

INTERACTIVE BROKERS PYTHON API

A Guide for Downloading Data,
Managing Accounts, and Trading



IB PYTHON API (IBAPI)



Account Info
- Balances & Positions



Market Data
- Historical & Real-Time



Orders & Execution
- Place, Modify, Cancel, Track

ENVIRONMENT SET-UP

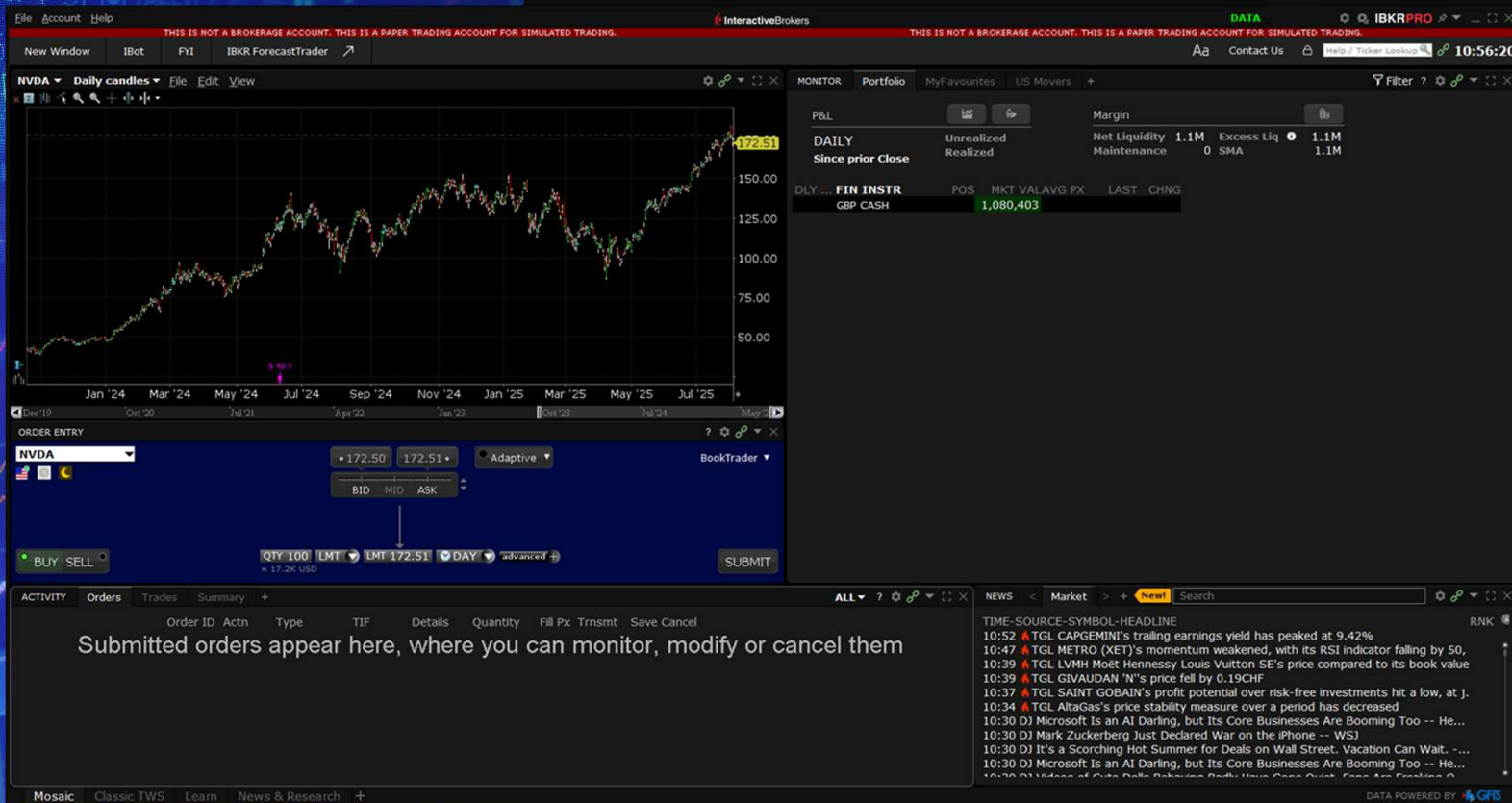
➤ IB Account

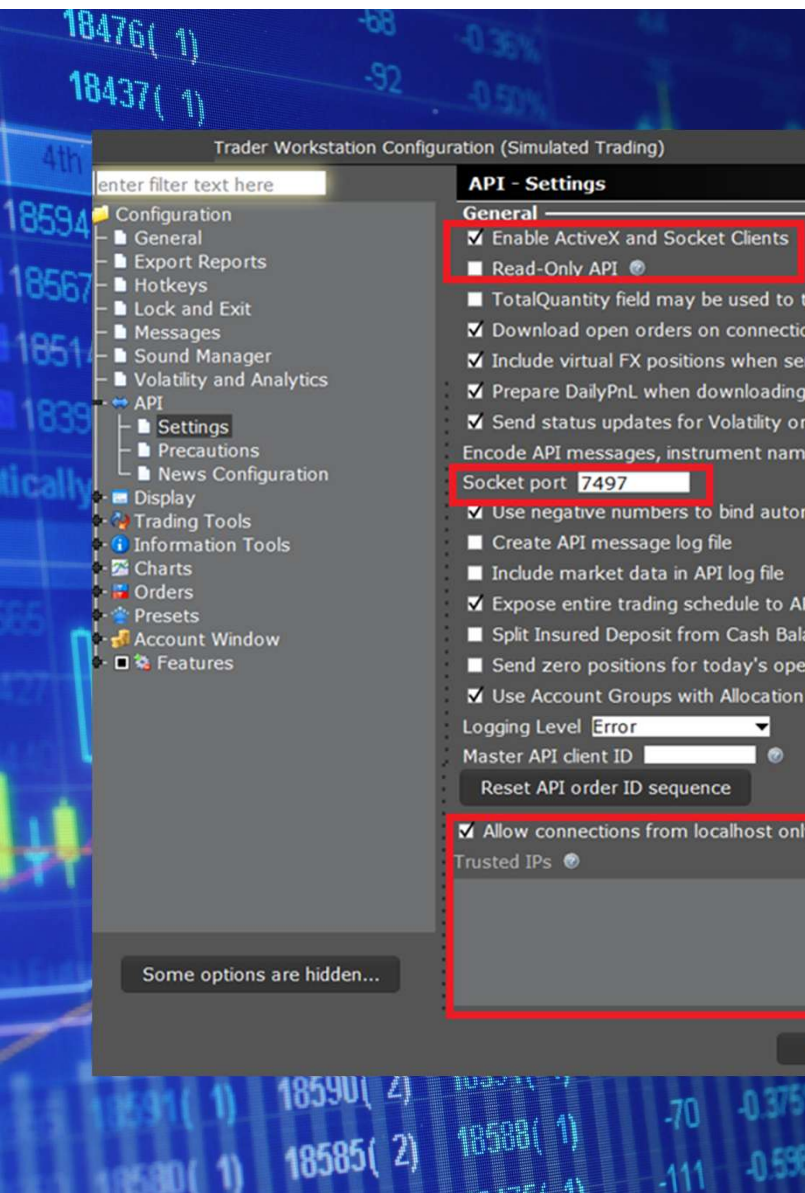
- ❖ Create IB account & enable API access
- ❖ Install Trader Workstation (TWS) or IB Gateway

➤ Python Environment

- ❖ Install Python 3.7+
- ❖ Install the IB API Python package: `pip install ibapi`
- ❖ Optionally, download latest API from: [interactivebrokers.github.io](https://github.com/interactivebrokers/ibapi)

TRADER WORKSTATION (TWS)





Trader Workstation Configuration (Simulated Trading)

enter filter text here

- Configuration
 - General
 - Export Reports
 - Hotkeys
 - Lock and Exit
 - Messages
 - Sound Manager
 - Volatility and Analytics
 - API
 - Settings**
 - Precautions
 - News Configuration
 - Display
 - Trading Tools
 - Information Tools
 - Charts
 - Orders
 - Presets
 - Account Window
 - Features

API - Settings

General

- ☒ Enable ActiveX and Socket Clients
- ☐ Read-Only API
- ☐ TotalQuantity field may be used to transmit monetary value for BUY orders for mutual fund
- ☒ Download open orders on connection.
- ☒ Include virtual FX positions when sending portfolio.
- ☒ Prepare DailyPnL when downloading positions.
- ☒ Send status updates for Volatility orders with "Continuous Update" flag.
- Encode API messages, instrument names: ASCII 7 (Python, Java, .NET...)
- Socket port: 7497
- ☒ Use negative numbers to bind automatic orders
- ☐ Create API message log file
- ☐ Include market data in API log file
- ☒ Expose entire trading schedule to API
- ☐ Split Insured Deposit from Cash Balance
- ☐ Send zero positions for today's opening positions only
- ☒ Use Account Groups with Allocation Methods.
- Logging Level: Error
- Master API client ID:
- Reset API order ID sequence
- ☒ Allow connections from localhost only
- Trusted IPs:

Create Edit Delete

Some options are hidden...

OK Apply Cancel

API SETTINGS

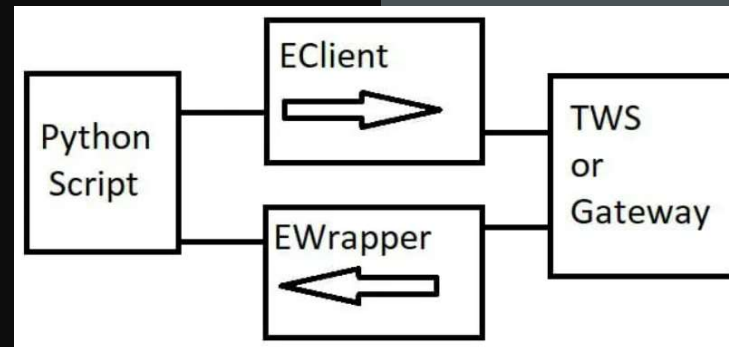
CONNECTION FRAMEWORK

- Connects through TWS or IB Gateway via TCP socket

Default Socket Ports

	Live	Demo
TWS	7496	7497
Gateway	4001	4002

- Combines sending requests (EClient) and receiving callbacks (EWrapper)



ASYNCHRONOUS EVENTS

- API is event-driven and asynchronous
- Requests are non-blocking; responses come via callbacks
- Common patterns:
 - ❖ Run API client in separate thread or event loop
 - ❖ Use synchronization methods (e.g., flags, events) to wait for data
 - ❖ Requires non-blocking programming mindset

TWS Connection Code

```
from ibapi.client import EClient
from ibapi.wrapper import EWrapper
import threading
import time
```

```
# Basic Code Structure
class IBAApp(EWrapper, EClient):
    def __init__(self):
        EClient.__init__(self, self)

    def nextValidId(self, orderId):
        time.sleep(1) # pause for connection
        print(f"Connected - Next Order ID: {orderId}")
```

```
# Connect
app = IBAApp()
app.connect('127.0.0.1', 7497, 123) # host, port, clientID

# Call app.run as a background thread
thread = threading.Thread(target=app.run)
thread.start()
```

Connection Response

ERROR means console output

```
ERROR -1 2104 Market data farm connection is OK:usfarm.nj
ERROR -1 2104 Market data farm connection is OK:eufarm
ERROR -1 2104 Market data farm connection is OK:cashfarm
ERROR -1 2104 Market data farm connection is OK:usfarm
ERROR -1 2106 HMDS data farm connection is OK:euhamds
ERROR -1 2106 HMDS data farm connection is OK:ushmds
ERROR -1 2158 Sec-def data farm connection is OK:secdefil
Connected - Next Order ID: 1
```

TWS
CONNECTION

Connections

?

↗

▼

✕

Market Data Connections

Farm Name	Purpose	Status
usfarm	Market Data	connected
ushmds	HMDS	connected
secdefil	Aux Services	connected
cashfarm	Market Data	connected
usfarm.nj	Market Data	connected
euhamds	HMDS	connected
fundfarm	HMDS	connected
cdc1.ibllc.com	Primary	connected

[More info](#)

API Connections (listening on *:7497)

Peer IP:port	Client ID	Status
127.0.0.1:56554	123	accepted

Reconnect All Farms

Redundant Backup Status

Site	Status
cdc1-hb1.ibllc.com	Accessible
cdc1-hb2.ibllc.com	Accessible

Last Login: Aug 02, 18:04

Close

TWS
CONNECTION

ACCOUNT SUMMARY & POSITIONS

- Use `reqAccountSummary` or `reqAccountUpdates` to get balances/cash
- Use `reqPositions()` to get held positions
- Important callbacks:
 - ❖ `accountSummary(...)` — for balances
 - ❖ `position(account, contract, position, avgCost)` — for positions
 - ❖ `positionEnd()` — signals completion

Account Summary & Positions Code

```
def accountSummary(self, reqId, account, tag, value, currency):
    print(f"Account Summary: Account: {account}, {tag} = {value} {currency}")
    self.account_values[tag] = (value, currency)

def accountSummaryEnd(self, reqId):
    # Avoid printing multiple times for the same reqId if triggered repeatedly
    if not self.account_summary_finished:
        print("Finished account summary request.")
        self.account_summary_finished = True
        self.try_disconnect()

def position(self, account, contract, position, avgCost):
    print(f"Position: {contract.symbol} | Qty: {position} | Avg Cost: {avgCost}")
    self.positions.append((account, contract, position, avgCost))

def positionEnd(self):
    if not self.positions_finished:
        print("Finished positions request.")
        self.positions_finished = True
        self.try_disconnect()
```

Account Summary Response

Simulation account has no positions and a cash balance

```
Finished positions request.
Account Summary: Account: [REDACTED], AvailableFunds = 1080402.51 GBP
Account Summary: Account: [REDACTED], BuyingPower = 4321610.04 GBP
Account Summary: Account: [REDACTED], NetLiquidation = 1083840.84 GBP
Account Summary: Account: [REDACTED], TotalCashValue = 1080402.51 GBP
Finished account summary request.
```

A/C SUMMARY
& POSITIONS

DOWNLOADING MARKET DATA

- Create a Contract object (specify symbol, exchange, secType, currency)
- Use reqHistoricalData method to request historical data
- Parameters to specify:
 - ❖ durationStr (e.g., '1 M' for 1 month)
 - ❖ barSizeSetting (e.g., '1 hour')
 - ❖ whatToShow ('TRADES' for trade data)

Market Data Requests

```
class IBAppl(EWrapper, EClient):
    def __init__(self):
        EWrapper.__init__(self)
        EClient.__init__(self, wrapper=self)
        self.data = []
        self.data_ready = threading.Event()
        self.connection_ready = threading.Event()

    def nextValidId(self, orderId):
        print(f"Connected (Order ID: {orderId})")
        self.connection_ready.set()

    # Fetch historical data
    def getData(self, contract, settings, req_id=1):
        if not self.isConnected():
            raise ConnectionError("Not connected to TWS or IB Gateway.")

        # Clear data and ready status
        self.data.clear()
        self.data_ready.clear()

        # Request data and wait for receipt
        self.reqHistoricalData(req_id, contract, **settings)
        self.data_ready.wait()

        # Create and Return Data in a Pandas DataFrame
        df = pd.DataFrame(self.data, columns=["Date", "Open", "High", "Low", "Close"])
        df["Date"] = pd.to_datetime(df["Date"])

        return df.set_index("Date")

    def historicalData(self, reqId, bar: BarData):
        self.data.append([bar.date, bar.open, bar.high, bar.low, bar.close])

    def historicalDataEnd(self, reqId, start, end):
        self.data_ready.set()
```

Request Market Data

MARKET DATA
REQUESTS

Market Data User Configuration

```
''' Demo Usage '''  
''' ----- '''
```

```
def run():  
    app = IBAApp()  
    app.connect('127.0.0.1', 7497, clientId=1)  
  
    # Start IB event loop in background  
    thread = threading.Thread(target=app.run, daemon=True)  
    thread.start()  
  
    app.connection_ready.wait()  
  
    ''' User Configuration '''  
    ''' ----- '''  
  
    symbol1 = "CSH2"  
    contract1 = app.createContract(symbol1, "STK", "LSE", "GBP")  
    settings1 = app.dataSettings("3 M", "1 day", whatToShow="TRADES")  
  
    df1 = app.getData(contract1, settings1)  
    app.displayData(df1)  
    app.plotData(df1, title=symbol1)  
  
    # Disconnect cleanly using thread.join() to wait for background thread to finish  
    app.disconnect()  
    thread.join()  
  
if __name__ == "__main__":  
    run()
```

Contract – CSH2 Money Mkt ETF

Daily Trade Data for 3 months

MARKET DATA
CONFIGURATION

Market Data Request

```
symbol1 = "CSH2"
contract1 = app.createContract(symbol1, "STK", "LSE", "GBP")
settings1 = app.dataSettings("3 M", "1 day", whatToShow="TRADES")

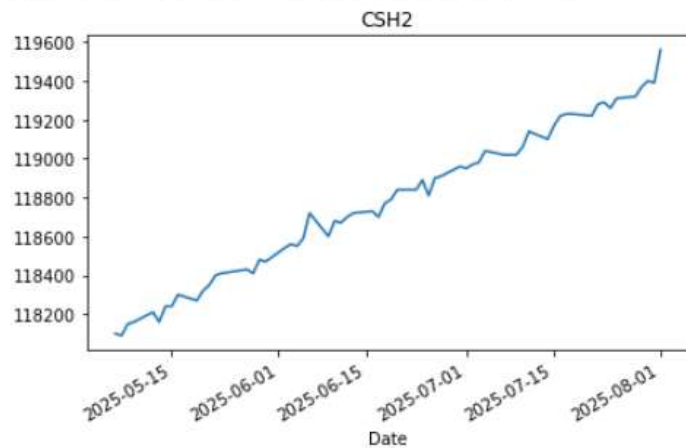
df1 = app.getData(contract1, settings1)
app.displayData(df1)
app.plotData(df1, title=symbol1)
```

Contract – CSH2 Money Mkt ETF

Daily Trade Data for 3 months

Market Data Results

Date	Open	High	Low	Close
2025-07-28	119310.0	119450.0	119310.0	119320.0
2025-07-29	119400.0	119400.0	119320.0	119370.0
2025-07-30	119380.0	119420.0	119350.0	119400.0
2025-07-31	119400.0	119440.0	119380.0	119390.0
2025-08-01	119420.0	119560.0	119380.0	119560.0



MARKET DATA
RESULTS

PLACING ORDERS

- Build an Order object specifying:
 - ❖ action (BUY/SELL)
 - ❖ orderType (e.g., MARKET, LMT)
 - ❖ totalQuantity
- Submit order via Python
 - ❖ `app.placeOrder(orderId, contract, order)`
- Track orders using callbacks:
 - ❖ `orderStatus(...)`
 - ❖ `openOrder(...)`
 - ❖ `execDetails(...)`

Placing Orders

```
# === Order Management ===
def createOrder(self, action="BUY", quantity=100, order_type="MKT", limit_price=None):
    order = Order()
    order.action = action
    order.orderType = order_type.upper()
    order.totalQuantity = quantity
    if order.orderType == "LMT" and limit_price:
        order.lmtPrice = limit_price
    # Send Order Immediately
    order.transmit = True
    # Ensure these legacy fields are NOT set
    order.eTradeOnly = False # <- you can omit this entirely in new APIs
    order.firmQuoteOnly = False
    return order

def placeOrderNow(self, contract, order):
    if self.nextOrderId is None:
        raise Exception("nextValidId not received yet.")
    order_id = self.nextOrderId
    self.nextOrderId += 1
    self.placeOrder(order_id, contract, order)
    print(f"New Order ID {order_id}: {order.action} {order.totalQuantity} {contract.symbol} ({order.orderType})")
    return order_id

# --- Place MKT Order ---
# =====
print(f"\n--- Place MKT Order ---")
order = app.createOrder(action="BUY", quantity=1, order_type="MKT")
order_id = app.placeOrderNow(contract, order)
```

TWS Order Screen

```
--- Place MKT Order ---
New Order ID 27: BUY 1 NVDA (MKT)
```

ACTIVITY										
Orders Trades Summary +										
	Order ID	Actn	Type	TIF	Details	Quantity	Fill Px	Trnsmt	Save	Cancel
NVDA	548443475	BUY	MKT	DAY		0/1	-			Cancel

PLACING
ORDERS

ORDER MANAGEMENT

MODIFY

- Modify by re-submitting order with **same orderId** and **new parameters**

CANCEL

- Cancel pending orders using `cancelOrder` method
- `app.cancelOrder(orderId)`

STATUS

- Get live order status using 'orderStatus' callback method
- Receive execution updates using 'execDetails' callback method

TWS Order Screen BEFORE

--- Place MKT Order ---
New Order ID 27: BUY 1 NVDA (MKT)

ACTIVITY Orders Trades Summary +											
	Order ID	Actn	Type	TIF	Details	Quantity	Fill Px	Trnsmt	Save	Cancel	
NVDA	548443475	BUY	MKT	DAY		0/1	-			Cancel	

Modify Order

```
# --- Modify MKT Order --- Use List Live Orders then update modify_order_id
# =====
print(f"\n--- Modify Order ---")
modify_order_id = 27
new_order = app.createOrder(action="BUY", quantity=5, order_type="MKT")
app.modifyOrder(modify_order_id, contract, new_order)
```

TWS Order Screen AFTER

--- Modify Order ---
Modified order 27

ACTIVITY Orders Trades Summary +											
	Order ID	Actn	Type	TIF	Details	Quantity	Fill Px	Trnsmt	Save	Cancel	
NVDA	548443475	BUY	MKT	DAY		0/5	0.00			Cancel	

MODIFY ORDER

TWS Order Screen BEFORE

--- Place MKT Order ---
New Order ID 27: BUY 1 NVDA (MKT)

ACTIVITY	Orders	Trades	Summary	+						
	Order ID	Actn	Type	TIF	Details	Quantity	Fill Px	Trnsmt	Save	Cancel
 NVDA	548443475	BUY	MKT	 DAY		0/1	-			

Cancel Order

```
# === Cancel Order --- Use List Live Orders then update cancel_order_id
# =====
print(f"\n--- Cancel Order ---")
cancel_order_id = 27
app.cancelOrderById(cancel_order_id)
```

TWS Order Screen AFTER

ACTIVITY	Orders	Trades	Summary	+																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																									

CANCEL ORDER

TWS Order Screen

--- Place MKT Order ---
New Order ID 27: BUY 1 NVDA (MKT)

ACTIVITY	Orders	Trades	Summary	+						
	Order ID	Actn	Type	TIF	Details	Quantity	Fill Px	Trnsmt	Save	Cancel
NVDA	548443475	BUY	MKT	DAY		0/1	-			Cancel

Order Status & Execution Details – Python Callback Code

```
=== Order Status Updates ===
def orderStatus(self, orderId, status, filled, remaining, avgFillPrice, permId, parentId,
    lastFillPrice, clientId, whyHeld, mktCapPrice):
    print(f"[OrderStatus] ID {orderId} | Status: {status} | "
        f"Filled: {filled}/{filled + remaining} | Avg Fill Price: {avgFillPrice}")

# === Executed Trade Details ===
def execDetails(self, reqId, contract, execution):
    print(f"[Execution] ID {execution.orderId} | Symbol: {contract.symbol} | "
        f"Side: {execution.side} | Price: {execution.price} | Qty: {execution.shares}")
```

Python Result

--- Place MKT Order ---
New Order ID 29: BUY 1 NVDA (MKT)

--- Disconnect ---

[OrderStatus] ID 29 | Status: PreSubmitted | Filled: 0.0/1.0 | Avg Fill Price: 0.0

ORDER STATUS

KEY METHODS & CALLBACKS

Task	Callback Method
Connect	connect(host, port, client_id)
Historical Data	reqHistoricalData(), historicalData()
Real-time Data	reqMktData(), tickPrice(), tickSize()
Account Summary	reqAccountSummary(), accountSummary()
Positions	reqPositions(), position(), positionEnd()
Place Orders	placeOrder(), orderStatus(), execDetails()
Modify Orders	placeOrder() with same orderId
Cancel Orders	cancelOrder()



RESOURCES

- **IB Official API Documentation**
<https://interactivebrokers.github.io/tws-api/index.html>
- **Algotrading101 IB Python API Guide**
<https://algotrading101.com/learn/interactive-brokers-python-api-native-guide/>
- **Interactive Brokers Campus - Python TWS API Course**
<https://www.interactivebrokers.com/campus/trading-course/python-tws-api/>



CLOSING REMARKS

➤ Tips

- ❖ First, try on a simulation account (demo account)
- ❖ Start small and do plenty of testing before placing real orders

➤ Subscribe for more info and resources

- ❖ Quant YouTube Channel: www.youtube.com/@algoquanthub
- ❖ Quant Newsletter: <https://algoquanthub.beehiiv.com/subscribe>

➤ Follow me on Linked-In www.linkedin.com/in/nburgessx