



Trading Algoritmico con Python

Robotrader 2018

Javier Falces Marin
javifalces@gmail.com





Trading Algoritmico

El trading algorítmico, o trading basado en reglas y procesos, es una modalidad de operación en mercados financieros (trading) que se caracteriza por el uso de algoritmos, reglas y procedimientos automatizados en diferentes grados, para ejecutar operaciones de compra o venta de instrumentos financieros. Es común relacionarlo con el trading automatizado toda vez que los procesadores y ordenadores utilizan algoritmos para sus operaciones, por cuanto existe un alto grado de cercanía entre ambos

¿Porque Python?

- Pandas
- Numpy
- Scikit
- Comunidad
- APIs con librerías
 - [Quandl](#)
 - [Alphavantage](#)



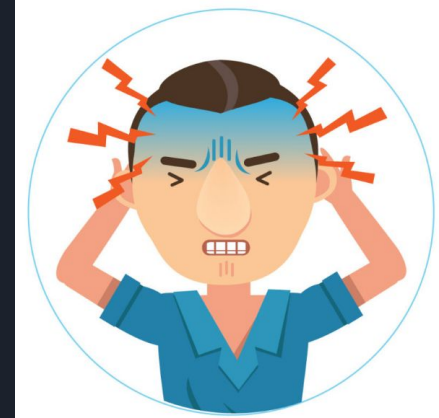


Arquitectura de framework de trading

1. Estructura de estrategias (clase strategy)
2. Estructura de conector a broker (clase broker)
3. Estructura de data source
4. Motor de backtesting
5. Motor de ejecucion

Desventajas de creación framework

- Tener que implementar motor de backtest
- Tener que implementar conectores a mercado
 - Aumentan posibilidades de error
 - Más tiempo de desarrollo hasta que desarrollas la primera estrategia
- Más posible que nuestro framework sea mas lento que uno comercial
 - Dependerá de lo que trabajemos sobre nuestro framework, podríamos mejorarlo también
- ESFUERZO





Ventajas de creación framework

- Diversificar , aparte de aprender trading algorítmico aprendes a programar en un lenguaje de programación que se usa en la empresa.
- Uso de librerías open source directamente sobre tu framework
- Posibilidad de adecuarlo a tu tipo de estrategia
- Posibilidad de cambiar de broker sin tener que migrar los algoritmos
 - Solo estarías atado a los brokers con api
- Posibilidad de hacer tipos especiales de backtesting
 - Forward Analysis
 - Cross Validation
 - Optimización de parámetros
- Posibilidad de incorporar gestión de portfolios de estrategias
- No tienes que encontrar un framework adecuado a ti
- FLEXIBILIDAD

Tipos de Frameworks



Event Driven

```
>>> data = {
>>>     "2017-02-01": 10.07,
>>>     "2017-02-02": 9.87,
>>>     "2017-02-03": 9.91,
>>>     "2017-02-04": 10.01
>>> }
>>> for date, price in data.items():
>>>     if price < 10:
>>>         buy_signal = True
>>>     else:
>>>         buy_signal = False
>>>     print(date, buy_signal)
2017-02-01 False
2017-02-02 True
2017-02-03 True
2017-02-04 False
```

Vectorized

```
>>> import pandas as pd
>>> data = {
>>>     "2017-02-01": 10.07,
>>>     "2017-02-02": 9.87,
>>>     "2017-02-03": 9.91,
>>>     "2017-02-04": 10.01
>>> }
>>> prices = pd.Series(data)
>>> buy_signals = prices < 10
>>> buy_signals.head()
2017-02-01    False
2017-02-02     True
2017-02-03     True
2017-02-04    False
dtype: bool
```



Event Driven



- Programacion mas sencilla
- Backtest y ejecucion real muy similares, solo cambia la fuente de datos
- Posibilidad de hacer referencias a eventos pasados mas sencillos

- Backtest mucho mas lento
- Menos optimizaciones por tiempo
- No exprime pandas



Vectorized




- Backtesting mucho mas rapidos
 - Más optimizaciones por tiempo
 - Exprime pandas al máximo
-
- Programación más compleja, hay que pensar bien como no hacer un cuello de botella
 - Aumentan posibilidad de errores Look-ahead bias
 - Backtest y ejecución real tendran un flujo diferente
 - En real no vamos a tener la matriz de precios completa



Jupyter -Notebook





Frameworks de trading algoritmico comunidad

1. [Backtrader](#)(ED)
2. [Zipline](#)(ED) + [ZiplineLive](#)
3. [PySystemTrade](#)(ED)
4. [FinMarketPy](#)(ED)
5. [QtPyLib](#)(ED)
6. [PyAlgoTrade](#)(ED)
7. [QuantRocket](#)(Vectorized)[\$\$]

[+Links](#)



Backtrader

- Conexion a IB
- Buena comunidad
- Backtest rapido
- Realizado por un español
- Python 2.7 y 3.x
- No usa pandas
- No se puede usar scikit



Zipline

- Enorme comunidad de quantopian
- Compatible con scikit, sklearn y cualquier cosa
- Con zipline live puedes usar los algos en IB
- Python 2.7 y 3.x
- Fuente de datos de backtesting por defecto son quandl y yahoo(muerta)
- Instalacion y backtesting de algoritmos mezclando python + cmd



QtPyLib

- Monta un sistema de notificaciones
- Sistema de monitorizacion de los algoritmos en real muy interesante
- Compatible con scikit, sklearn y cualquier cosa
- Puedes usar los algos en IB
- Blotter guarda datos tick
- Python 3.x
- Fuente de datos de backtesting por defecto es IB
- Requiere instalacion de base de datos, complejo
- No funciona en vela diaria

Instalacion[[link](#)]



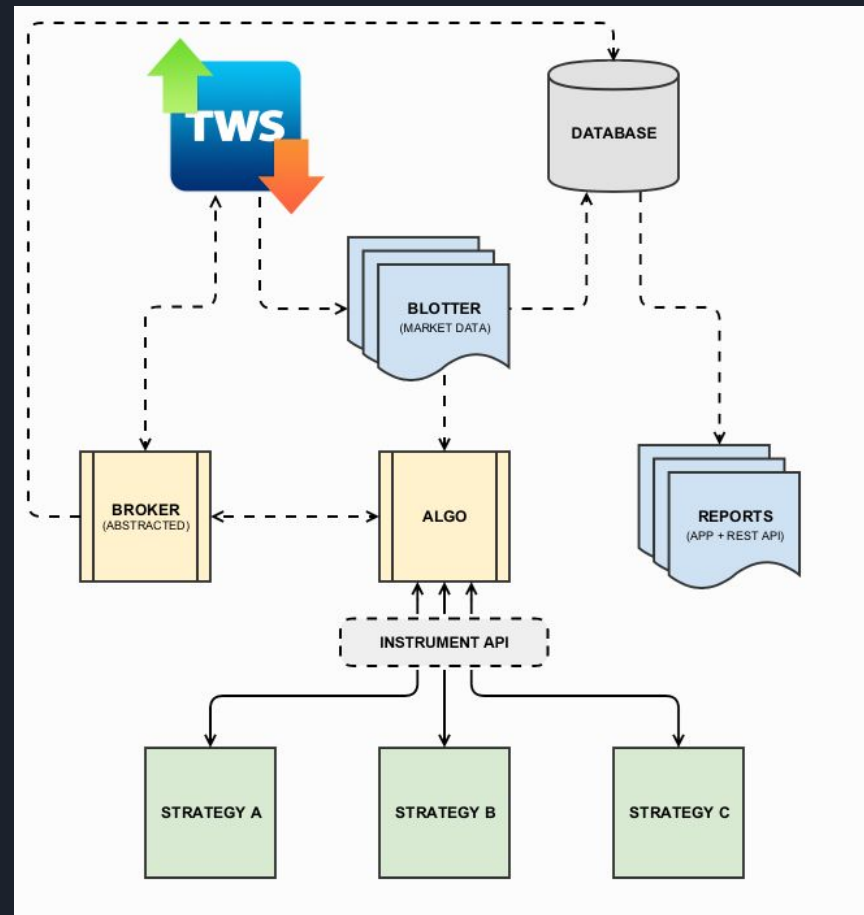
1. [Instalar Anaconda3.6](#)
2. Instalar Requirements de Zipline
 - a. `$ pip install qtpylib --upgrade --no-cache-dir`
 - b. Install rest of requirements
3. Instalar MySQL
 - a. `mysql -u root -p -e "create database qtpy;"`

[Download TWS \(Traders Workstation\)](#)

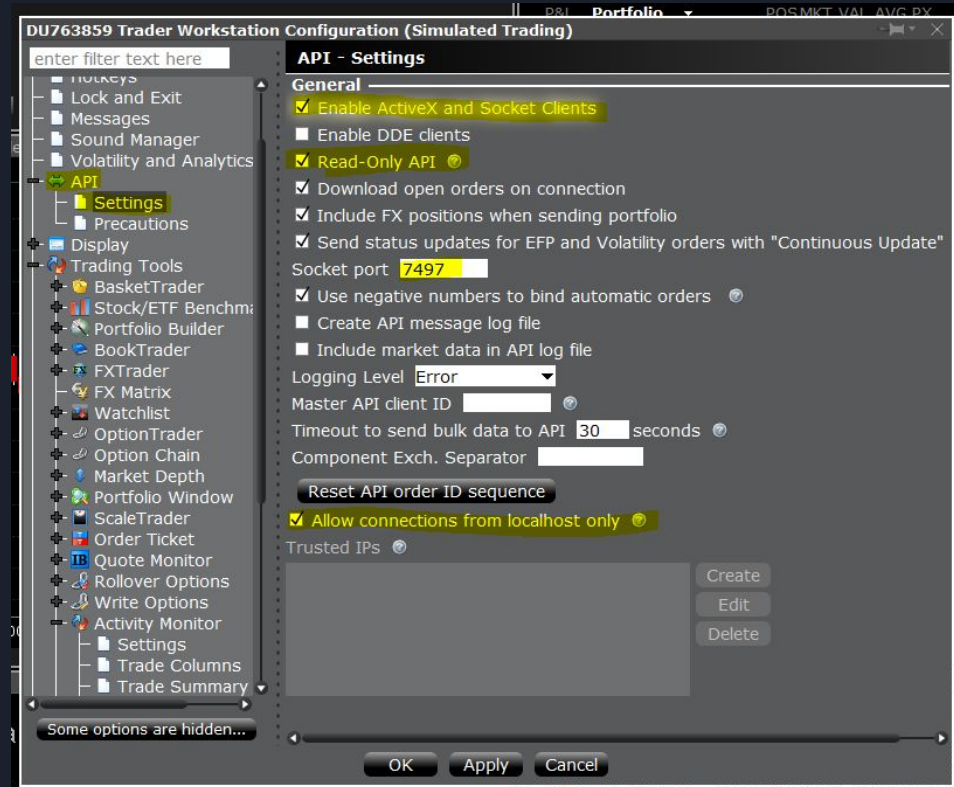
[Download IB Gateway](#)

- [Python](#) `_>=3.4`
- [Pandas](#) (tested to work with `>=0.18.1`)
- [Numpy](#) (tested to work with `>=1.11.1`)
- [ØMQ](#) (tested to with with `>=15.2.1`)
- [PyMySQL](#) (tested to with with `>=0.7.6`)
- [pytz](#) (tested to with with `>=2016.6.1`)
- [dateutil](#) (tested to with with `>=2.5.1`)
- [Nexmo](#) for SMS support (tested to with with `>=1.2.0`)
- [Twilio](#) for SMS support (tested to with with `>=6.0.0`)
- [Flask](#) for the Dashboard (tested to work with `>=0.11`)
- [Requests](#) (tested to with with `>=2.10.0`)
- [Beautiful Soup](#) (tested to work with `>=4.3.2`)
- [IbPy2](#) (tested to work with `>=0.8.0`)
- [ezIBpy](#) (IbPy wrapper, tested to with with `>=1.12.56`)
- Latest Interactive Brokers' [TWS](#) or [IB Gateway](#) installed and running on the machine
- [MySQL Server](#) installed and running with a database for QTPyLib

Arquitectura



Start TWS



Blotter [[link](#)]



1. Deberemos configurar para que se conecte a nuestra base de datos y a IB

```
from qtpylib.blotter import Blotter

class MainBlotter(Blotter):
    pass # we just need the name

if name == " main ":
    blotter = MainBlotter(
        dbhost = "localhost", # MySQL server
        dbname = "qtpy",      # MySQL database
        dbuser = "robotrader", # MySQL username
        dbpass = "robotrader", # MySQL password
        ibport = 7497,         # IB port (7496/7497 = TWS, 4001 = IBGateway)
        orderbook = True      # fetch and stream order book data
    )

    blotter.run()
```

2. Cuando hagamos start parar , y escribir sobre el fichero CSV instrumentos a escanear

Añadir instrumentos



1. Para saber los datos de un instrumento, buscarlo en TWS y click derecho



2. En nuestro symbols deberemos poner
 - a. `SPX, IND, CBOE, USD, , 0.0,`
 - b. `AAPL, STK, SMART, USD, , 0.0,`
3. Volver a arrancar blotter y comprobar que está escribiendo en la base de datos

Primer Algo : prueba

Todos los algos interactúan con los datos y el mercado a través de estas funciones

Haremos una prueba con un algoritmo tonto y lo debugaremos para ver que nos llega la información:

- En `on_start` es lo que haremos al inicio del algoritmo , setear variables propias o estados iniciales
- En `on_tick` cada tick llegará información del mercado en `instrument`
- En `on_bar` cada barra
- ...etc

```
def on_start(self):  
    # optional method that gets called once upon start  
    pass  
  
def on_fill(self, instrument, order):  
    # optional method that gets called on every order fill  
    pass  
  
def on_orderbook(self, instrument):  
    # optional method that gets called on every orderbook change  
    pass  
  
def on_quote(self, instrument):  
    # optional method that gets called on every quote change  
    pass  
  
def on_tick(self, instrument):  
    # optional method that gets called on every tick received  
    pass  
  
def on_bar(self, instrument):  
    # optional method that gets called on every bar received
```

Descarga de precios historicos

- Te deja elegir varios proveedores
 - Google
 - Yahoo
 - IB
- Elijo IB porque tenemos cuenta y sera mas fiable

```
from qtpylib import workflow as wf
import settings
import datetime

'''resolution:: 1 sec, 5 secs, 15 secs, 30 secs, 1 min (default), 2 mins, 3 mins, 5 mins,
def download(symbol, start, resolution="1 hour"):
    # load your existing market data as Pandas DataFrame.
    # here, we'll download 1-min intraday data from Google
    startStr = datetime.datetime.strftime(start, format='%d/%m/%Y')

    # instrument, start, resolution = "1 min", blotter = None, output_path = None)
    external_data = wf.get_data_ib(symbol, start=startStr, resolution=resolution)

    # convert the data into a QTPyLib-compatible
    # data will be saved in ~/Desktop/AAPL.csv
    df = wf.prepare_data(symbol, data=external_data, output_path=settings.csvData)

    # store converted bar data in MySQL
    # optional, requires a running Blotter
    wf.store_data(df, kind="BAR")
```

Backtest y Live



Backtest


Los precios de los instrumentos en esas fechas deben haberse descargado previamente!

```
import settings
if __name__ == "__main__":
    strategy = CrossOver(
        instruments=[("AAPL", "STK", "SMART", "USD", "", 0.0,)],
        resolution="1H",
        backtest=True,
        start='2018-02-10', #YYY-MM-DD [HH:MM:SS[.MS]]
        end='2018-02-16',
        output='crossOver_portfolio.pkl',
        data=settings.csvData
    )
    strategy.run()
```

Live

```
if __name__ == "__main__":
    strategy = CrossOver(
        instruments=[("AAPL", "STK", "SMART", "USD", "", 0.0,)],
        resolution="1H"
    )
    strategy.run()
```

Dashboard [\[link\]](#)



All Algos ▾

OPEN POSITIONS											
ALGO	SYMBOL	DIR	QTY	ENTRY		PRICE	MARKET	TARGET	STOP	EXIT	
TEST STRATEGY	ESU2016	SRT	1	12-AUG 10:29:46 +0300 LMT		2,173.50	2,173.50	-	-	-	2,179.75 -6.25
TEST STRATEGY	ESU2016	LNG	1	12-AUG 10:12:56 +0300 MKT		2,173.25	2,173.25	2,173.75	2,172.75	-	2,179.75 6.50
TRADES											
ALGO	SYMBOL	DIR	QTY	ENTRY		PRICE	MARKET	TARGET	STOP	EXIT	
TEST STRATEGY	ESU2016	SRT	1	14-AUG 10:15:20 +0300 MKT		2,179.50	2,179.75	-	-	14-AUG 10:15:23 +0300 SGL	2,179.75 0.25
TEST STRATEGY	ESU2016	SRT	1	12-AUG 19:27:08 +0300 MKT		2,174.25	2,174.25	-	-	12-AUG 19:27:10 +0300 SGL	2,174.50 0.25
TEST STRATEGY	ESU2016	LNG	1	11-AUG 21:24:47 +0300 MKT		2,172.00	2,172.00	2,172.50	2,171.50	11-AUG 21:25:24 +0300 STP	2,171.50 -0.50

Para levantarlo en <http://localhost:5000>

```
# dashboard.py
from qtpylib.reports import Reports

class Dashboard(Reports):
    pass # we just need the name

if __name__ == "__main__":
    dashboard = Dashboard()
    dashboard.run()
```