, solo vamos a poder graficar un ticker en una fecha desde y hasta. Si quiero tomar otro rango de fechas para la misma empresa, entonces, tengo dos opciones, o borro el archivo actual o amplio las fechas en donde se visualizan las variaciones de cotizaciones.

- Luego de generar los print correspondientes de visualización, se realiza la conexión a la base de datos, creación de un cursor para poder visualizar las tablas en SQLITE

- Los FOR que se encuentran incluidos dentro de fechas y valores cierre, los uso para recorrer todos los valores obtenidos con el comando fetchall(), con la única salvedad que solo el for obtendrá el valor en posición 0 para fechas y en posición 1 para Valores de Cierre.

```
fechas = [x[0] for x in datos]
valores_cierre = [x[1] for x in datos]
```

- El resto de los comandos de grafica en ejes x, y + la gráfica de los valores obtenidos mediante los FOR anteriores se ven plasmadas con los siguientes comandos.

```
plt.figure(figsize=(15, 10))
plt.plot(fechas, valores_cierre)
plt.ylabel('Valor al cierre')  # Leyenda en el eje y
plt.xticks(rotation=45)
plt.xlabel('Fecha')  # Leyenda en el eje x

fecha_min = fechas[0]
fecha_max = fechas[len(fechas) - 1]
```

- El resultado final es la gráfica que se muestra a continuación: