**KUNSKAPSKONTROLL 2 : FÖRDJUPAD PYTHONPROGRAMMERING**

**DOCUMENTATION**

**EC UTBILDNING**

**AIKATERINI ANTONIOU**

**INTRODUCTION**

In this project, a Python program was developed to automate the process of retrieving cryptocurrency data from the CoinGecko API, processing it and storing it in an SQLite database. The program was designed to handle errors efficiently and log them for future reference. Additionally, automated tests were implemented to ensure the accuracy of data handling. Finally, the Windows Task Scheduler was configured to execute the program automatically at a scheduled time, providing a complete automated pipeline for data retrieval, processing and storage. Thus, this project demonstrates the use of Python for API data retrieval, data processing, database management and automation. The tools and technologies utilized include Python, SQLite, the CoinGecko API and the Windows Task Scheduler.

The steps that were followed are showed below:

**STEP1: Fetching Data from the CoinGecko API**

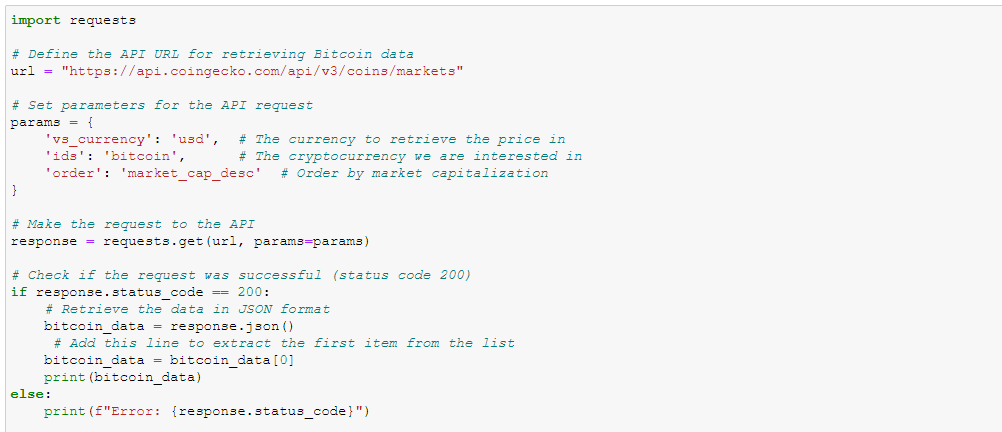
To retrieve data for Bitcoin, the CoinGecko API was used, which provides real-time information on cryptocurrencies. The /coins/markets endpoint was utilized to fetch the data with the following parameters:

vs\_currency: Set to "usd" to retrieve the price in US dollars.

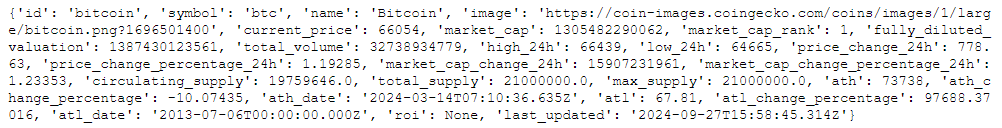
ids: Specified as "bitcoin" to focus on the Bitcoin cryptocurrency.

order: Set to "market\_cap\_desc" to sort by market capitalization.

The following code shows the API request and the check for a successful response (status code 200):



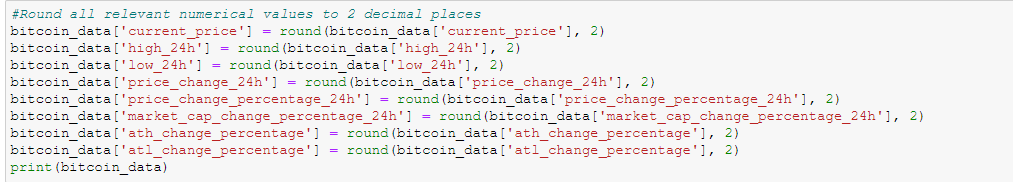
Example of the API response is the following:

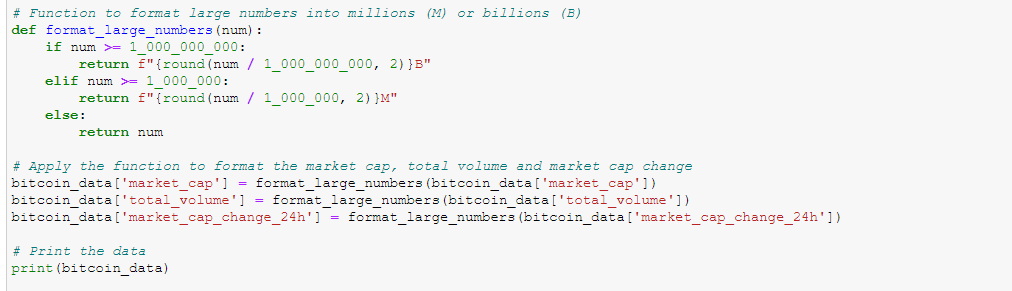


**STEP2: Data Processing, Rounding and Formatting**

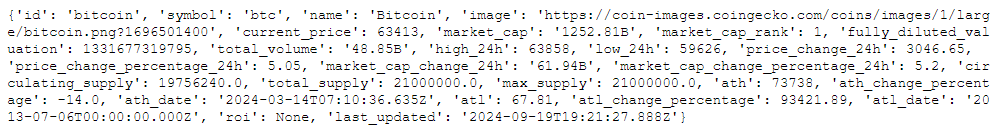
After fetching the data from the API, the numerical values were processed to be presented in a more user-friendly manner. This involved rounding the values to two decimal places and formatting large numbers (such as market capitalization) into millions (M) or billions (B).

The following code shows how this data processing and formatting was done:



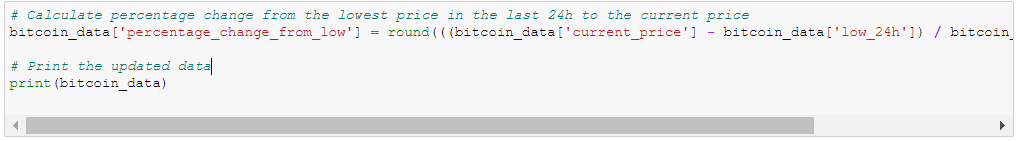


Examples after processing are the following:

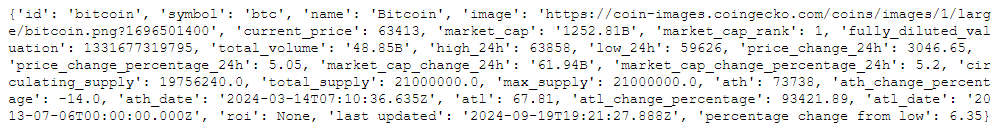


A further processing was applied. This included calculating the percentage change from the lowest price in the last 24 hours to the current price. This allows us to see how much the cryptocurrency's price has changed during the day, expressed as a percentage.

The code for calculating the percentage change is as follows:

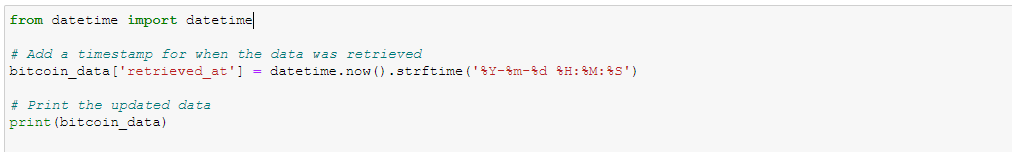


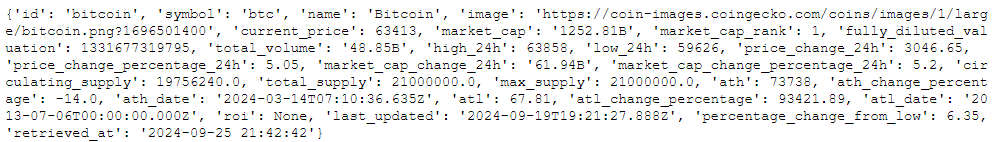
And the results are the following:



After retrieving the data and processes such as rounding numerical values, formatting large numbers, calculating the percentage change, adding a timestamp was applied. The timestamp records the exact time when the data was retrieved, providing a historical reference for each update.

The code for adding the timestamp is as follows having the following results:

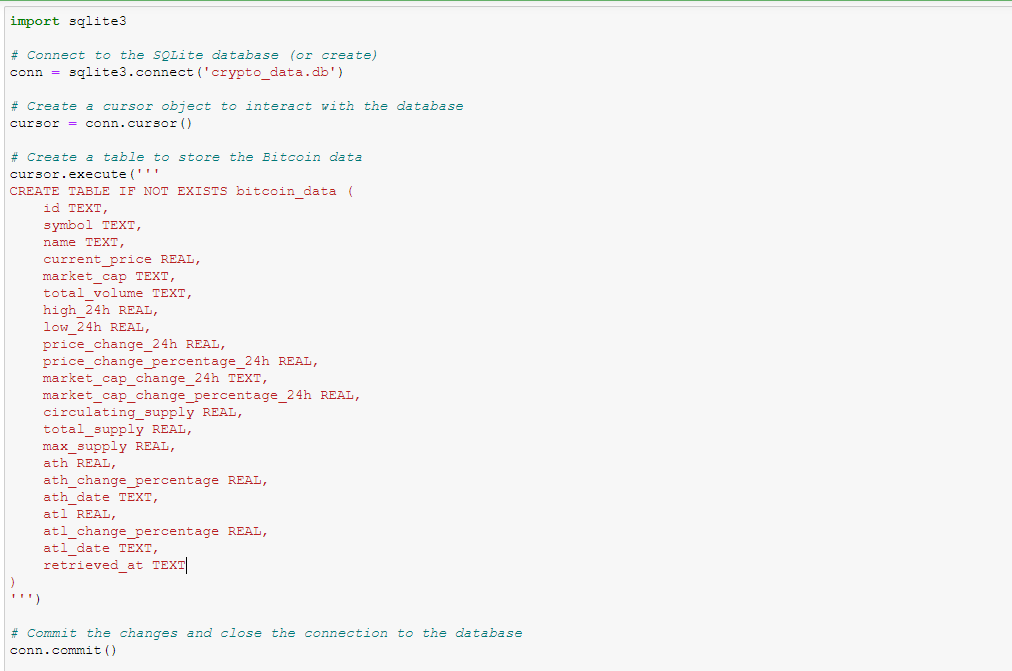


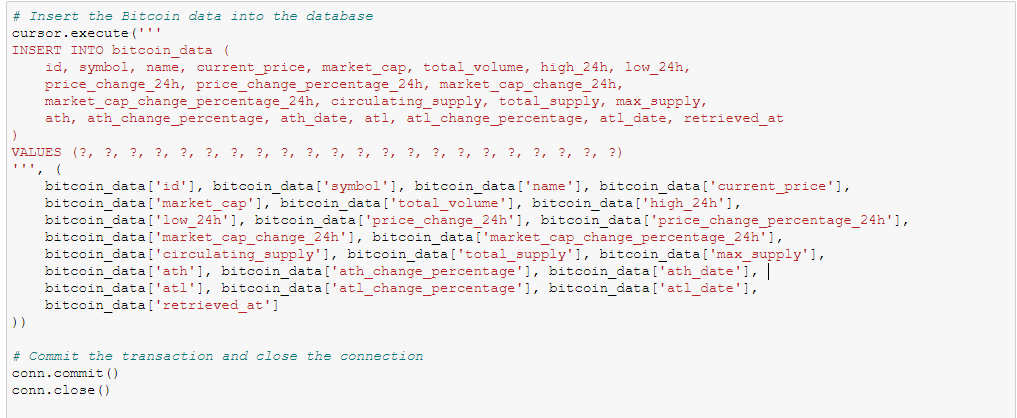


**Step 3: Storing Data in SQLite**

In this step an SQLite database was created to store the data retrieved from the API. A table was created using the CREATE TABLE statement, which includes all relevant fields such as current price, market capitalization, retrieval timestamp, etc. Once the table was created, the data was inserted using the INSERT INTO statement. Finally, the data insertion was verified by retrieving all records from the table with the SELECT statement.

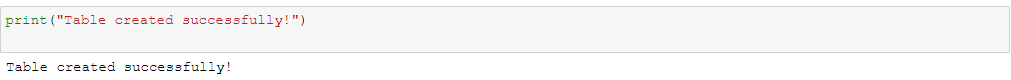
The code for creating the Database and inserting Data is depicted below:

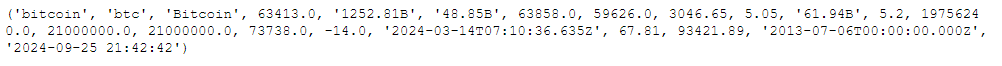




The code for retrieving and verifying Data is depicted below with the results coming next:







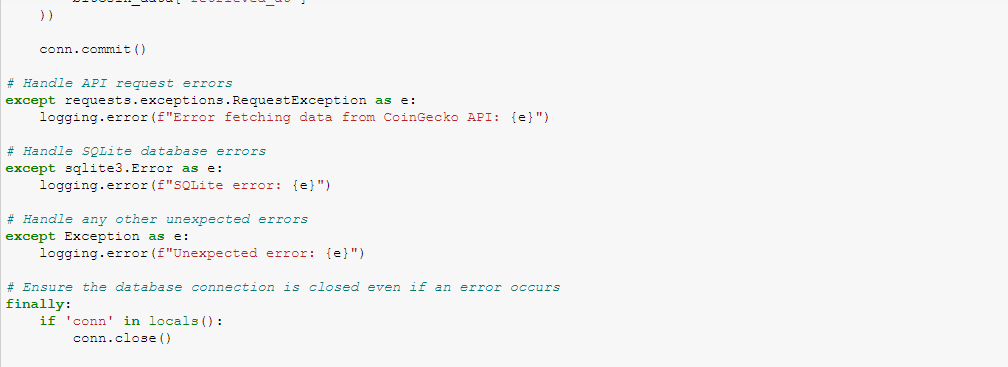
**Step 4: Error Handling and Logging**

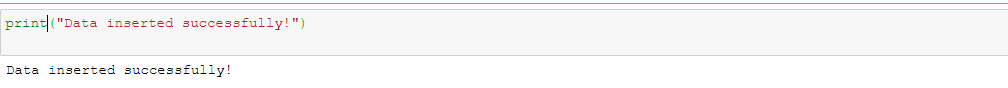
In this step, error handling was implemented to ensure that any error occurring during data retrieval or database interaction is properly logged. The crypto\_log.log file was used to record errors, while Python offers specific exceptions to handle issues that may arise with the API or the database. Additionally, using the try-except-finally structure ensures that the database connection closes properly, even if errors occur.

To ensure the proper functioning of the program, error handling was implemented at various stages of the process, such as during data retrieval from the API and during connection to the SQLite database. If any error occurs, it is logged into a log file with all the necessary details, such as the type of error and the time it occurred.

The following code demonstrates the structure used for error handling and logging taking the results that follow:



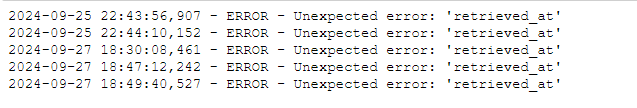




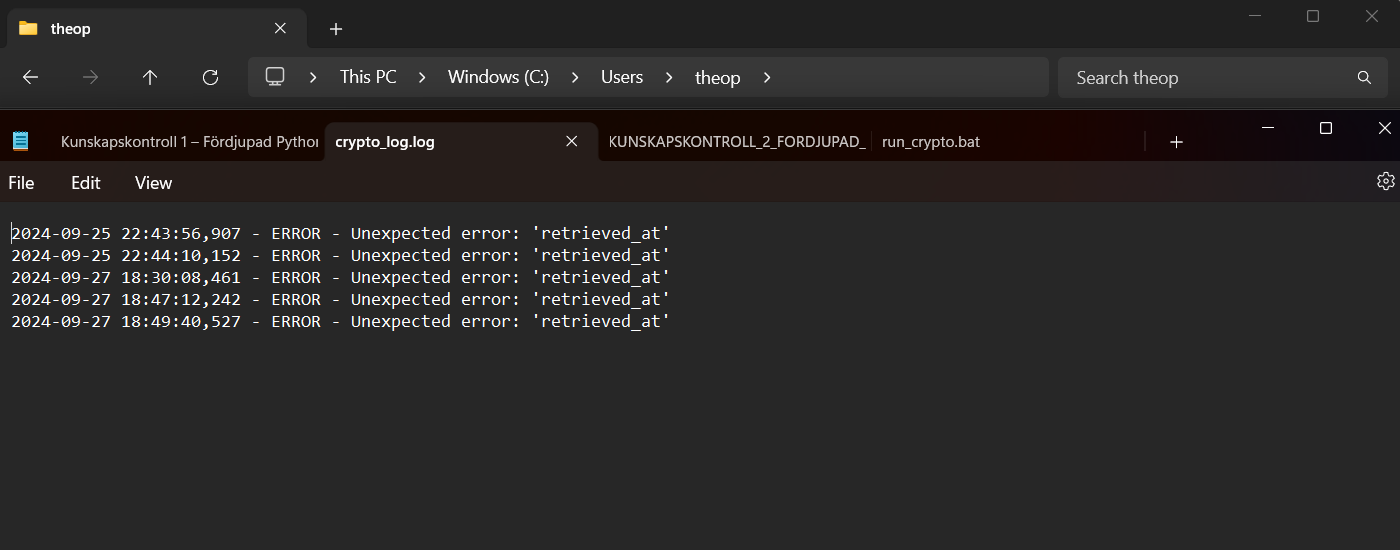
Error Analysis in the Log File

Upon reviewing the crypto\_log.log file, multiple errors related to the 'retrieved\_at' field were detected. These errors were logged at various timestamps. The issue likely occurs when attempting to access or insert the retrieved\_at field into the database. Logging these errors allows for easy tracking and identification of issues, enabling prompt resolution. In this case, it is essential to carefully examine the structure of the data and the method used to insert and store the retrieved\_at field in the database.

Monitoring the log file provides valuable insights into any errors, making it easier to address problems and improve the application's stability. Example errors from the log file follows (appearing as results in the code in Jupyter Notebook):



and as it appears opening the file crypto\_log.log



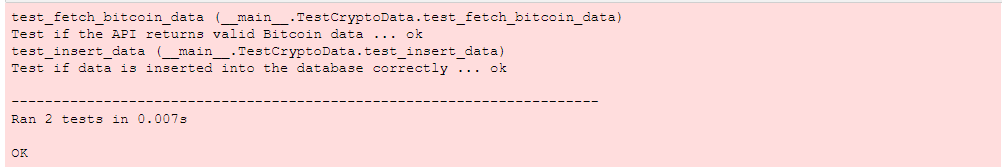
**Step 5: Automated Testing**

In this step, automated tests were created and executed to verify that the program functions as expected. Unit tests were designed for three critical components of the project: retrieving Bitcoin data from the API, inserting the data into the SQLite database, and error handling. Additionally, tests were implemented for formatting large numerical values. The tests were run using the unittest module in Python, and all tests passed successfully, confirming the program's functionality.

The code for verifying Data retrieval and insertion into the Database is below:



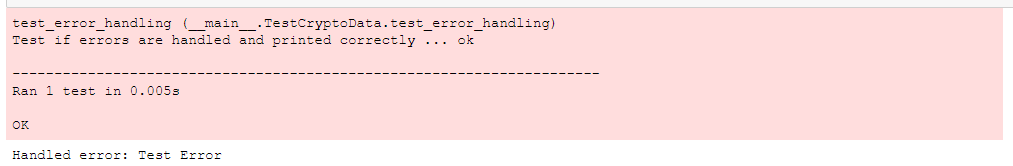
with the following results:



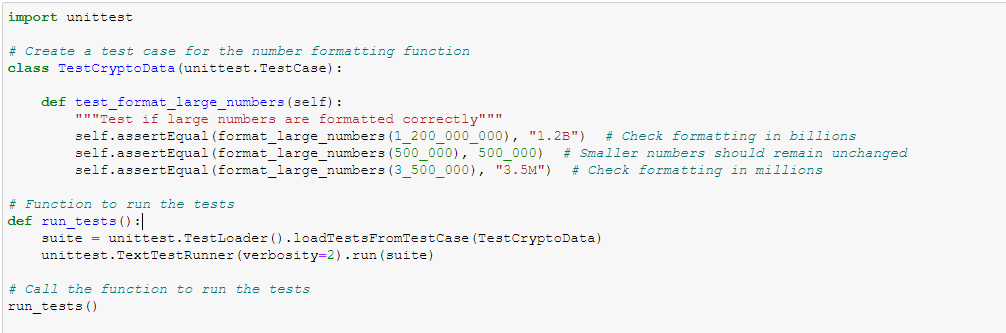
The code for Error Handling is as it shows below:



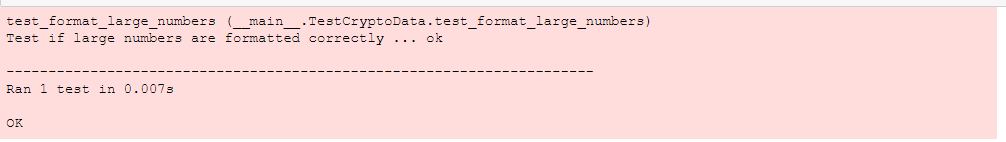
with the following results:



The code for Formatting Large Numbers follows next:



with the following results:



All these tests confirm that the program's core functionalities, including data retrieval, database insertion and number formatting are working correctly.

**Step 6: Automating the Script Execution with Windows Task Scheduler**

In this step, the Windows Task Scheduler was used to automate the execution of the Python script. Initially, a batch file named run\_crypto.bat was created to execute the Python script but an error message indicated that the system could not find the specified path. As a result, the decision was made to configure the Task Scheduler to execute the Python script directly, eliminating potential path issues.

Creating the Batch File

The batch file was created and named run\_crypto.bat. It contained the following command to run the Python script:

@echo off

cd "C:\Users\theop\Downloads" # path where is Python file

python "KUNSKAPSKONTROLL\_2\_FORDJUPAD\_PYTHONPROGRAMMERING.py" # name of Python file

pause

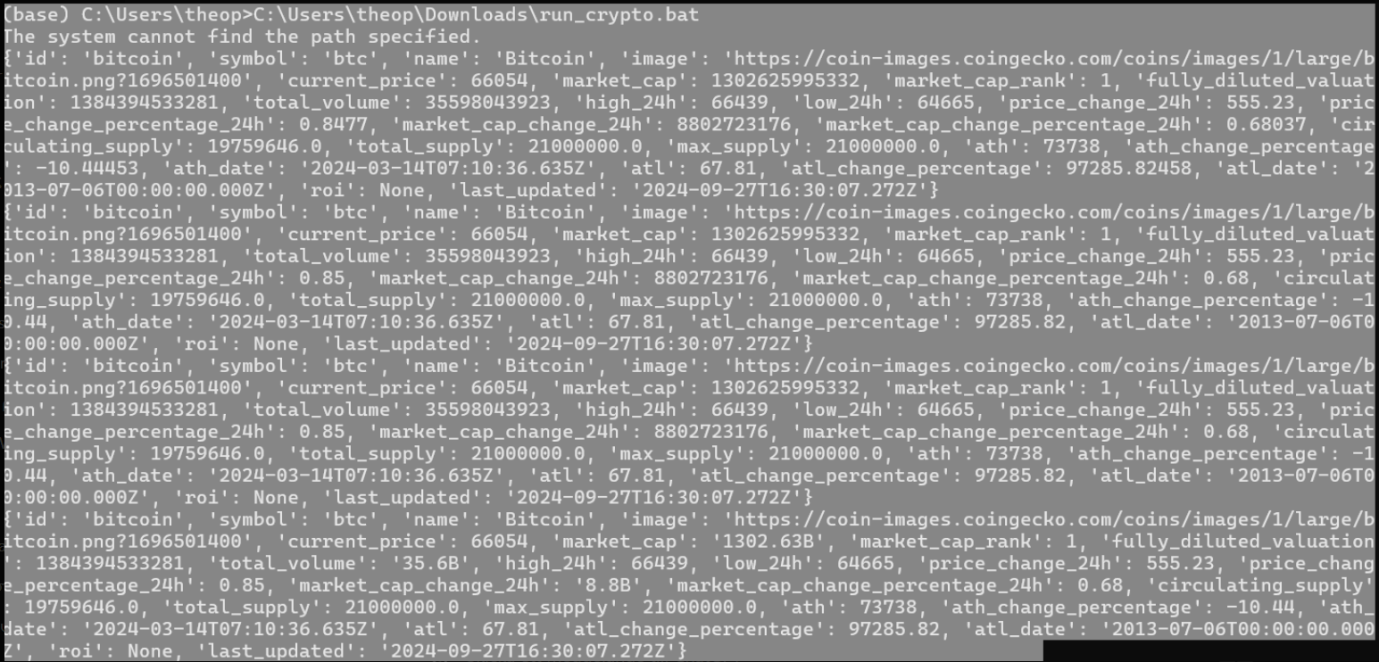
@echo off

python "KUNSKAPSKONTROLL\_2\_FORDJUPAD\_PYTHONPROGRAMMERING.py"

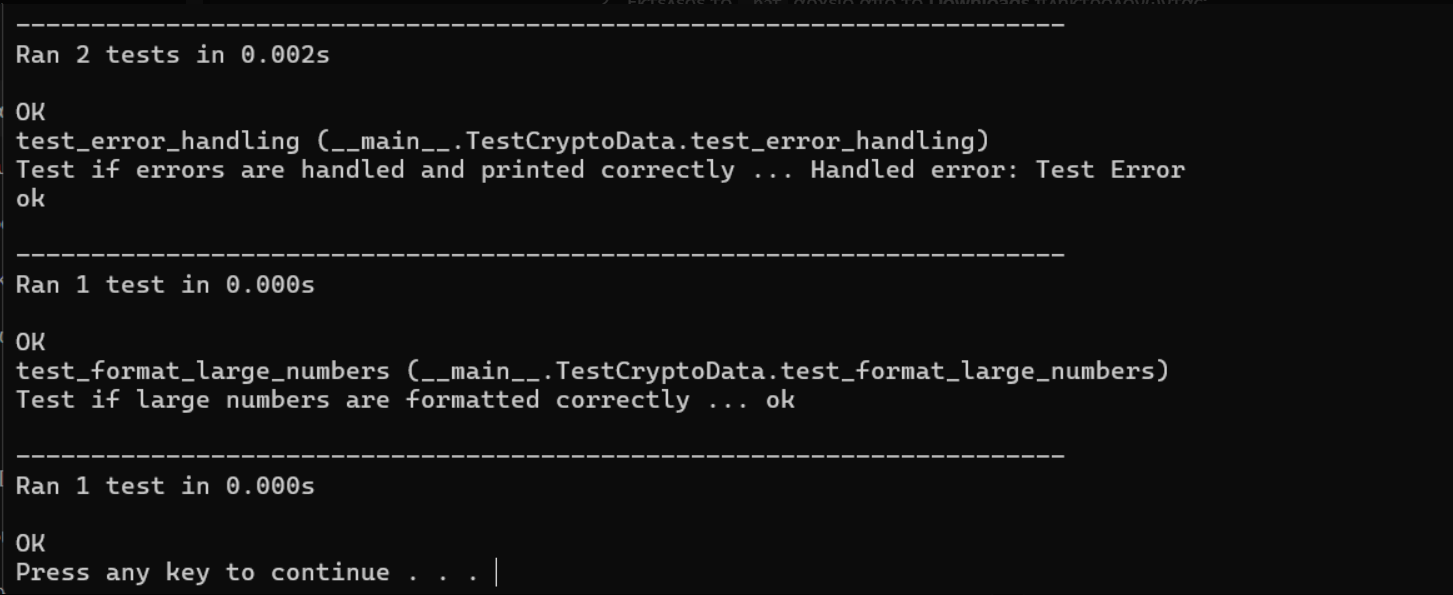
pause

The purpose of this file was to execute the Python script using the installed Python environment on the system. However, upon running the batch file, the following error appeared: "The system cannot find the path specified." To address this, it was decided to bypass the batch file and directly schedule the Python script in Task Scheduler.

Despite this message it shows results while running with Anaconda Prompt.







(C:\Users\theop\Downloads\run\_crypto.bat)

Running the code:

Python "C:\Users\theop\Downloads\KUNSKAPSKONTROLL\_2\_FORDJUPAD\_PYTHONPROGRAMMERING.py" direct in Anaconda Prompt is noticed that there is no the above message which means that the problem is with .bat file and not with python script.



Steps for Configuring Task Scheduler Without a Batch File:

I opened the Task Scheduler from the start menu.

I selected Create Task to set up a new task.

In the General tab i assigned a name to my task: Run Crypto Script.

In the Triggers tab i created a new trigger to define when the script should run (I chose daily at 9:30:00).

In the Actions tab:

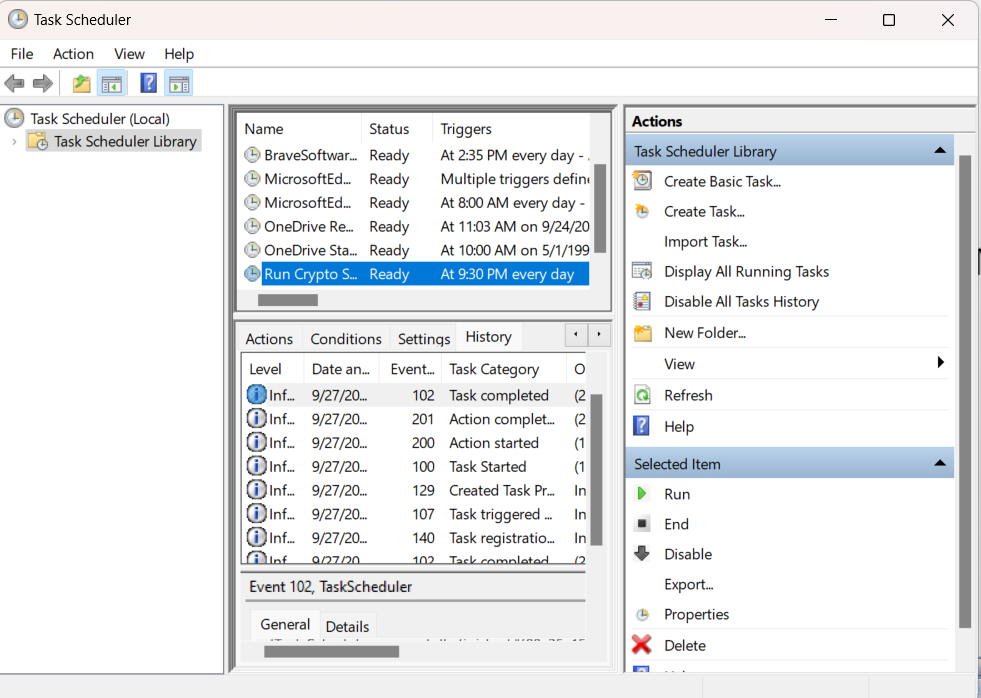
I clicked New and in the Program/Script field, entered python.

In the Add arguments field i entered the path to the Python script:

"C:\Users\theop\Downloads\KUNSKAPSKONTROLL 2\_FÖRDJUPAD PYTHONPROGRAMMERING.py".

In the Conditions tab, enabled the option Wake the computer to run this task to ensure the task runs even if the computer is asleep.

I saved the task, which is now ready for execution.



Verifying the Execution

To confirm the task was executed correctly the following steps were taken:

Checking Task History: In the Task Scheduler, the History tab was enabled to log the results of each task execution.

Log File Verification: The crypto\_log.log file was reviewed for any errors that occurred during script execution.

Task Scheduler Results: The task completed successfully with exit code (102), confirming the script ran correctly.