Algorithmic trading and backtest using Python

Dr. Hui Liu
IBridgePy@gmail.com
www.IBridgePy.com
San Jose, CA, United States

Learning goals

- Understand the steps to build algorithmic trading strategies.
- Know what can happen during live trading and how to handle the issues.
- Get familiar with techniques to backtest strategies

Contents

- Intro to setup a private fund
- Python choices to build strategies
- Intro of finding strategies using Machine learning
- Improvements to moving average crossover strategy
- Example of “from academic paper to algo trading strategy in Python”
- Options trading
- Backtesting skills

What other topics do you want to cover?

Steps to start hedge fund

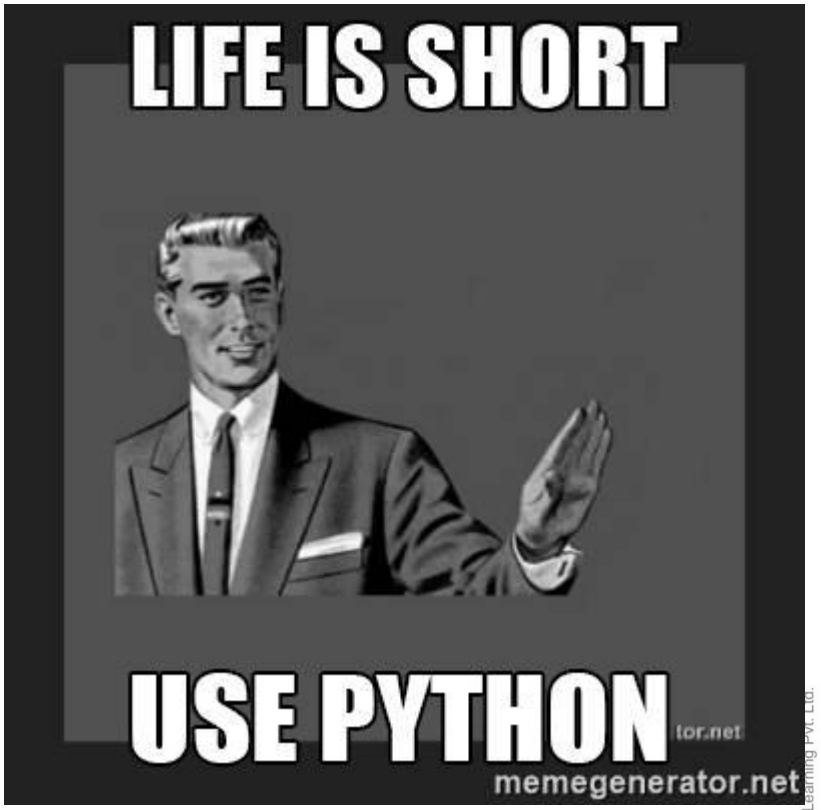
1. Research to find good strategies(fundamental analysis vs. quantitative analysis)
2. Translate strategies into python code if you don’t want to trade manually
3. Setup a server to run python code
4. Summary trading results
5. Improve performance
6. Show tracked records to investors
7. Set up investment accounts(advisor account vs pooled account)
8. Manage client portfolio

Major components of algo trading desk

1. Successful strategies
2. Python codes translated from the strategies
3. A computer(local or cloud)
4. A python platform/environment to backtest and live trade(backward testing vs forward testing)
5. Internet
6. Broker accounts
7. Customers

Why Python?

- Easy to learn
- Availability of variety of packages(pip install), especially AI, machine learning packages)
- Open source



Python choices For Algo Trading

	Example	Pro(advantages)	Con(disadvantages)
Web-based trading platforms	Quantopian, Backtrader	• Huge data • No installation • Backtest	• NO PRIVACY • Coding limitations • Limited trading products
Low-level python wrappers	IBpy, IB_insync	• Live trade • Basic functions	• So hard to learn • No backtest • No support
Backtesting engines	Zipline	• Backtest features • Great community	• No live trading • Limited trading products
Brokers' native API	IB, TD Ameritrade	• Live trade • Some supports	• So hard to learn • No backtest • Long time to code
Self-hosted solutions	IBridgePy	• PRIVACY • Easy to use • Backtest and live	• Backtest needs data from providers • a dedicated server

IBridgePy

www.IBridgePy.com

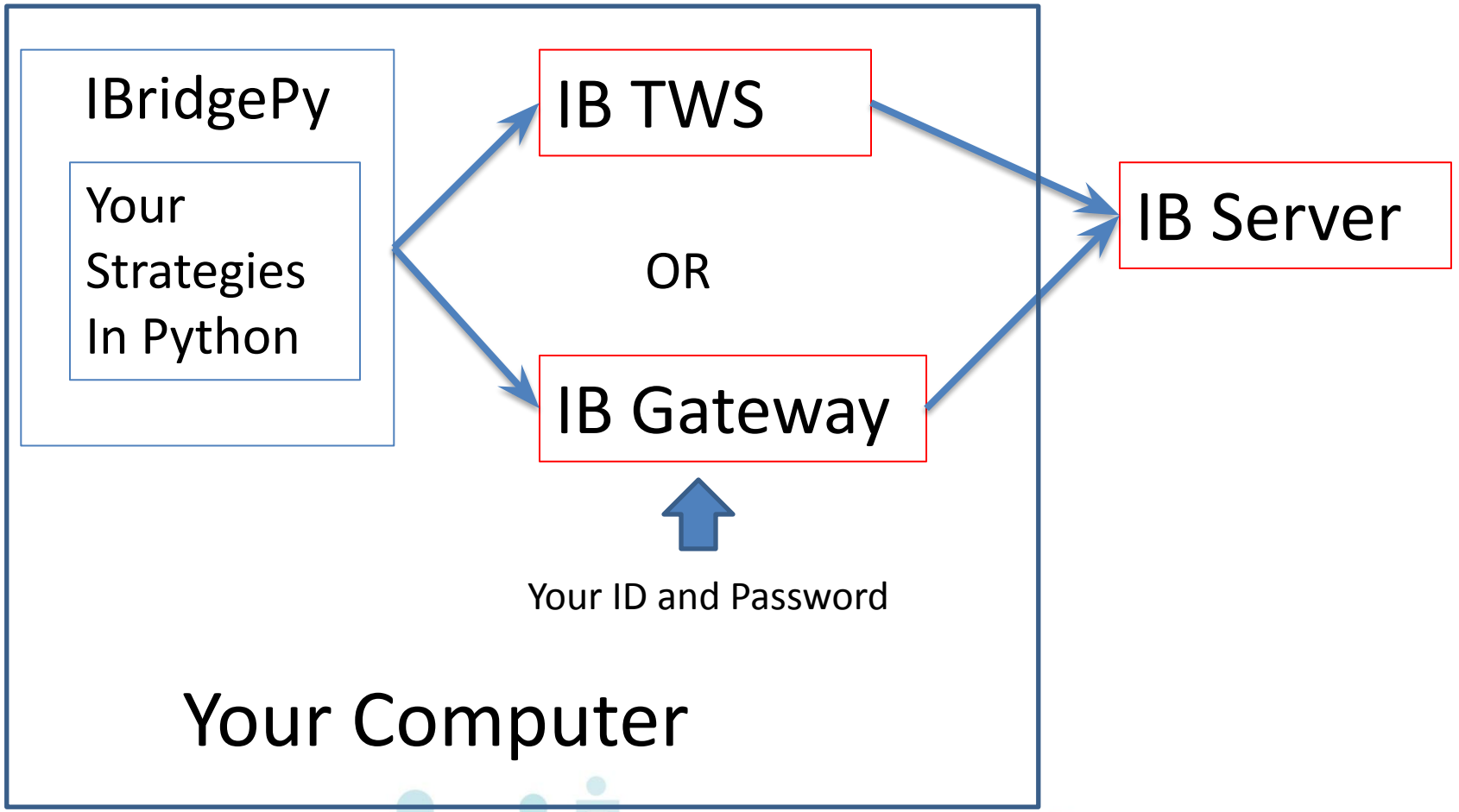
- **Algorithmic backtest and live trading platform in Python**
  - Privacy
  - Flexible and easy to use
- **Main features:**
  - Trade any securities or commodities offered by IB, TD Ameritrade and Robinhood.
  - Manage multiple accounts at same time
  - Execute multiple trading strategies at same time
  - Run Quantopian-style algorithms

IBridgePy

www.IBridgePy.com

- Introduce YouTube channel:
  https://www.youtube.com/channel/UCOSvCJJG7PTH-jhp0phvlrg
- IBridgePy API doc
  https://ibridgepy.com/documentation/
- Community

IBridgePy protects privacy



Brief intro to IBridgePy

Basics code structure.

- initialize
- handle_data

Three cornerstones:

- Real time price
- Historical data
- Place order

Every one should have completed the prerequisite of IBridgePy training in Quatra. If not, please complete before the next session.

From Idea to Trade robot

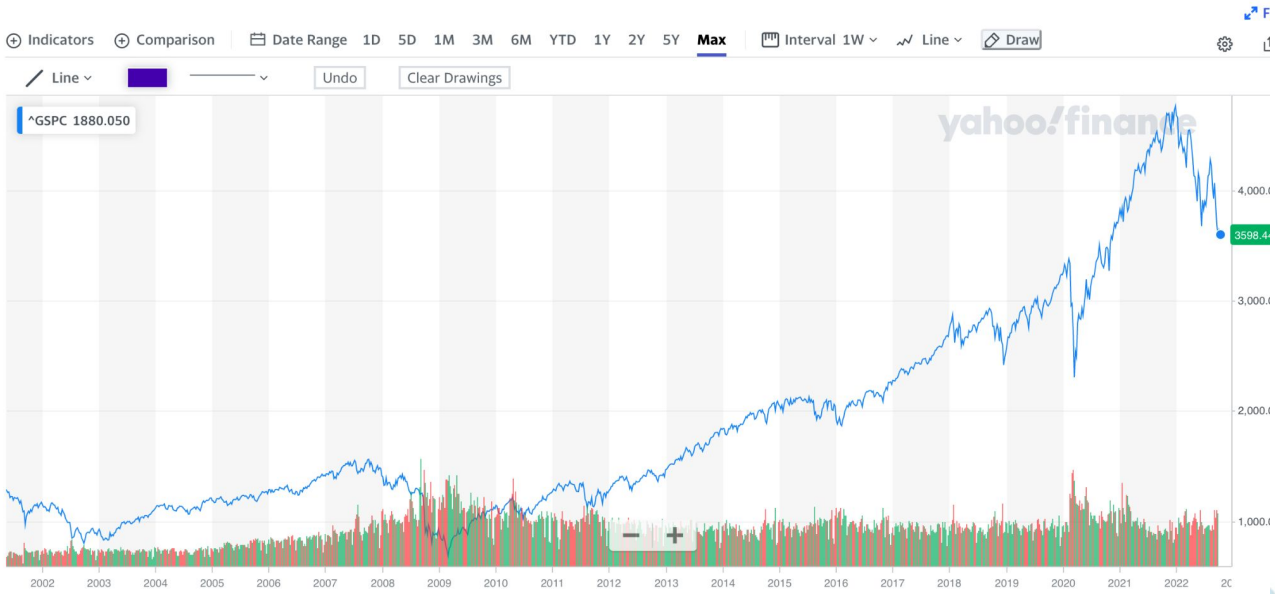
1. Come up a trading idea
2. Collect related data to the trading idea
3. Build a mathematical model to test
4. Draw graph to help visualization
5. Program in IBridgePy
6. Backtest the strategy
7. Forward-test the strategy
8. Live trade with small portfolio
9. Live trade with full portfolio

7 - minute Break

Trend or Swing?

Let’s vote to see which you believe

Swing vs Trend



Trend or Swing?

Buy Low Sell High

1. Come up an idea - If price drops compared to yesterday, buy in. Otherwise, sell off. Trade daily.
2. Collect related data - SPY as an example.
3. Build a mathematical model to test - linear regression
4. Draw graph to help visualization - matplotlib
5. Program in IBridgePy - a strategy py file
6. Backtest the strategy

Buy Low Sell High

In class:

$ jupyter notebook

Open EPAT_buyLowSellHighModel.ipynb

7 - minute Break

Buy Low Sell High

In class:

$ jupyter notebook

Open EPAT_buyLowSellHighModel-2.ipynb

"if you torture the data long enough, it will confess to anything"

- By Ronald H. Coase, a British Economist

See you in next session!

Steps to build algo trading program

- What contracts do you want to trade?
- When to make trading decision? Frequency?
- What info are needed to make trading decisions?
- What strategy to place orders?
- When to Stop loss and take profits?
- When to exit the positions?
- Continuously vs run-as-needed?

Buy Low Sell High

If today's close price is lower than yesterday's close price, buy SPY using all cash. Otherwise, sell off all positions.

- Manually input contract
- Reversion strategy
- Daily
- Need historical data
- Trading decision made at spot time = 3:59 PM Eastern time
- Market order for instant execution

Buy Low Sell High

In class:

explain implementation of buy-low-sell-high

EPAT_demo_buy_low_sell_high.py

Moving Average Crossover

```python
import pandas as pd
def initialize(context):
    context.run_once=False # To show if the handle_data has been run in a day
    context.security=symbol('SPY') # Define a security for the following part

def handle_data(context, data):
    sTime=get_datetime()
    # sTime is the IB server time.
    # get_datetime() is the build-in fuciton to obtain IB server time
    if sTime.weekday()<=4:
        # Only trade from Mondays to Fridays
        if sTime.hour==15 and sTime.minute==58 and context.run_once==True:
            # 2 minutes before the market closes, reset the flag
            # get ready to trade
            context.run_once=False
        if sTime.hour==15 and sTime.minute==59 and context.run_once==False:
            # 1 minute before the market closes, do moving average calcualtion
            # if MA(5) > MA(15), then BUY the security if there is no order
            # Keep the long positions if there is a long position
            # if MA(5) < MA(15), clear the position
            hist = data.history(context.security, 'close', 20, '1d')
            mv_5=pd.rolling_mean(hist,5)[-1]
            mv_15=pd.rolling_mean(hist,15)[-1]
            if mv_5>mv_15:
                print(sTime)
                orderId=order_target_percent(context.security, 1.0)
                order_status_monitor(orderId, target_status='Filled')
            else:
                orderId=order_target_percent(context.security, -1.0)
                order_status_monitor(orderId, target_status='Filled')
            context.run_once=True
```

Everyone should have learned the code already

Ready to trade just before the market closes

Request historical data

Place order if MAs cross

Calculate moving average

Handle errors in Live trading

- What errors should you expect in live trading?

Handle errors in Live trading

Many errors can happen in live trading

1. User has no permission to access data -- Call broker's rep.
2. The market is not open -- the real time price is not fresh
3. Request historical data too often -- minimize requests
4. No enough margin to place orders -- try - except
5. Order status is unknown -- order_status_monitor
6. etc.

Look for solutions:

1. IBridgePy YouTube channel
2. https://ibridgepy.com/tutorials/
3. https://ibridgepy.com/documentation/
4. https://ibridgepy.com/blog/

Moving Average Crossover

In class discussion:

What are the potential issues that the code has during LIVE trading?

What features can be added/improved for LIVE trading?

Improvement to Moving Average Crossover strategy

- Improvement 1: Handle holidays
- Improvement 2: handle market early close
- Improvement 3: Send out email notification
- Improvement 4: Handle market exceptions during live trading
- Expansion: change it to a minute-bar strategy

Improvement to Moving Average Crossover strategy

- Improvement 1: Handle holidays

```
# https://en.wikipedia.org/wiki/Moving_average_crossover
def handle_data(context, data):

    # If today is not a trading day, just do not continue
    if not isTradingDay():
        return
```

Improvement to Moving Average Crossover strategy

- Improvement 2: Handle market early close

https://www.nyse.com/markets/hours-calendars

In class explain the changes to handle early close.

moving_average_crossover_early_close_EPAT.py

7 - minute Break

Improvement to Moving Average Crossover strategy

- Improvement 3: Send out email notification
- Configure steps in YouTube tutorial

https://youtu.be/jkeos2QrkfQ

In class code practice:

I want to know my account balance after the market is closed everyday.

Answers is here:

moving_average_crossover_email_EPAT.py

Improvement to Moving Average Crossover strategy

- Improvement 4: Handle market exceptions during LIVE trading

In class code practice:

If any error happens when placing order, just ignore the error, send an email notification and keep the code running.

Answers is here:

moving_average_crossover_handle_error_EPAT.py

Improvement to Moving Average Crossover strategy

Feature expansion: change it to a minute-bar strategy

"I want to look for moving average crossover opportunity every minute instead of every day"

Answers is here:

Explain IBridgePy's setting system:

EPAT/settings_demo.py

moving_average_crossover_minute_bar_EPAT.py

VIX predict returns

The relationships between implied volatility indices and stock index returns

The investor uses the past 20 year history of the VIX index to create 20 equally spaced percentiles. He/she then goes long on the equity index at the close for one day if the VIX index on the current day is higher than on any other day during the rolling 2 year window or if the VIX index value is in the highest 2 percentile boxes. He/she goes short on the equity index at the close for one day if the VIX index on the current day is lower than on any other day during the rolling 2 year window or if the VIX index value is in the lowest 2 percentile boxes.

http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=5ABA12147DEB71866EF16F4D51343C90?doi=10.1.1.199.4665&rep=rep1&type=pdf

7 - minute Break

Implement VIX predict returns

- What contracts do you want to trade?
- When to make trading decision? Frequency?
- What info are needed to make trading decisions?
- What strategy to place orders?
- When to Stop loss and take profits?
- When to exit
- Continuously vs run-as-needed?

Sample code:

example_VIX_predicts_returns.py

Premium features

- Get_contract_details: option chains
- Get_scanner_results : stock scanner
- Get_option_greeks: trading options

hash.conf for this EPAT(3 months from now)

NANANA

Contact with EPAT assistant for passcode extension if using for EPAT projects.

Trade most active stocks

Goal:

Hold 10 most active stocks, each for 10% of portfolio, assuming that the account does not have any positions.

Sample code:

EPAT_screener_buy_most_active.py

Trading Options

- Search Options in IB security database
- Search Options using IBridgePy
- Get greeks of Options

Futures, Options, etc.

- Symbol : NIFTY17NOV10300PE
- Name : NIFTY50
- IPOyear :
- Sector : INDEX
- industry :
- primaryExchange: NSE
- secType : OPT
- Currency : INR
- exchange : NSE
- priceIncrement: 0.05

How to find the correct info of a contract?

IB security search website
https://www.interactivebrokers.com/en/index.php?f=463

```
context.sec=superSymbol(secType='OPT',symbol='NIFTY50',currency='INR',
                        exchange='NSE', primaryExchange='NSE', right='P',
                        expiry='201711', strike=10300.0)
```

Search Options to trade

Goal:

1. Filter the options which has an expiry closest to a target datetime.
2. Filter the options which has a strike price closest to a target strike price.

Sample code: EPAT_get_option_chains.py

Get_option_greeks

Goal:

Get greeks of Options for trading decisions.

Sample code: example_get_option_info.py

High Frequency Trading

A: High frequency trading is an automated trading platform used by large investment banks, hedge funds and institutional investors which utilizes powerful computers to transact a large number of orders at extremely high speeds. These high frequency trading platforms allow traders to execute millions of orders and scan multiple markets and exchanges in a matter of seconds, thus giving the institutions that use the platforms a huge advantage in the open market.

What are the considerations when implementing HFT?

- Internet Latency
- High-performance CPU
- Enough memory
- Fast algorithm
- Stable tick-by-tick data
- Low transaction cost

Handle Multiple Accounts

- **IBridgePy is able to handle multiple accounts**

  A very useful feature for fund managers

In the following example, a signal triggers BUY 100 shares in account 1 and BUY 500 shares in account 2

```
if mv_5>mv_15:
    orderId1=order_target(context.security, 100, ACCOUNT1)
    orderId2=order_target(context.security, 500, ACCOUNT2)
    order_status_monitor(orderId1, target_status='Filled')
    order_status_monitor(orderId2, target_status='Filled')
else:
    orderId1=order_target(context.security, 0, ACCOUNT1)
    orderId2=order_target(context.security, 0, ACCOUNT2)
    order_status_monitor(orderId1, target_status='Filled')
    order_status_monitor(orderId1, target_status='Filled')
context.run_once=True
```

Deployment

on a local computer	computer in cloud
Lower cost	Monthly charge
Self managed utilities (power and internet)	99.99% uptime
Local development, quick try	Push code to remote
Large disk storage	Small storage(cost)
Full resources	Shared computing(cost)
Full privacy	Full privacy
Local Operating system	Limited choices

Debug Codes

- Easiest error: syntax error.

```
15 def initialize(context):
16     #context.security=symbol('CASH,EUR,USD')
17     context.security=symbol('SPY')
18
19 def handle_data(context, data):
20     cTime= get_datetime()
21     print cTime.hour, cTime.minute
22     print show_real_time_price(context.security,'ask_price')
23     tp=data[context.security].ask_price_flow
24     print tp
25
```

Spyder editor will help you on syntax errors but they may be false alarms.
The checker complains that functions are used before the declaration.
Is it true?

Backtesting using historical data from IB

1. Open TEST_ME.py - Entrance of IBridgePy backtesting system
2. Change accountCode
3. Pick startTime and endTime of backtesting

In class demo: EPAT_TEST_ME_easiest.ipynb

Pros: Easy to backtest

Cons: Ask historical data from Brokers every time

Backtesting using historical data from IB

Declare histIngectionPlan, an IBridgePy reserved word

a. If handle_data is used, must ingest minute data at least
b. If schedule_function and/or request_historical_data is used, timeframe of requested historical data should adjust accordingly
c. Hist data of all contracts in the code should be declared
d. IBridgePy will fetch hist data from IB server following histInjectionPlan

Backtest result analysis: two log files (BalanceLog, TransactionLog) are saved in Output folder. tools/portfolio_sharpe_ratio.py

In class demo: EPAT_TEST_ME_ingest_hist.ipynb

Pros: Ingest hist at the beginning, save time

Cons: Broker may not have the data needed(for example, minute bars)

Backtesting using any historical data

Load local files to backtest.

In class demo: EPAT_TEST_ME_localHistFile.ipynb

Pros: Use data from any data providers.

Cons: Many data are not used.

Speed up Backtesting

Designating spot times to backtest.

In class demo: EPAT_TEST_ME_spot_times.ipynb

Pros: Backtest is much faster

Cons: What if minute bars are not available?

Backtesting without enough data

Ues the close price of daily bars to simulate minute data.

In class demo: EPAT_TEST_ME_simulated_by_daily_bars.ipynb

Pros: Backtest without enough data

Cons: Accuracy is compromised.

Summary

- IBridgePy can help you:
  - Trade any securities/commodities that IB offers
  - Handle multiple accounts at same time
  - Use any python packages
  - Handle tick data
  - Has a debug tool
  - Can be used to backtest strategies

IBridgePy is Flexible and Easy-to-use

Appendix

Code structure

```
def initialize(context):
    pass

def handle_data(context, data):
    print get_datetime().strftime("%Y-%m-%d %H:%M:%S %Z")
    print "ask_price=",show_real_time_price(symbol('CASH,EUR,USD'),'ask_price')
```

- "initialize()" is an built-in method to claim variables. It will only be run once
- "handle_data()" is also an built-in method where trading decisions are made. Two inputs are given here (context, data). "Context" contains the variables claimed in initialize(). "data" contains account information and near real-time quotes received from IB

Ask for real-time quotes

- When you need a real time price, simply call a build-in function of show_real_time_price

```
19      print show_real_time_price(symbol('CASH,EUR,USD'),'ask_price')
20
```

- For most of US stocks and ETF, use “symbol( ) “
- For other security types, use “superSymbol( ) ”

Live market demo YouTube video:
https://youtu.be/x8mGFSq0jJo

Ask for historical data

- Function: **request_historical_data**(security, barSize, goBack)
- Example: request_historical_data(symbol('SPY'), '1 day', '5 D')

	close	high	low	open	volume
2016-09-15	215.28	215.73	212.75	212.96	975343
2016-09-16	213.37	213.69	212.57	213.48	914867
2016-09-19	213.41	214.88	213.03	214.13	608436
2016-09-20	213.42	214.59	213.38	214.41	466162
2016-09-21	215.82	216.03	213.44	214.24	939522

- Request_historical_data https://youtu.be/7jmHRVsRcI0
- Save historical data to local files while gracefully handling errors https://youtu.be/Vuf7Jb9-hCk
- Request historical data of other than U.S. equities and handle "No data permission" and "No security definition" https://youtu.be/iiRierq6sTU
- Download historical data convert to epoch index https://youtu.be/hYL6SYgy7wE
- Download historical data and make a machine learning https://youtu.be/hNCwNxeXrwA
- Request historical data of FOREX, provided by Interactive Brokers for free https://youtu.be/7jmHRVsRcI0
- Handle errors related to request hist https://youtu.be/Mq9SUX-iwS4

Place orders

- Place market orders: **order**(symbol('SPY'), 100)
  - Place a market order
  - The target security is SPY
  - The action is to BUY 100 shares when n > 0
  - Negative number means SELL, -100 = SELL 100 shares

- Place Limit/Stop orders
  - **order**(symbol('SPY'), 100, LimitOrder(213.42)) place a limit order to BUY 100 shares of SPY at price = $213.42 per share*
  - **order**(symbol('SPY'), -100, StopOrder(213.42)) place a stop order to SELL 100 shares of SPY at price = $213.42 per share*

https://youtu.be/xAHb6Pl2PiI --> Place order

https://youtu.be/6zCTCYT5JAI --> Close all positions

Smart Routing

- Interactive Brokers offers its customers a software product referred to as "Smart" order routing. Smart Routing software continually scans competing markets and automatically routes orders directly to the best ECN or market center -- based on **price** but also taking into account factors such as the **availability of automatic order execution**.

  Pros-?

  Cons-?

Specify exchange if needed