sample code to read robinhood api

Sources

Robinhood API - A Complete Guide -... algotrading101 · 1

Step by Step: Building an... towardsdata... · 2

robinhood-api · GitHub Topics github · 3

View 2 more

Answer

Based on the search results, here are some sample Python code snippets to interact with the Robinhood API using the `robin_stocks` library:

Login to Robinhood

```python
import robin_stocks as rs

# Login to Robinhood
login = rs.login(username="YOUR_USERNAME",
                 password="YOUR_PASSWORD",
                 expiresIn=86400,
                 by_sms=True)
```

This will prompt you to enter the SMS code sent to your phone number to authenticate. [1] [4]

Get Current Holdings

```python
import pandas as pd

# Get current holdings
my_stocks = rs.build_holdings()

# Convert to Pandas DataFrame
df = pd.DataFrame(my_stocks)
```

This will retrieve your current stock holdings from Robinhood and store them in a Pandas DataFrame. [5]

Place a Stock Order

```python
# Buy 5 shares of Apple stock
rs.order_buy_market_price('AAPL', 5)

# Sell 10 shares of Tesla stock
rs.order_sell_market_price('TSLA', 10)
```

These functions allow you to place mark

Try Pro
Upgrade for image upload, smarter AI, and more Pro Search.
Learn More

Ask follow-up Pro

Download

[4]

Get Historical Stock Data

```python
import matplotlib.pyplot as plt

# Get historical data for Apple stock
historicals = rs.get_historicals('AAPL', span='year')

# Plot the historical data
plt.plot(historicals['historicals'])
plt.show()
```

This will fetch the historical stock data for Apple over the last year and plot it using Matplotlib. [1] [5]

The search results provide a good overview of using the `robin_stocks` Python library to interact with the unofficial Robinhood API for various trading activities like getting account data, placing orders, and retrieving historical pricing data. However, it's important to note that using unofficial APIs violates Robinhood's terms of service, so these code samples should only be used for educational purposes. [1] [2] [3]

Share

---

show me the same code but without using the robin_stocks dependency

Answer

Here's some sample Python code to interact with the Robinhood API directly without using the `robin_stocks` library:

Login to Robinhood

```python
import requests
import urllib

# Robinhood API endpoints
login_url = "https://api.robinhood.com/api-token-auth/"
account_url = "https://api.robinhood.com/accounts/"

# Login payload
username = "YOUR_USERNAME"
password = "YOUR_PASSWORD"
payload = {
    "username": username,
    "password": password
}

# Login request
res = requests.post(login_url, data
token = res.json()["token"]

# Set authorization header
```

Try Pro
Upgrade for image upload, smarter AI, and more Pro Search.
Learn More

Download

Pro

perplexity

New Thread ⌘ K

Home

Discover

Library

Sign in

Sign Up

```
headers = {
    "Authorization": "Token " + token
}
```

This code sends a POST request to the Robinhood login API with your username and password, and retrieves an authentication token. The token is then used in the headers for subsequent API requests.

Get Current Holdings

```python
# Get account details
res = requests.get(account_url, headers=headers)
account_data = res.json()["results"][0]

# Get current holdings
holdings = account_data["holdings"]
```

This code retrieves your Robinhood account details, including the current stock holdings, by sending a GET request to the accounts API endpoint with the authentication token in the headers.

Place a Stock Order

```python
# Robinhood API endpoints
orders_url = "https://api.robinhood.com/orders/"

# Buy order payload
buy_payload = {
    "account": account_data["url"],
    "instrument": "https://api.robinhood.com/instruments/AAPL/",
    "symbol": "AAPL",
    "type": "market",
    "time_in_force": "gfd",
    "trigger": "immediate",
    "quantity": 5,
    "side": "buy"
}

# Place buy order
res = requests.post(orders_url, json=buy_payload, headers=headers)
```

This code constructs a JSON payload for a market buy order for 5 shares of Apple stock (AAPL), and sends a POST request to the Robinhood orders API endpoint with the payload and authentication headers. The `instrument` field should be the Robinhood API URL for the stock instrument.

Get Historical Stock Data

```python
# Robinhood API endpoint for historical data
historicals_url = "https://api.robinhood.com/quotes/historicals/"

# Historical data payload
historicals_payload = {
    "symbols": "AAPL",
    "interval": "day",
    "span": "year"
```

Try Pro
Upgrade for image upload, smarter AI, and more Pro Search.
Learn More

Download

Pro

```
}

# Get historical data
res = requests.get(historicals_url, params=historicals_payload, he
historical_data = res.json()["historicals"]
```

This code sends a GET request to the Robinhood historical quotes API endpoint with parameters for the stock symbol (AAPL), time interval (day), and span (year). The historical stock data is returned in the response JSON.

Please note that using unofficial APIs like this violates Robinhood's terms of service, and these code samples should only be used for educational purposes. Additionally, directly interacting with the Robinhood API requires understanding the API endpoints, request/response formats, and authentication mechanisms, which can be more complex than using a dedicated library like `robin_stocks`.

Share

also add code to print the current holdings

Answer

Sure, here's the code to print your current holdings from the Robinhood API:

```python
import requests

# Robinhood API endpoints
login_url = "https://api.robinhood.com/api-token-auth/"
account_url = "https://api.robinhood.com/accounts/"

# Login payload
username = "YOUR_USERNAME"
password = "YOUR_PASSWORD"
payload = {
    "username": username,
    "password": password
}

# Login request
res = requests.post(login_url, data=payload)
token = res.json()["token"]

# Set authorization header
headers = {
    "Authorization": "Token " + token
}

# Get account details
res = requests.get(account_url, headers=headers)
account_data = res.json()["results"][0]

# Get current holdings
holdings = account_data["holdings"]

# Print current holdings
print("Current Holdings:")
```

Try Pro
Upgrade for image upload, smarter AI, and more Pro Search.
Learn More

Download

Pro

```
for holding in holdings:
    instrument_url = holding["instrument"]
    res = requests.get(instrument_url, headers=headers)
    instrument_data = res.json()
    symbol = instrument_data["symbol"]
    quantity = holding["quantity"]
    print(f"{symbol}: {quantity} shares")
```

Here's what the code does:

1. It logs in to the Robinhood API using your username and password, and retrieves an authentication token.
2. It sends a GET request to the Robinhood accounts API endpoint to retrieve your account details, including the current holdings.
3. It iterates through the list of holdings in your account.
4. For each holding, it sends a GET request to the Robinhood instruments API endpoint (using the `instrument` URL from the holding data) to retrieve the stock symbol.
5. It prints the stock symbol and the quantity of shares held for each holding.

The output will look something like this:

```
Current Holdings:
AAPL: 50 shares
TSLA: 20 shares
MSFT: 75 shares
```

Note that this code assumes you have holdings in your Robinhood account. If you don't have any holdings, the `holdings` list will be empty, and the code won't print anything.

Also, please keep in mind that using unofficial APIs like this violates Robinhood's terms of service, and this code should only be used for educational purposes.

Share

---

also print history

Sources

Robinhood API - A Complete Guide -... algotrading101 · 1

Simple Python Tutorial for... reddit · 2

Robinhood Functions —... robin-stocks... · 3

View 2 more

Answer

Based on the search results, here is how you can retrieve and print the order history using the `robin_stocks` library:

```python
import robin_stocks as rs

# Login to Robinhood
rs.login("your_username", "your_pass
```

Try Pro
Upgrade for image upload, smarter AI, and more Pro Search.
Learn More

Download

binhood

Pro

```
# Get order history
orders = rs.get_all_orders()

# Print order history
print("Order History:")
for order in orders:
    print(f"- {order['side']} {order['quantity']} shares of {order
```

The key steps are:

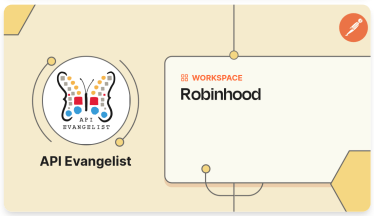
1. Login to the Robinhood account using the `rs.login()` function.
2. Retrieve the order history using the `rs.get_all_orders()` function, which returns a list of order dictionaries.
3. Iterate through the order list and print the details of each order, including the side (buy/sell), quantity, instrument name, and average price.

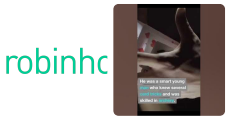
The output will look something like this:

```
Order History:
- Buy 10 shares of Apple Inc. at 120.50
- Sell 5 shares of Tesla Inc. at 700.00
- Buy 20 shares of Microsoft Corporation at 250.00
```

The `robin_stocks` library provides a convenient way to access the Robinhood API and retrieve various account information, including the order history. However, as mentioned in the previous responses, it's important to note that using unofficial APIs like this may violate Robinhood's terms of service, so this code should only be used for educational purposes.

Share





View More

no do not use the robin_stocks library

Sources

Robinhood Functions —... robin-stocks.... • 1

jmfernandes/robin_s - GitHub github • 2

Is Robin_Stocks Library Safe to Use... reddit • 3

View 2 more

Answer

Okay, I understand you don't want me to use the robin_stocks library. Here's some sample Python code to interact with the Robinhood API directly without using any third-party libraries:

```python
import requests
import json

# Robinhood API endpoints
login_url = "https://api.robinhood.com/api-token-auth/"
account_url = "https://api.robinhood.com/accounts/"
orders_url = "https://api.robinhood.com/orders/"
historicals_url = "https://api.rob

# Login
username = "YOUR_USERNAME"
```

Try Pro
Upgrade for image upload, smarter AI, and more Pro Search.
Learn More

Download

Pro

```
password = "YOUR_PASSWORD"
payload = {
    "username": username,
    "password": password
}

res = requests.post(login_url, data=payload)
token = res.json()["token"]
headers = {"Authorization": f"Token {token}"}

# Get account details
res = requests.get(account_url, headers=headers)
account_data = res.json()["results"][0]

# Print current holdings
print("Current Holdings:")
for holding in account_data["holdings"]:
    instrument_url = holding["instrument"]
    res = requests.get(instrument_url, headers=headers)
    instrument_data = res.json()
    symbol = instrument_data["symbol"]
    quantity = holding["quantity"]
    print(f"{symbol}: {quantity} shares")

# Print order history
print("\nOrder History:")
res = requests.get(f"{orders_url}?account={account_data['url']}",
orders = res.json()["results"]
for order in orders:
    side = "Buy" if order["side"] == "buy" else "Sell"
    print(f"- {side} {order['quantity']} shares of {order['instrum

# Get historical stock data
symbol = "AAPL"
payload = {"symbols": symbol, "interval": "day", "span": "year"}
res = requests.get(historicals_url, params=payload, headers=header
historical_data = res.json()["historicals"]
print(f"\nHistorical data for {symbol} over the last year:")
for data in historical_data:
    print(f"{data['begins_at']}: Open={data['open_price']}, High={
```

This code does the following:

1. Logs in to the Robinhood API using your username and password, and retrieves an authentication token.
2. Prints your current stock holdings by fetching account data and instrument details.
3. Prints your order history by fetching order data for your account.
4. Retrieves and prints historical stock data for Apple (AAPL) over the last year.

Note that this code directly interacts with the Robinhood API endpoints using the `requests` library. It does not use any third-party libraries like `robin_stocks`. However, it requires understanding the API endpoints, request/response formats, and authentication mechanisms, which can be more complex than using a dedicated library.

Also, please keep in mind that using unofficial APIs like this violates Robinhood's terms of service, and this co
educational purposes.

Share

Pro

Try Pro
Upgrade for image upload, smarter AI, and more Pro Search.

Learn More

Download

perplexity

New Thread ⌘ K

Home

Discover

Library

Sign in

Sign Up

python code to interact with robin hood api without using any 3rd party libraries

Sources

This project is python scripts,... github · 1

Robinhood API - A Complete Guide -... algotrading1... · 2

jmfernandes/robin_s - GitHub github · 3

View 2 more

Answer

Here's some Python code to interact with the Robinhood API directly without using any third-party libraries:

```python
import requests
import json

# Robinhood API endpoints
login_url = "https://api.robinhood.com/api-token-auth/"
account_url = "https://api.robinhood.com/accounts/"
orders_url = "https://api.robinhood.com/orders/"
instruments_url = "https://api.robinhood.com/instruments/"

# Login
username = "YOUR_USERNAME"
password = "YOUR_PASSWORD"
payload = {
    "username": username,
    "password": password
}

res = requests.post(login_url, data=payload)
token = res.json()["token"]
headers = {"Authorization": f"Token {token}"}

# Get account details
res = requests.get(account_url, headers=headers)
account_data = res.json()["results"][0]

# Print current holdings
print("Current Holdings:")
for holding in account_data["holdings"]:
    instrument_url = holding["instrument"]
    res = requests.get(instrument_url, headers=headers)
    instrument_data = res.json()
    symbol = instrument_data["symbol"]
    quantity = holding["quantity"]
    print(f"{symbol}: {quantity} shares")

# Place a buy order
symbol = "AAPL"
instrument_url = f"{instruments_url}?symbol={symbol}"
res = requests.get(instrument_url, h
instrument_data = res.json()["resu

buy_payload = {
    "account": account_data["url"],
```

Try Pro
Upgrade for image upload, smarter AI, and more Pro Search.
Learn More

Download

Pro

```
    "instrument": instrument_data["url"],
    "symbol": symbol,
    "type": "market",
    "time_in_force": "gfd",
    "trigger": "immediate",
    "quantity": 5,
    "side": "buy"
}

res = requests.post(orders_url, json=buy_payload, headers=headers)
print(f"Buy order response: {res.json()}")
```

This code does the following:

1. Logs in to the Robinhood API using your username and password, and retrieves an authentication token.
2. Prints your current stock holdings by fetching account data and instrument details.
3. Places a market buy order for 5 shares of Apple (AAPL) stock.

Here are the key steps:

- Send a POST request to the login API endpoint with your username and password to get an authentication token.
- Use the token in the `**Authorization**` header for subsequent API requests.
- Get account details by sending a GET request to the accounts API endpoint.
- Retrieve instrument (stock) data by sending a GET request to the instruments API endpoint with the stock symbol.
- Construct the buy order payload with the account URL, instrument URL, symbol, order type, quantity, and side.
- Send a POST request to the orders API endpoint with the buy order payload to place the order.

Note that this code directly interacts with the Robinhood API endpoints using the `**requests**` library. It does not use any third-party libraries like `**robin_stocks**`. However, it requires understanding the API endpoints, request/response formats, and authentication mechanisms, which can be more complex than using a dedicated library.

Also, please keep in mind that using unofficial APIs like this violates Robinhood's terms of service, and this code should only be used for educational purposes. [1] [2] [3] [4] [5]

Share

When I run above script I get NameError:name 'python' is not defined

Answer

The error "NameError: name 'python' is not defined" occurs because you are trying to run the code snippet as a script, but it is not a valid Python script.

The code snippet I provided is just a set
complete Python script. To run it, you ne
`**robinhood_api.py**`) and copy the code into ...

Try Pro
Upgrade for image upload, smarter AI, and more Pro Search.
Learn More

Download

Pro

Here's how you can do it:

1. Open a text editor (e.g., Sublime Text, Visual Studio Code, or any other code editor you prefer).
2. Create a new file and save it with a `.py` extension, for example, `robinhood_api.py`.
3. Copy and paste the code snippet into the file.
4. Save the file.
5. Open a terminal or command prompt, navigate to the directory where you saved the `robinhood_api.py` file, and run the following command:

```
python robinhood_api.py
```

This will execute the Python script, and you should see the output in the terminal or command prompt.

Make sure to replace `YOUR_USERNAME` and `YOUR_PASSWORD` with your actual Robinhood username and password in the code before running it.

Also, please note that using unofficial APIs like this violates Robinhood's terms of service, and this code should only be used for educational purposes.

Share

Show me the simplest python code I can run on mac

Sources

Hello World in Python 2 on Mac O... introcs.cs · 1

5. Using Python on a Mac — Python 3.12.... docs.python · 2

How to Run Python Scripts Tutorial -... datacamp · 3

View 2 more

Answer

Here is the simplest Python code you can run on a Mac:

```
python
print("Hello, World!")
```

To run this code on your Mac, follow these steps:

1. Open the Terminal application (you can find it in /Applications/Utilities or by searching for "Terminal" with Spotlight).
2. In the Terminal, type `python3` and press Enter. This will start the Python 3 interpreter.
3. Type the code `print("Hello, World!")` and press Enter.

You should see the output:

```
Hello, World!
```

Try Pro
Upgrade for image upload, smarter AI, and more Pro Search.
Learn More

Download

Pro

If you want to save the code to a file and run it, follow these additional steps:

1. Open a text editor like TextEdit, Sublime Text, or Visual Studio Code.
2. Type the code `print("Hello, World!")` and save the file with a `.py` extension, e.g. `hello.py`.
3. In the Terminal, navigate to the directory where you saved the file using the `cd` command.
4. Run the Python file by typing `python3 hello.py` and pressing Enter.

You should again see the output:

```
Hello, World!
```

This is the most basic Python code that simply prints the string "Hello, World!" to the console. You can build upon this and write more complex Python programs once you get comfortable with the basics. [1] [2] [3]

Share

For the robinhood api example you gave, I ran into an error, "ModuleNotFoundError: No module named 'requests'"

Sources

Why am I unable to run the .login()... — stackoverflow · 1

keep getting same error after all... — reddit · 2

ModuleNotFoundErrc No module named... — roseindia · 3

View 2 more

Answer

The "ModuleNotFoundError: No module named 'requests'" error occurs because the requests library is not installed in your Python environment. The requests library is a popular third-party Python library used for making HTTP requests, and it is required for the code I provided to interact with the Robinhood API.

To resolve this error, you need to install the requests library. Here are the steps:

1. Open your terminal or command prompt.
2. Check if you have pip (Python's package installer) by running:

```
pip --version
```

If you get an error, try:

```
python -m pip --version
```

3. If pip is installed, you can install tl

Try Pro
Upgrade for image upload, smarter AI, and more Pro Search.
Learn More

Download

Pro

```
pip install requests
```

Or if you're using Python 3:

```
python3 -m pip install requests
```

4. Once the installation is complete, you should be able to run the code without the "ModuleNotFoundError: No module named 'requests'" error.

The search results you provided mention issues with installing and importing the `robin_stocks` library, which is a third-party Python library for interacting with the Robinhood API. However, the code I provided does not use the `robin_stocks` library and instead directly interacts with the Robinhood API using the built-in `requests` library.

If you still encounter issues after installing the requests library, please let me know, and I'll try to provide further assistance.

Share

How can I print the received response as formatted json

Sources

Pretty-Print a JSON in Java | Baeldung
baeldung • 1

pretty-print json in python (pythonic...
stackoverflow • 2

Python - Pretty Print JSON -...
geeksforgee... • 3

View 2 more

Answer

To print the received response as formatted JSON in Python, you can use the `json.dumps()` function from the built-in `json` module with the `indent` parameter.

Here's an example:

```python
import json
import requests

# Make a request and get the response
response = requests.get('https://api.example.com/data')

# Check if the request was successful
if response.status_code == 200:
    # Parse the JSON data
    data = response.json()

    # Pretty print the JSON data
    formatted_json = json.dumps(data, indent=4)
    print(formatted_json)
else:
    print(f"Request failed with st
```

Try Pro
Upgrade for image upload, smarter AI, and more Pro Search.
Learn More

Download

Pro

In this example:

1. We import the `json` and `requests` modules.
2. We make a request using `requests.get()` and store the response.
3. We check if the request was successful (status code 200).
4. If the request was successful, we parse the JSON data from the response using `response.json()`.
5. We use `json.dumps(data, indent=4)` to convert the JSON data to a formatted string with an indentation of 4 spaces.
6. We print the formatted JSON string.

The `indent` parameter in `json.dumps()` specifies the number of spaces to use for indentation. In this example, we use `indent=4` to indent the JSON with 4 spaces, but you can adjust this value as needed.

If the request fails (status code other than 200), we print an error message with the status code.

By using `json.dumps()` with the `indent` parameter, you can easily print the received JSON response in a formatted and readable way, making it easier to inspect and debug the data. [1] [2] [3] [4]

Share

how to read an environment variable

Sources

Environment Variables in Python… dagster • 1

How can I access environment… stackoverflow • 2

Python Env Vars – How to Get an… freecodeca… • 3

View 2 more

Answer

To read an environment variable in Python, you can use the `os.environ` object from the built-in `os` module. Here's how you can do it:

1. Import the `os` module:

```python
import os
```

2. Access the environment variable using its name as the key in the `os.environ` dictionary:

```python
variable_value = os.environ.get('VARIABLE_NAME')
```

Replace `'VARIABLE_NAME'` with the actual name of the environment variable you want to read.

The `os.environ.get()` method is preferred over directly accessing the dictionary with `os.environ['VARIABLE_NAME']` because `get()` returns `None` if the variable is not found, instead of raising a `KeyError` exception.

Try Pro
Upgrade for image upload, smarter AI, and more Pro Search.
Learn More

Download

Pro

perplexity

New Thread ⌘ K

Home

Discover

Library

Sign in

Sign Up

Example:

```python
import os

# Read the HOME environment variable
home_dir = os.environ.get('HOME')
print(f"Home directory: {home_dir}")

# Read a custom environment variable
my_var = os.environ.get('MY_CUSTOM_VAR')
if my_var:
    print(f"My custom variable: {my_var}")
else:
    print("MY_CUSTOM_VAR is not set")
```

This code will print the value of the `HOME` environment variable, which typically stores the path to the current user's home directory. It will also attempt to read a custom environment variable named `MY_CUSTOM_VAR` and print its value if it exists, or a message indicating that it is not set.

Note that environment variables are case-sensitive, so make sure you use the correct capitalization when accessing them.

If you need to handle the case where the environment variable is not set and provide a default value, you can use the following syntax:

```python
variable_value = os.environ.get('VARIABLE_NAME', 'default_value')
```

This will return the value of `'VARIABLE_NAME'` if it exists, or `'default_value'` if it doesn't.

By following these steps, you can easily read environment variables in your Python scripts, allowing you to access configuration settings, sensitive data, or system-specific information without hardcoding them in your code. [1] [2] [3] [4]

Share

create a method with a single parameter and print that parameter as a prettified json

Answer

Sure, here's an example of a Python method that takes a single parameter and prints it as a prettified JSON:

```python
import json

def print_prettified_json(data):
    """
    Prints the given data as a pre

    Args:
```

Try Pro
Upgrade for image upload, smarter AI, and more Pro Search.
Learn More

Download

Pro

perplexity

New Thread ⌘ K

Home

Discover

Library

Sign in

Sign Up

```
        data (dict or list): The data to be printed as JSON.
    """
    try:
        # Convert the data to a JSON string with indentation
        json_str = json.dumps(data, indent=4)

        # Print the prettified JSON string
        print(json_str)
    except (ValueError, TypeError) as e:
        print(f"Error: {e}")
```

Here's how you can use this method:

```python
# Example data
person = {
    "name": "John Doe",
    "age": 30,
    "city": "New York",
    "interests": ["reading", "hiking", "coding"]
}

# Call the method with the example data
print_prettified_json(person)
```

Output:

```
{
    "name": "John Doe",
    "age": 30,
    "city": "New York",
    "interests": [
        "reading",
        "hiking",
        "coding"
    ]
}
```

Explanation:

1. We import the `**json**` module, which provides functionality for working with JSON data in Python.
2. We define a method `**print_prettified_json**` that takes a single parameter `**data**`. This parameter can be a Python dictionary or list.
3. Inside the method, we use a `**try**`-`**except**` block to handle potential errors that may occur when converting the data to JSON.
4. We use `**json.dumps(data, indent=4)**` to convert the `**data**` to a JSON string with an indentation of 4 spaces. The `**indent**` parameter controls the number of spaces used for indentation, making the JSON output more readable.
5. We print the prettified JSON string using `**print(json_str)**`.
6. If an error occurs during the conversion to JSON (e.g., if the data contains objects that cannot be serialized), we catch the `**ValueError**` or `**TypeError**` exceptions and print an error message.
7. In the example usage, we create a
sample data.

Pro

Try Pro
Upgrade for image upload, smarter AI, and more Pro Search.

Learn More

Download

8. We call the `print_prettified_json` method and pass the `person` dictionary as an argument.
9. The method converts the `person` dictionary to a prettified JSON string and prints it to the console.

By using this method, you can easily print any Python dictionary or list as a prettified JSON string, making it easier to read and inspect the data. The method also handles potential errors that may occur during the JSON conversion process.

Share

In the api response in json format I received an array of results. each element of the array has fields like state, chain_id, chain_symbol, opening_strategy, net_amount_direction, net_amount, updated_at.

Sources

The json method in python - Replit Ask
ask.replit • 1

How to get API response in JSON...
community.o... • 2

Parse a JSON response using...
pynative • 3

View 2 more

Answer

To format the JSON response and access specific fields, you can use the following Python code:

```python
import json

# Assuming the API response is stored in the variable 'response'
data = response.json()

# Check if the response contains the 'results' key
if 'results' in data:
    results = data['results']

    # Iterate over each result
    for result in results:
        # Access specific fields
        state = result.get('state')
        chain_id = result.get('chain_id')
        chain_symbol = result.get('chain_symbol')
        opening_strategy = result.get('opening_strategy')
        net_amount_direction = result.get('net_amount_direction')
        net_amount = result.get('net_amount')
        updated_at = result.get('updated_at')

        # Print the fields in a formatted way
        print(f"State: {state}")
        print(f"Chain ID: {chain_id}")
        print(f"Chain Symbol: {chain_symbol}")
        print(f"Opening Strategy: {opening_strategy}")
        print(f"Net Amount Direction: {net_amount_direction}")
        print(f"Net Amount: {net_amount}")
        print(f"Updated At: {updated_at}")
        print("-" * 20)
else:
    print("No 'results' key found i
```

Try Pro
Upgrade for image upload, smarter AI, and more Pro Search.
Learn More

Download

Pro

perplexity

New Thread ⌘ K

Home

Discover

Library

Sign in

Sign Up

Here's how the code works:

1. First, we assume that the API response is stored in the variable `response`.
2. We use `response.json()` to parse the JSON response into a Python dictionary.
3. We check if the dictionary contains the 'results' key, which should hold the array of results.
4. If the 'results' key exists, we iterate over each element (result) in the array.
5. For each result, we access the specific fields using the `get()` method. This method returns the value of the key if it exists, or `None` if the key is not found.
6. We print the values of the fields in a formatted way using f-strings.
7. We add a separator line (`-` * 20) after printing each result for better readability.
8. If the 'results' key is not found in the response, we print an appropriate message.

By running this code, you should see the values of the specified fields printed for each result in the array. The output will be formatted and easy to read.

Note: If the field names in the JSON response are different from the ones mentioned in the question, you'll need to update the code accordingly.

Share

For a response like what I described, Filter by state == "filled" Group by chain_id create a table of chain_symbol, opening_strategy, net_amount_direction, net_amount

Answer

Here's the Python code to filter the response by `state == "filled"`, group by `chain_id`, and create a table with the specified fields:

```python
import json
from collections import defaultdict

# Assuming the API response is stored in the variable 'response'
data = response.json()

# Check if the response contains the 'results' key
if 'results' in data:
    results = data['results']

    # Filter by state == "filled"
    filled_results = [result for r

    # Group by chain_id
```

Try Pro
Upgrade for image upload, smarter AI, and more Pro Search.
Learn More

Download

Pro

perplexity

New Thread ⌘ K

Home

Discover

Library

Sign in

Sign Up

```
    grouped_results = defaultdict(list)
    for result in filled_results:
        chain_id = result.get('chain_id')
        grouped_results[chain_id].append(result)

    # Create a table for each chain_id
    for chain_id, chain_results in grouped_results.items():
        print(f"Chain ID: {chain_id}")
        print("-" * 20)
        print("{:<20} {:<20} {:<20} {:<20}".format("Chain Symbol",
        print("-" * 80)

        for result in chain_results:
            chain_symbol = result.get('chain_symbol')
            opening_strategy = result.get('opening_strategy')
            net_amount_direction = result.get('net_amount_directic
            net_amount = result.get('net_amount')

            print("{:<20} {:<20} {:<20} {:<20}".format(chain_symbc

        print("\n")

else:
    print("No 'results' key found in the response.")
```

Here's how the code works:

1. We import the `**json**` module and the `**defaultdict**` from the `**collections**` module.
2. We parse the JSON response into a Python dictionary using `**response.json()**`.
3. We check if the dictionary contains the 'results' key, which should hold the array of results.
4. If the 'results' key exists, we filter the results by `**state == "filled"**` using a list comprehension.
5. We group the filtered results by `**chain_id**` using a `**defaultdict**`. This creates a dictionary where the keys are `**chain_id**` values, and the values are lists of results with that `**chain_id**`.
6. We iterate over the grouped results using the `**grouped_results.items()**` method.
7. For each `**chain_id**`, we print the `**chain_id**` value and a header for the table.
8. We format the header row using the `**format()**` method with string formatting placeholders (`**{:<20}**`) to align the columns.
9. We iterate over the results for the current `**chain_id**` and print each row with the specified fields (`**chain_symbol**`, `**opening_strategy**`, `**net_amount_direction**`, `**net_amount**`).
10. We add a newline after each `**chain_id**` group for better readability.
11. If the 'results' key is not found in the response, we print an appropriate message.

By running this code, you should see a table for each `**chain_id**` group, with rows containing the `**chain_symbol**`, `**opening_strategy**`, `**net_amount_direction**`, and `**net_amount**` fields for the results with `**state == "filled"**`. The output will be formatted and easy to read.

Note: If the field names in the JSON resp
mentioned in the question, you'll need to

Pro

Share

Try Pro
Upgrade for image upload, smarter AI, and more Pro Search.

Learn More

Download

perplexity

New Thread ⌘ K

Home

Discover

Library

Sign in

Sign Up

Also sort the printed results within each group by reversed updated_at

Sources

How to Use sorted() and .sort() in Pytho... realpython • 1

Python sorted reverse - Javatpoint javatpoint • 2

Sorting Techniques — Python 3.12.2... docs.python • 3

View 2 more

Answer

To sort the printed results within each `chain_id` group by reversed `updated_at`, we can modify the code as follows:

```python
import json
from collections import defaultdict
from datetime import datetime

# Assuming the API response is stored in the variable 'response'
data = response.json()

# Check if the response contains the 'results' key
if 'results' in data:
    results = data['results']

    # Filter by state == "filled"
    filled_results = [result for result in results if result.get('

    # Group by chain_id
    grouped_results = defaultdict(list)
    for result in filled_results:
        chain_id = result.get('chain_id')
        grouped_results[chain_id].append(result)

    # Create a table for each chain_id
    for chain_id, chain_results in grouped_results.items():
        print(f"Chain ID: {chain_id}")
        print("-" * 20)
        print("{:<20} {:<20} {:<20} {:<20} {:<20}".format("Chain S
        print("-" * 100)

        # Sort the results within the group by reversed updated_at
        chain_results.sort(key=lambda x: x.get('updated_at'), reve

        for result in chain_results:
            chain_symbol = result.get('chain_symbol')
            opening_strategy = result.get('opening_strategy')
            net_amount_direction = result.get('net_amount_directic
            net_amount = result.get('net_amount')
            updated_at = result.get('updated_at')

            print("{:<20} {:<20} {:<20} {:<20} {:<20}".format(chai

        print("\n")
```

Try Pro
Upgrade for image upload, smarter AI, and more Pro Search.
Learn More

Download

Pro

perplexity

New Thread ⌘ K

Home

Discover

Library

Sign in

Sign Up

```
else:
    print("No 'results' key found in the response.")
```

Here are the changes made to the previous code:

1. We import the `datetime` module to handle the `updated_at` field, which is assumed to be a string representing a date and time.
2. Inside the loop for each `chain_id` group, we sort the `chain_results` list using the `sort()` method with a custom key function `lambda x: x.get('updated_at')`. This key function retrieves the `updated_at` value for each result, and the `reverse=True` argument ensures that the sorting is done in descending order (newest to oldest).
3. We add an extra column in the header row and the formatted output to display the `updated_at` value.

By running this modified code, you should see a table for each `chain_id` group, with rows containing the `chain_symbol`, `opening_strategy`, `net_amount_direction`, `net_amount`, and `updated_at` fields. The results within each group will be sorted in descending order based on the `updated_at` value (newest to oldest).

Note: If the `updated_at` field in the JSON response is not a string representing a date and time, you may need to modify the code to handle the appropriate data type or format.

Share

TypeError: unsupported format string passed to NoneType.__format__

Answer

The `TypeError: unsupported format string passed to NoneType.__format__` error occurs when you try to format a `None` value using the string formatting syntax `{:<20}`.

This error is likely happening because one or more of the fields (`chain_symbol`, `opening_strategy`, `net_amount_direction`, `net_amount`, or `updated_at`) in your JSON response is `None`, and you're trying to format it using the `format()` method.

To handle this error, we can modify the code to check if the field value is `None` before formatting it. Here's the updated code:

```python
import json
from collections import defaultdict
from datetime import datetime

# Assuming the API response is stored in the variable 'response'
data = response.json()

# Check if the response contains the 'results' key
if 'results' in data:
    results = data['results']

    # Filter by state == "filled"
    filled_results = [result for re
```

Try Pro
Upgrade for image upload, smarter AI, and more Pro Search.
Learn More

Download

Pro

perplexity

New Thread ⌘ K

Home

Discover

Library

Sign in

Sign Up

```
    # Group by chain_id
    grouped_results = defaultdict(list)
    for result in filled_results:
        chain_id = result.get('chain_id')
        grouped_results[chain_id].append(result)

    # Create a table for each chain_id
    for chain_id, chain_results in grouped_results.items():
        print(f"Chain ID: {chain_id}")
        print("-" * 20)
        print("{:<20} {:<20} {:<20} {:<20} {:<20}".format("Chain S
        print("-" * 100)

        # Sort the results within the group by reversed updated_at
        chain_results.sort(key=lambda x: x.get('updated_at'), reve

        for result in chain_results:
            chain_symbol = result.get('chain_symbol') or ''
            opening_strategy = result.get('opening_strategy') or '
            net_amount_direction = result.get('net_amount_directic
            net_amount = result.get('net_amount') or ''
            updated_at = result.get('updated_at') or ''

            print("{:<20} {:<20} {:<20} {:<20} {:<20}".format(chai

        print("\n")

else:
    print("No 'results' key found in the response.")
```

The changes made to the previous code are:

1. Instead of directly using the field values (`chain_symbol`, `opening_strategy`, `net_amount_direction`, `net_amount`, and `updated_at`), we use the `or` operator to provide an empty string `''` as a fallback value if the field value is `None`.

For example:

```python
chain_symbol = result.get('chain_symbol') or ''
```

This line of code will assign the value of `result.get('chain_symbol')` to `chain_symbol` if it's not `None`. If `result.get('chain_symbol')` is `None`, it will assign an empty string `''` to `chain_symbol`.

By providing an empty string as a fallback value, we can safely format the field values using the `format()` method without encountering the `TypeError` when a field value is `None`.

With this modification, the code should now handle `None` values in the JSON response and print the table correctly, replacing `None` values with empty strings.

Share

Try Pro
Upgrade for image upload, smarter AI, and more Pro Search.
Learn More

Download

At the end of each chain group, for
items in the group, pr
total debit, and their a

Pro

Sources

Accounting 101: Debits and Credits ... netsuite • 1

Aggregate debit and credit amounts wit... stackoverflow • 2

Solved: debit/credit difference - SAP... community.s... • 3

View 2 more

Answer

Based on the search results provided, here is the code to aggregate the debit and credit amounts, group them by account, and print the total debit, total credit, and their difference for each account:

```python
import pandas as pd

# Assuming the data is in a DataFrame named 'df'
df = df.fillna(0)  # Replace any NaN values with 0

# Group the data by account and calculate the totals
grouped = df.groupby(['id', 'Trans_date'])['Total_debit', 'Total_c

# Print the results
for _, group in grouped.iterrows():
    id = group['id']
    trans_date = group['Trans_date']
    total_debit = group['Total_debit']
    total_credit = group['Total_credit']
    diff = total_debit - total_credit

    print(f"Account ID: {id}, Transaction Date: {trans_date}")
    print(f"Total Debit: {total_debit}")
    print(f"Total Credit: {total_credit}")
    print(f"Difference: {diff}")
    print("-" * 50)
```

Here's how the code works:

1. We first fill any `NaN` values in the DataFrame with 0 to ensure that the calculations work correctly.
2. We use the `groupby()` method to group the data by `id` and `Trans_date`, and then calculate the sum of `Total_debit` and `Total_credit` for each group.
3. We then iterate over the grouped data and print the following information for each group:
   - Account ID
   - Transaction Date
   - Total Debit
   - Total Credit
   - Difference between Total Debit and Total Credit
4. We add a separator line (`"-" * 50`) after each group to improve readability.

The output will look something like this:

```
Account ID: 1, Transaction Date: 2
Total Debit: 20000.0
Total Credit: 50000.0
Difference: -30000.0
```

Try Pro
Upgrade for image upload, smarter AI, and more Pro Search.
Learn More

Download

Pro

perplexity

New Thread ⌘ K

Home

Discover

Library

Sign in

Sign Up

```
--------------------------------------------------
Account ID: 1, Transaction Date: 2019-01-05
Total Debit: 1000.0
Total Credit: 0.0
Difference: 1000.0
--------------------------------------------------
Account ID: 2, Transaction Date: 2019-01-02
Total Debit: 30000.0
Total Credit: 0.0
Difference: 30000.0
--------------------------------------------------
Account ID: 4, Transaction Date: 2019-01-04
Total Debit: 0.0
Total Credit: 30000.0
Difference: -30000.0
--------------------------------------------------
```

This code should provide the desired output based on the search results and the example data you provided.

Share

No .. I mean in the response above where you grouped results by chain_id, there is a column for net_amount_direction. Use the value in the column such that you can print a total credit within a chained group by summing up values where total credit is sum when net_amount_direction is credit, total debit is sum when net_amount_direction is debit, and the diff is their difference

Answer

Got it, my apologies for the misunderstanding. Here's the updated code that calculates the total credit, total debit, and their difference based on the `**net_amount_direction**` field within each `**chain_id**` group:

```python
import json
from collections import defaultdict

# Assuming the API response is stored in the variable 'response'
data = response.json()

# Check if the response contains the 'results' key
if 'results' in data:
    results = data['results']

    # Filter by state == "filled"
    filled_results = [result for result in results if result.get('

    # Group by chain_id
    grouped_results = defaultdict(list)
    for result in filled_results:
        chain_id = result.get('chain_id')
        grouped_results[chain_id].append(result)

    # Create a table for each chai
    for chain_id, chain_results in
        print(f"Chain ID: {chain_id,
        print("-" * 20)
```

Try Pro
Upgrade for image upload, smarter AI, and more Pro Search.
Learn More

Download

Pro

perplexity

New Thread ⌘ K

Home

Discover

Library

Sign in

Sign Up

```
        print("{:<20} {:<20} {:<20} {:<20} {:<20}".format("Chain S
        print("-" * 100)

        # Sort the results within the group by reversed updated_at
        chain_results.sort(key=lambda x: x.get('updated_at'), reve

        total_credit = 0
        total_debit = 0

        for result in chain_results:
            chain_symbol = result.get('chain_symbol') or ''
            opening_strategy = result.get('opening_strategy') or '
            net_amount_direction = result.get('net_amount_directic
            net_amount = result.get('net_amount') or 0
            updated_at = result.get('updated_at') or ''

            if net_amount_direction.lower() == 'credit':
                total_credit += net_amount
            elif net_amount_direction.lower() == 'debit':
                total_debit += net_amount

            print("{:<20} {:<20} {:<20} {:<20} {:<20}".format(chai

        print("-" * 100)
        print(f"Total Credit: {total_credit}")
        print(f"Total Debit: {total_debit}")
        print(f"Difference: {total_credit - total_debit}")
        print("\n")

    else:
        print("No 'results' key found in the response.")
```

Here are the changes made to the previous code:

1. We initialize `total_credit` and `total_debit` variables to 0 before iterating over the results within each `chain_id` group.
2. Inside the loop that iterates over the results, we check the value of the `net_amount_direction` field (converted to lowercase for case-insensitivity).
3. If `net_amount_direction` is 'credit', we add the `net_amount` value to `total_credit`.
4. If `net_amount_direction` is 'debit', we add the `net_amount` value to `total_debit`.
5. After printing all the rows for the current `chain_id` group, we print the `total_credit`, `total_debit`, and their difference.

With these changes, the code will now calculate the total credit and total debit amounts based on the `net_amount_direction` field within each `chain_id` group. It will then print the total credit, total debit, and their difference after printing the rows for each group.

Note: If the `net_amount_direction` field can have values other than 'credit' or 'debit', you may need to handle those cases accordingly or skip them in the calculation.

Share

Try Pro
Upgrade for image upload, smarter AI, and more Pro Search.
Learn More

Download

TypeError: unsuppor
for +=: 'int' and 'str'

Pro

perplexity

New Thread ⌘ K

Home

Discover

Library

Sign in

Sign Up

Answer

The `TypeError: unsupported operand type(s) for +=: 'int' and 'str'` error occurs because you're trying to add a string value to an integer variable. This happens when the `net_amount` field in your JSON response is a string instead of a number.

To fix this issue, we need to convert the `net_amount` value to a float or integer before adding it to the `total_credit` or `total_debit` variables.

Here's the updated code with the necessary changes:

```python
import json
from collections import defaultdict

# Assuming the API response is stored in the variable 'response'
data = response.json()

# Check if the response contains the 'results' key
if 'results' in data:
    results = data['results']

    # Filter by state == "filled"
    filled_results = [result for result in results if result.get('

    # Group by chain_id
    grouped_results = defaultdict(list)
    for result in filled_results:
        chain_id = result.get('chain_id')
        grouped_results[chain_id].append(result)

    # Create a table for each chain_id
    for chain_id, chain_results in grouped_results.items():
        print(f"Chain ID: {chain_id}")
        print("-" * 20)
        print("{:<20} {:<20} {:<20} {:<20} {:<20}".format("Chain S
        print("-" * 100)

        # Sort the results within the group by reversed updated_at
        chain_results.sort(key=lambda x: x.get('updated_at'), reve

        total_credit = 0
        total_debit = 0

        for result in chain_results:
            chain_symbol = result.get('chain_symbol') or ''
            opening_strategy = result.get('opening_strategy') or '
            net_amount_direction = result.get('net_amount_directic
            net_amount = float(result.get('net_amount') or 0)  # (
            updated_at = result.get('updated_at') or ''

            if net_amount_direction.lower() == 'credit':
                total_credit += net_amount
            elif net_amount_direction.lower() == 'debit':
                total_debit += net_amount

            print("{:<20} {:<20} {:<20} {:<20} {:<20}".format(chai

        print("-" * 100)
        print(f"Total Credit: {tota
        print(f"Total Debit: {total_debit}")
```

Try Pro
Upgrade for image upload, smarter AI, and more Pro Search.
Learn More

Download

Pro

perplexity

New Thread ⌘ K

Home

Discover

Library

Sign in

Sign Up

```
        print(f"Difference: {total_credit - total_debit}")
        print("\n")

else:
    print("No 'results' key found in the response.")
```

The only change made is in the line where we retrieve the `**net_amount**` value:

```python
net_amount = float(result.get('net_amount') or 0)
```

We now use the `**float()**` function to convert the `**net_amount**` value to a float before assigning it to the `**net_amount**` variable. If the `**net_amount**` field is `**None**` or doesn't exist in the JSON response, it will be treated as 0.

By converting the `**net_amount**` value to a float, we ensure that it's a numeric type, and we can safely add it to the `**total_credit**` or `**total_debit**` variables without encountering the `**TypeError**`.

If you want to treat the `**net_amount**` as an integer instead of a float, you can use the `**int()**` function instead:

```python
net_amount = int(float(result.get('net_amount') or 0))
```

This will first convert the value to a float, and then to an integer, effectively rounding down any decimal values.

With this change, the code should now work correctly and calculate the total credit, total debit, and their difference based on the `**net_amount_direction**` field within each `**chain_id**` group.

Share

Keep the float rounded to 2 decimal points

Answer

Got it, here's the updated code that rounds the `**net_amount**` to 2 decimal places:

```python
import json
from collections import defaultdict

# Assuming the API response is stored in the variable 'response'
data = response.json()

# Check if the response contains the 'results' key
if 'results' in data:
    results = data['results']

    # Filter by state == "filled"
    filled_results = [result for re
```

Try Pro
Upgrade for image upload, smarter AI, and more Pro Search.
Learn More

Download

Pro

perplexity

New Thread ⌘ K

Home

Discover

Library

Sign in

Sign Up

```
    # Group by chain_id
    grouped_results = defaultdict(list)
    for result in filled_results:
        chain_id = result.get('chain_id')
        grouped_results[chain_id].append(result)

    # Create a table for each chain_id
    for chain_id, chain_results in grouped_results.items():
        print(f"Chain ID: {chain_id}")
        print("-" * 20)
        print("{:<20} {:<20} {:<20} {:<20} {:<20}".format("Chain S
        print("-" * 100)

        # Sort the results within the group by reversed updated_at
        chain_results.sort(key=lambda x: x.get('updated_at'), reve

        total_credit = 0
        total_debit = 0

        for result in chain_results:
            chain_symbol = result.get('chain_symbol') or ''
            opening_strategy = result.get('opening_strategy') or '
            net_amount_direction = result.get('net_amount_directic
            net_amount = round(float(result.get('net_amount') or 0
            updated_at = result.get('updated_at') or ''

            if net_amount_direction.lower() == 'credit':
                total_credit += net_amount
            elif net_amount_direction.lower() == 'debit':
                total_debit += net_amount

            print("{:<20} {:<20} {:<20} {:<20} {:<20}".format(chai

        print("-" * 100)
        print(f"Total Credit: {total_credit:.2f}")
        print(f"Total Debit: {total_debit:.2f}")
        print(f"Difference: {total_credit - total_debit:.2f}")
        print("\n")

else:
    print("No 'results' key found in the response.")
```

The changes made are:

1. In the line where we retrieve the `**net_amount**` value, we use the `**round()**` function to round the float to 2 decimal places:

```python
net_amount = round(float(result.get('net_amount') or 0), 2)
```

2. When printing the `**total_credit**`, `**total_debit**`, and `**Difference**`, we use the `**.2f**` format specifier to limit the output to 2 decimal places:

```python
print(f"Total Credit: {total_credit:.2f}")
print(f"Total Debit: {total_debit:.2f}")
print(f"Difference: {total_credit -
```

Try Pro
Upgrade for image upload, smarter AI, and more Pro Search.
Learn More

Download

Pro

With these changes, the `**net_amount**` val
places, and the totals and difference will also be displayed with 2 decimal

perplexity

New Thread ⌘ K

Home

Discover

Library

Sign in

Sign Up

places.

The output should now look similar to this:

```
Chain ID: 1234
--------------------
Chain Symbol        Opening Strategy   Net Amount Direction Net Am
--------------------
ABCD                Buy                credit               1000.0
EFGH                Sell               debit                500.00
IJKL                Buy                credit               750.00
--------------------
Total Credit: 1750.00
Total Debit: 500.00
Difference: 1250.00

Chain ID: 5678
--------------------
Chain Symbol        Opening Strategy   Net Amount Direction Net Am
--------------------
MNOP                Sell               debit                1200.5
QRST                Buy                credit               800.25
--------------------
Total Credit: 800.25
Total Debit: 1200.50
Difference: -400.25
```

Share

Now sum up the values of the diff you calculated across all chained groups and print the result. Keep the result to 2 decimal points.

Answer

Here's the updated code that sums up the differences across all `**chain_id**` groups and prints the result rounded to 2 decimal places:

```python
import json
from collections import defaultdict

# Assuming the API response is stored in the variable 'response'
data = response.json()

# Check if the response contains the 'results' key
if 'results' in data:
    results = data['results']

    # Filter by state == "filled"
    filled_results = [result for res

    # Group by chain_id
    grouped_results = defaultdict(list)
    for result in filled_results:
```

Try Pro
Upgrade for image upload, smarter AI, and more Pro Search.
Learn More

Download

Pro

perplexity

New Thread ⌘ K

Home

Discover

Library

Sign in

Sign Up

```
        chain_id = result.get('chain_id')
        grouped_results[chain_id].append(result)

    total_diff = 0

    # Create a table for each chain_id
    for chain_id, chain_results in grouped_results.items():
        print(f"Chain ID: {chain_id}")
        print("-" * 20)
        print("{:<20} {:<20} {:<20} {:<20} {:<20}".format("Chain S
        print("-" * 100)

        # Sort the results within the group by reversed updated_at
        chain_results.sort(key=lambda x: x.get('updated_at'), reve

        total_credit = 0
        total_debit = 0

        for result in chain_results:
            chain_symbol = result.get('chain_symbol') or ''
            opening_strategy = result.get('opening_strategy') or '
            net_amount_direction = result.get('net_amount_directic
            net_amount = round(float(result.get('net_amount') or 0
            updated_at = result.get('updated_at') or ''

            if net_amount_direction.lower() == 'credit':
                total_credit += net_amount
            elif net_amount_direction.lower() == 'debit':
                total_debit += net_amount

            print("{:<20} {:<20} {:<20} {:<20} {:<20}".format(chai

        print("-" * 100)
        print(f"Total Credit: {total_credit:.2f}")
        print(f"Total Debit: {total_debit:.2f}")
        group_diff = total_credit - total_debit
        print(f"Difference: {group_diff:.2f}")
        total_diff += group_diff
        print("\n")

    print(f"Total Difference across all groups: {total_diff:.2f}")

else:
    print("No 'results' key found in the response.")
```

Here are the changes made:

1. We initialize a `total_diff` variable to 0 before iterating over the `chain_id` groups.
2. After calculating the difference (`group_diff`) for each `chain_id` group, we add it to the `total_diff` variable:

```python
total_diff += group_diff
```

3. After printing the results for all `chain_id` groups, we print the `total_diff` rounded to 2 decimal places:

```python
print(f"Total Difference across all
```

Try Pro
Upgrade for image upload, smarter AI, and more Pro Search.
Learn More

Download

Pro

With these changes, the code will now sum up the differences calculated for each `chain_id` group and print the total difference across all groups, rounded to 2 decimal places.

The output should look something like this:

```
Chain ID: 1234
--------------------
Chain Symbol        Opening Strategy   Net Amount Direction Net Am
--------------------
ABCD                Buy                credit               1000.0
EFGH                Sell               debit                500.00
IJKL                Buy                credit               750.00
--------------------
Total Credit: 1750.00
Total Debit: 500.00
Difference: 1250.00

Chain ID: 5678
--------------------
```

Anonymous 3 minutes ago Share

```
MNOP                Sell               debit                1200.5
QRST                Buy                credit               800.25
--------------------
Total Credit: 800.25
Total Debit: 1200.50
Difference: -400.25

Total Difference across all groups: 849.75
```

In this example, the total difference across all `chain_id` groups is `849.75`.

Share

The results array is part of a paginated response from Robinhood. The paginated response has the following items:
next, previous, results.

The `next` item contains the API url we can use to get the next response.

How do I repeat the above chained group calculations all the way until the `next` item is null?

Sources

REST API pagination loop and append... powerusers.... • 1

SSRS MS Report Builder: How to... sqlservercen... • 2

Nested repeat creating blank... community.e... • 3

View 2 more

Answer

To repeat the chained group calculations
can use a loop that fetches the next pag

Try Pro
Upgrade for image upload, smarter AI, and more Pro Search.
Learn More

Download

Pro

perplexity

New Thread ⌘ K

Home

Discover

Library

Sign in

Sign Up

the existing data. Here's how you can modify the code to achieve this:

```python
import json
from collections import defaultdict
import requests

# Assuming the initial API response is stored in the variable 'res
data = response.json()

all_results = []

# Check if the response contains the 'results' key
if 'results' in data:
    all_results.extend(data['results'])

    # Loop until the 'next' item is null
    while 'next' in data and data['next']:
        # Fetch the next page of results
        next_url = data['next']
        response = requests.get(next_url)
        data = response.json()

        # Append the new results to the existing data
        if 'results' in data:
            all_results.extend(data['results'])

    # Process the combined results
    filled_results = [result for result in all_results if result.g

    # Group by chain_id
    grouped_results = defaultdict(list)
    for result in filled_results:
        chain_id = result.get('chain_id')
        grouped_results[chain_id].append(result)

    # Create a table for each chain_id
    for chain_id, chain_results in grouped_results.items():
        print(f"Chain ID: {chain_id}")
        print("-" * 20)
        print("{:<20} {:<20} {:<20} {:<20} {:<20}".format("Chain S
        print("-" * 100)

        # Sort the results within the group by reversed updated_at
        chain_results.sort(key=lambda x: x.get('updated_at'), reve

        total_credit = 0
        total_debit = 0

        for result in chain_results:
            chain_symbol = result.get('chain_symbol') or ''
            opening_strategy = result.get('opening_strategy') or '
            net_amount_direction = result.get('net_amount_directic
            net_amount = round(float(result.get('net_amount') or 0
            updated_at = result.get('updated_at') or ''

            if net_amount_direction.lower() == 'credit':
                total_credit += net
            elif net_amount_direct
                total_debit += net_
```

Try Pro
Upgrade for image upload, smarter AI, and more Pro Search.

Learn More

Download

Pro

```
            print("{:<20} {:<20} {:<20} {:<20} {:<20}".format(chai

        print("-" * 100)
        print(f"Total Credit: {total_credit:.2f}")
        print(f"Total Debit: {total_debit:.2f}")
        print(f"Difference: {total_credit - total_debit:.2f}")
        print("\n")

else:
    print("No 'results' key found in the response.")
```

Here are the changes made to the previous code:

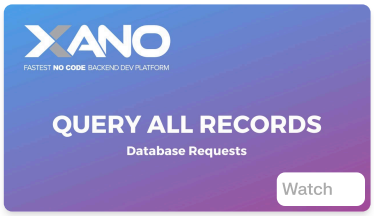
1. We import the `requests` library to make HTTP requests for fetching the next pages of results.
2. We initialize an empty list `all_results` to store all the results from the paginated response.
3. After processing the initial response, we extend the `all_results` list with the `results` from the initial response.
4. We use a `while` loop to fetch the next pages of results until the `next` item is `null` or doesn't exist.
5. Inside the loop, we fetch the next page of results using the URL from the `next` item and append the new `results` to the `all_results` list.
6. After the loop finishes, we process the combined `all_results` list, filtering for `state == "filled"`, grouping by `chain_id`, and performing the chained group calculations as before.

With these changes, the code will fetch all the pages of results from the paginated response and process them together, repeating the chained group calculations on the combined data.

Note: This code assumes that the `next` item contains the URL for the next page of results. If the structure of the paginated response is different, you may need to modify the code accordingly.

Share

In addition to filtering for state == "filled", also filter out any data where updated_at is before September 2023



Sources

Query All Records - Filter, Search, Sort,... youtube • 1

c# - Trying to pass filter variables insid... stackoverflow • 2

How to Use sorted() and .sort() in Pytho... realpython • 3

View 2 more

Answer

Here's the updated code that filters out any data where `updated_at` is before September 2023, in addition to filtering for `state == "filled"`:

```python
import json
from collections import defaultdic
from datetime import datetime
```

Try Pro
Upgrade for image upload, smarter AI, and more Pro Search.
Learn More

Download

Pro

perplexity

New Thread ⌘ K

Home

Discover

Library

Sign in

Sign Up

```
# Assuming the API response is stored in the variable 'response'
data = response.json()

# Check if the response contains the 'results' key
if 'results' in data:
    results = data['results']

    # Filter by state == "filled" and updated_at >= September 2023
    filled_results = [result for result in results if result.get('

    # Group by chain_id
    grouped_results = defaultdict(list)
    for result in filled_results:
        chain_id = result.get('chain_id')
        grouped_results[chain_id].append(result)

    total_diff = 0

    # Create a table for each chain_id
    for chain_id, chain_results in grouped_results.items():
        print(f"Chain ID: {chain_id}")
        print("-" * 20)
        print("{:<20} {:<20} {:<20} {:<20} {:<20}".format("Chain S
        print("-" * 100)

        # Sort the results within the group by reversed updated_at
        chain_results.sort(key=lambda x: x.get('updated_at'), reve

        total_credit = 0
        total_debit = 0

        for result in chain_results:
            chain_symbol = result.get('chain_symbol') or ''
            opening_strategy = result.get('opening_strategy') or '
            net_amount_direction = result.get('net_amount_directic
            net_amount = round(float(result.get('net_amount') or 0
            updated_at = result.get('updated_at') or ''

            if net_amount_direction.lower() == 'credit':
                total_credit += net_amount
            elif net_amount_direction.lower() == 'debit':
                total_debit += net_amount

            print("{:<20} {:<20} {:<20} {:<20} {:<20}".format(chai

        print("-" * 100)
        print(f"Total Credit: {total_credit:.2f}")
        print(f"Total Debit: {total_debit:.2f}")
        group_diff = total_credit - total_debit
        print(f"Difference: {group_diff:.2f}")
        total_diff += group_diff
        print("\n")

    print(f"Total Difference across all groups: {total_diff:.2f}")

else:
    print("No 'results' key found in the response.")
```

Here are the changes made to the previo

1. We import the `**datetime**` modul
   handle date and time operations.

Pro

Try Pro
Upgrade for image upload, smarter AI, and more Pro Search.

Learn More

Download

perplexity

New Thread ⌘ K

Home

Discover

Library

Sign in

Sign Up

2. In the list comprehension where we filter the `results`, we add an additional condition to check if the `updated_at` value is greater than or equal to September 2023:

```python
filled_results = [result for result in results if result.get('stat
```

This condition uses the `datetime.fromisoformat()` function to convert the `updated_at` string (in ISO format) to a `datetime` object. If the `updated_at` field is missing or `None`, we use a default value of `'2023-09-01'`.

We then compare the date part of the `updated_at` datetime object with `datetime(2023, 9, 1).date()`, which represents September 1, 2023. This condition ensures that only results with `updated_at` on or after September 1, 2023, are included in the `filled_results` list.

3. The rest of the code remains the same, performing the chained group calculations and printing the results as before.

With these changes, the code will now filter out any data where `updated_at` is before September 2023, in addition to filtering for `state == "filled"`. Only the results that meet both conditions will be included in the chained group calculations and the final output.

Note: This code assumes that the `updated_at` field in the JSON response is a string in ISO format (e.g., `"2023-09-15T12:34:56.789Z"`). If the format is different, you may need to modify the `datetime.fromisoformat()` call accordingly.

Share

ValueError: Invalid isoformat string: '2024-03-25T15:10:57.063284Z'

Answer

Ah, I see the issue. The `datetime.fromisoformat()` function expects the ISO format to be in the format `"YYYY-MM-DDTHH:MM:SS"`, but the `updated_at` field in your response has additional microsecond precision.

To handle this, we can use the `datetime.datetime.strptime()` function instead, which provides more flexibility in parsing the date and time string.

Here's the updated code:

```python
import json
from collections import defaultdict
from datetime import datetime

# Assuming the API response is stored in the variable 'response'
data = response.json()

# Check if the response contains the 'results' key
if 'results' in data:
    results = data['results']

    # Filter by state == "filled" a
    filled_results = [result for result in results if result.get('
```

Try Pro
Upgrade for image upload, smarter AI, and more Pro Search.
Learn More

Download

Pro

perplexity

New Thread ⌘ K

Home

Discover

Library

Sign in

Sign Up

```
    # Group by chain_id
    grouped_results = defaultdict(list)
    for result in filled_results:
        chain_id = result.get('chain_id')
        grouped_results[chain_id].append(result)

    total_diff = 0

    # Create a table for each chain_id
    for chain_id, chain_results in grouped_results.items():
        print(f"Chain ID: {chain_id}")
        print("-" * 20)
        print("{:<20} {:<20} {:<20} {:<20} {:<20}".format("Chain S
        print("-" * 100)

        # Sort the results within the group by reversed updated_at
        chain_results.sort(key=lambda x: x.get('updated_at'), reve

        total_credit = 0
        total_debit = 0

        for result in chain_results:
            chain_symbol = result.get('chain_symbol') or ''
            opening_strategy = result.get('opening_strategy') or '
            net_amount_direction = result.get('net_amount_directic
            net_amount = round(float(result.get('net_amount') or 0
            updated_at = result.get('updated_at') or ''

            if net_amount_direction.lower() == 'credit':
                total_credit += net_amount
            elif net_amount_direction.lower() == 'debit':
                total_debit += net_amount

            print("{:<20} {:<20} {:<20} {:<20} {:<20}".format(chai

        print("-" * 100)
        print(f"Total Credit: {total_credit:.2f}")
        print(f"Total Debit: {total_debit:.2f}")
        group_diff = total_credit - total_debit
        print(f"Difference: {group_diff:.2f}")
        total_diff += group_diff
        print("\n")

    print(f"Total Difference across all groups: {total_diff:.2f}")

else:
    print("No 'results' key found in the response.")
```

The changes made are:

1. In the list comprehension where we filter the `results`, we use `datetime.strptime()` instead of `datetime.fromisoformat()`:

```python
filled_results = [result for result in results if result.get('stat
```

The `datetime.strptime()` function takes two arguments: the date and time string, and the format string that describes the structure of the input string. In this case, the format string `'%'` the format of the `updated_at` field in y

If the `updated_at` field is missing or `None`, we use a default value of

Pro

Try Pro
Upgrade for image upload, smarter AI, and more Pro Search.

Learn More

Download

`'2023-09-01T00:00:00'` as the fallback.

The rest of the code remains the same.

With these changes, the code should now handle the `updated_at` field with microsecond precision and filter out any results where the `updated_at` is before September 2023.

Share

Try Pro

Upgrade for image upload, smarter AI, and more Pro Search.

Learn More

Pro

Download