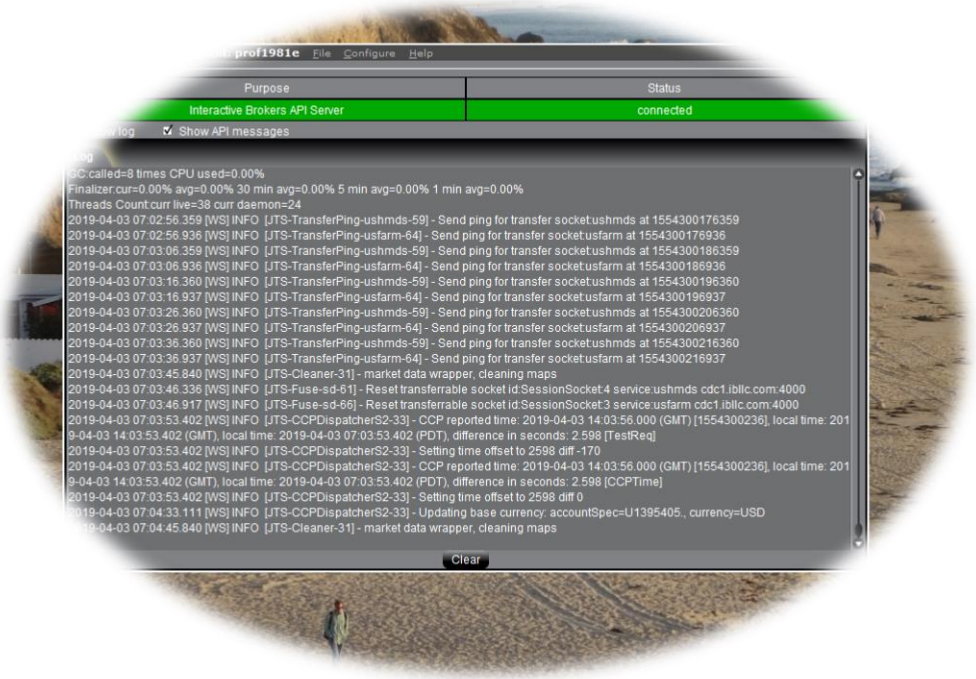


Developing an IB interface

using ib_insync

Key IB Programs

and what they do ..

Econ 136

Spring 2020

Jupyter

- ib_inpos_strangle_v1.8 dev.ipynb
  - ✓ also v1.7_spy
  - ✓ most recently tested/used: Used daily (non-Jup) as a daemon
  - ✓ status: works well – spy version is frequently used
  - ✓ purpose: This program is used specifically to monitor the daily IV and net value of a strangle that has already been put in position. Therefore the user supplies the two strike prices and the 252-day (or 90 or 30-day) historical volatility. One version autoloads HV from hv_iexmaster_short.py.
- ibapistranglev1_9_dev.ipynb
  - ✓ most recently tested/used: Used almost daily
  - ✓ status: works well – spy version is frequently used
  - ✓ purpose: This model is designed to select options for a strangle given input about the stock and the earnings date. This model is substantially automated. User must provide (1) the stock symbol, (2) expiry (which will be the friday of the week of earnings), and sigma, which must come from the hvieksmaster program or a file written by that program (and that can now be autoloaded for hv_iexmaster.short.py. This program now adjusts strikes to make sure that neither is too close to the money. Multiple tests in July 2019 show that it works very well in doing this.. See ibapi_experiment for early documentation for this.
- monte_carlo_stock_price_v1_3
  - ✓ student version also, used in class Spr 2020
  - ✓ this does include the half/variance adjustment
  - ✓ most recently tested/used: needs to be refitted for either jump diffusion or Poisson.
  - ✓ status: seemed to work well last time used
  - ✓ purpose: This is a version of the Monte Carlo simulator that is consistent with the modeling contained in the **Monte Carlo Simulations** lecture in Economics 136, assuming **Geometric Brownian Motion**. Prepared by Professor Evans on March 3, 2019, modified in April. THIS STILL NEEDS TO BE CLEANED UP, MOSTLY BY MODIFYING INDEXES TO NUMPY LIST COMPREHENSION.

Jupyter (page 2)

- hviexmaster_v3_5_ipynb
  - ✓ important program!
  - ✓ most recently tested/used: February 8, 2020
  - ✓ status: graphics repaired on 4 Jan and program works very well except IEX no longer provides earnings
  - ✓ purpose: The program accepts 2 years of stock or ETF data from IEXCloud, calculates daily continuous growth rates, fits to a Gaussian distribution and calculates drift and various estimates of sigma and XSigma. This version includes the Kolmogorov-Smirnov test for normalcy, accepts earnings dates and outputs to an Excel file. A feature was added that removes kurtosis.

- hviexmaster_short_v3_1.ipynb
  - ✓ extremely useful utility program!
  - ✓ most recently tested/used: Non-Jupyter version used almost daily
  - ✓ status: works very well, no graphics,
  - ✓ purpose: This is a quick version of hviexmaster (long) that downloads only one year of data and quickly prints out the main volatility dataframe as a Pandas dataframe for quick reference. This is now converted to a method that can me called by any program that needs up-to-date volatility numbers,

- ibapi_call_option_limit_order_v1_3.ipynb
  - ✓ most recently tested/used: used almost daily
  - ✓ status: core program has been used over and over and works extremely well, BUT there may still be a small asyncio bug in the program, which is why we can't yet rely on it outside of Jupyter.
  - ✓ purpose: This model is designed to issue a limit order to buy or sell a call option. This version uses an algo to figure out the price for the limit order.

Jupyter (page 3)

- ibapi_put_option_limit_order_v1_3.ipynb
  - ✓most recently tested/used: almost daily, usually with the call version to buy or sell strangles.
  - ✓status: [same as call program].
- call_ITM_prob_v4_1.ipynb (and a put version and a call-True version)
  - ✓ most recently tested/used: February 11, 2020
  - ✓ status: converted to an internal method in finutil.py [note: employs the Ito Method] and is currently being further modified to calculate both the probability of being ITM (it's current purpose) and the delta simultaneously.
  - ✓ purpose: This program allows calculation of the true probability the a call option will be ITM (and OTM of course).
- callidv.ipynb (and a put version, putidv.ipynb)
  - ✓ most recently tested/used: January 12, 2020
  - ✓ status: now a student model using crude iteration used in Econ 136. Replaced by the callidv (and similar) internal method of finutil.py, which uses my "divide and conquer" conversion technique.
  - ✓ purpose: calculates the idv for a call, non automated (old model)
- earn_calender_ipynb
  - ✓ Uses the REST API for http:.//www.earningscalendar.net
  - ✓ Various options for downloading JSON data for extensive earnings info
  - ✓ most recently tested/used: December 2019

non-Jupyter

finutil.py is my master financial utility program which has condensed many financial calculations to callable methods. These feed into the IB interface algo programs as utilities. Includes, among others ...

- csnd - integrates a standard normal distribution up to some sigma.
- cnd - integrates a Gaussian distribution up to some value.
- copo - calculating the BSM call option price, traditional model.
- dcount - time-discount function to discount the value of a future payment discounted at the risk-free rate.
- itm_call - the estimator for the probability that the call will be ITM at expiry.
- itm_option - itm_call and itm_put rolled into one.
- itm_put - the estimator for the probability that the call will be ITM at expiry.
- norm_dist - accepts a single value for x, mu, and sigma and returns a scalar solution for the pdf
- norm_dist_vec – returns a vector (NUMPY array) solution for the norm pdf, mostly for mapping.
- norm_dist_cdf - integrates a normal vector using a lambda function and scipy's integration.quad function
- otranche – a brute force option tranche value calulator used by the Taboga option pricing models and oidv.
- ftranche – a brute force full tranche value calculator used by the Aruba options pricing model (and other apps)
- oidv – calculates the implied daily volatility of a call or put using my “divide and conquer” iteration (fast!)
- oidvnm - calculates the implied daily volatility of a call or put using Newton’s Method.
- popo - calculating the BSM put option price, traditional model.
- snormpdf - maps the standard normal probability density function of a point.
- stan_norm_dist – same as snormpdf, but designed for use with lambda function.
- stand_norm_cdf - integrates a standard normal vector using a lambda function and scipy's integration quad
- tdecay - calculate one-day time decay (for Taboga model)

IB and ininsync documentation (very limited) ..

[Although limited, you should still review this before even starting. Allow many hours of time if you want to truly understand what you are doing].

We are using the Interactive Brokers API but that is primarily designed to be used with C++ or Java. To use it with Python it is best if you use a 3rd party interface. By far the most robust seems to be ibinsync. Fortunately the internal comment-based documentation is very useful ... best way to figure out how it works is to look directly at their wrappers.

The Interactive Brokers gethub API instruction set (you **must** review the part of this that shows how to set up TWS to accept a handshake from you API):

https://interactivebrokers.github.io/tws-api/#gsc.tab=0

To get started on ibinsync (3rd party extension to IB API):

https://rawgit.com/erdewit/ib_insync/master/docs/html/index.html

and for their very useful (for examples) Notebook see:

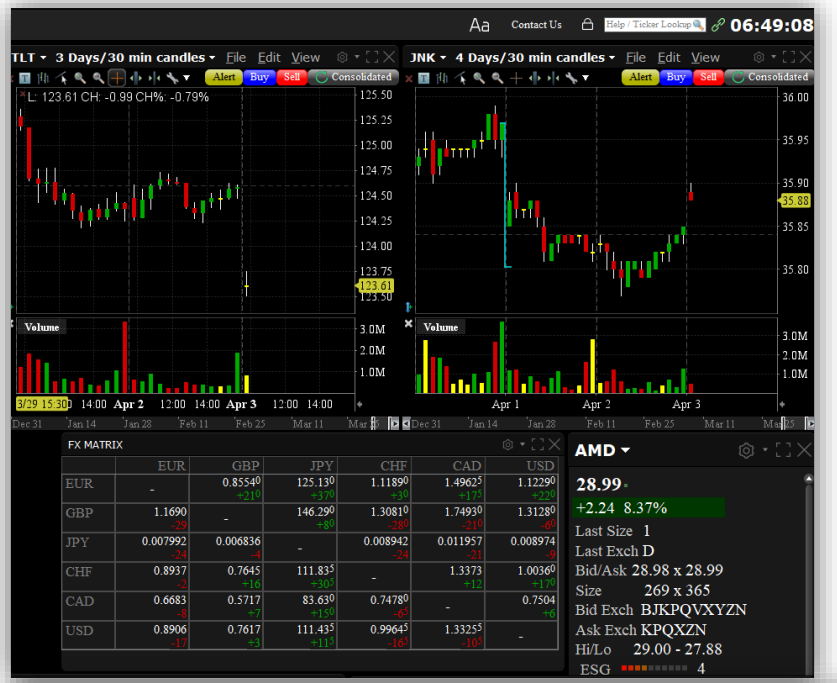
https://rawgit.com/erdewit/ib_insync/master/docs/html/notebooks.html

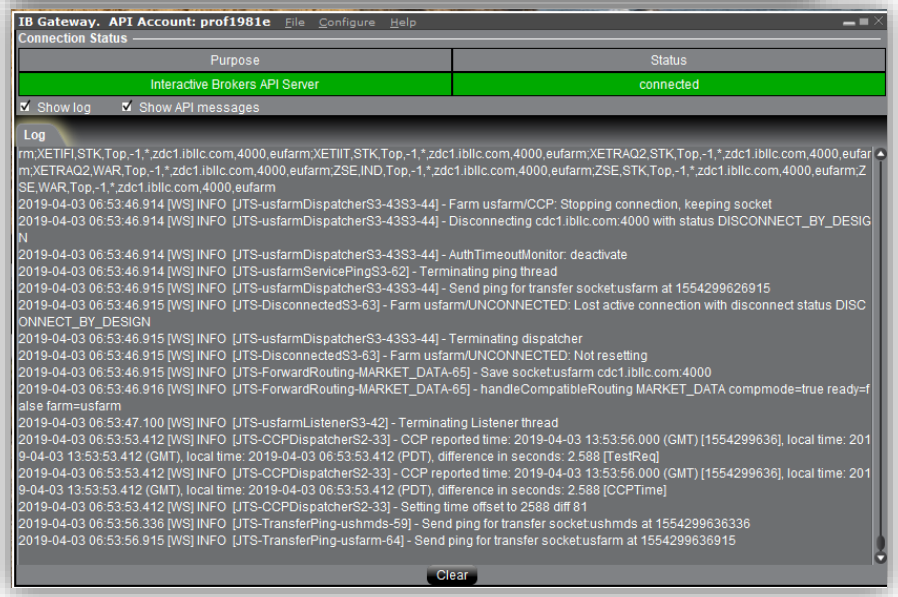
The actual IB interface ..

IBKR servers

LOGIN

LOGIN





The handshake (and beginning of loop):

```
In [86]:  1 import sys
          2 util.startLoop()
          3 ib = IB()
          4 ib.connect('127.0.0.1', 7496, clientId=7)

Out[86]: <IB connected to 127.0.0.1:7496 clientId=7>
```

The handshake and exit:

```
In [80]:
1  import math
2  import numpy as np
3  import datetime
4  from datetime import date
5  import sys
6  sys.path.append('c:/Users/Prof Gary Evans/Dropbox/PyGo/PyFi')
7  import finutil as fu
8  import timeutil as tu
9  sys.path.append('d:/TWS API/source/pythonclient/')
10 import ib_insync
11 from ib_insync import *
```

(This top command is starting an asyncio loop. This program runs asyncio and that is not optional).

Interactive Brokers handshake

```
In [7]:
util.startLoop()
ib = IB()
ib.connect('127.0.0.1', 7496, clientId=26)

Out[7]:
<IB connected to 127.0.0.1:7496 clientId=26>
```

(can be any integer, must be different for every process that you are running, and must be closed out with disconnect)

```
In [78]:
1  ib.disconnect()
```

The arrangement of ib market sources into “contracts” ...

In [6]:
```
1 contract = Stock('GLD', 'SMART', 'USD')
2 cds = ib.reqContractDetails(contract)
3 cds
```

Out[6]:
```
[ContractDetails(summary=Contract(conId=51529211, symbol='GLD', secType='STK', exchange='SMART', primaryExchange='ARCA', cu
rrency='USD', localSymbol='GLD', tradingClass='GLD'), marketName='GLD', minTick=0.01, orderTypes='ACTIVETIM,ADJUST,ALERT,AL
GO,ALLOC,AON,AVGCOST,BASKET,COND,CONDORDER,DARKONLY,DARKPOLL,DAY,DEACT,DEACTDIS,DEACTEOD,DIS,GAT,GTC,GTD,GTT,HID,IBKRATS,IC
E,IOC,LIT,LMT,LOC,MIT,MKT,MOC,MTL,NGCOMB,NODARK,NONALGO,OCA,OPG,OPGREROUT,PEGBENCH,POSTONLY,PREOPGRTH,REL,RTH,SCALE,SCALEOD
D,SCALERST,SNAPMID,SNAPMKT,SNAPREL,STP,STPLMT,SWEEP,TRAIL,TRAILLIT,TRAILLMT,TRAILMIT,WHATIF', validExchanges='SMART,AMEX,CB
OE,ISE,CHX,ARCA,ISLAND,VWAP,DRCTEDGE,NSX,BEX,BATS,EDGEA,CSFBALGO,JEFFALGO,BYX,IEX,PSX', priceMagnifier=1, longName='SPDR GO
LD SHARES', timeZoneId='EST5EDT', tradingHours='20180102:0400-2000;20180103:0400-2000;20180104:0400-2000;20180105:0400-2000
;20180106:CLOSED;20180107:CLOSED;20180108:0400-2000;20180109:0400-2000;20180110:0400-2000;20180111:0400-2000;20180112:0400-
2000;20180113:CLOSED;20180114:CLOSED;20180115:CLOSED;20180116:0400-2000;20180117:0400-2000;20180118:0400-2000;20180119:0400
-2000;20180120:CLOSED;20180121:CLOSED;20180122:0400-2000;20180123:0400-2000;20180124:0400-2000;20180125:0400-2000;20180126:
0400-2000;20180127:CLOSED;20180128:CLOSED;20180129:0400-2000;20180130:0400-2000;20180131:0400-2000;20180201:0400-2000;20180
202:0400-2000;20180203:CLOSED;20180204:CLOSED;20180205:0400-2000', liquidHours='20180102:0930-1600;20180103:0930-1600;20180
104:0930-1600;20180105:0930-1600;20180106:CLOSED;20180107:CLOSED;20180108:0930-1600;20180109:0930-1600;20180110:0930-1600;2
0180111:0930-1600;20180112:0930-1600;20180113:CLOSED;20180114:CLOSED;20180115:CLOSED;20180116:0930-1600;20180117:0930-1600;
20180118:0930-1600;20180119:0930-1600;20180120:CLOSED;20180121:CLOSED;20180122:0930-1600;20180123:0930-1600;20180124:0930-1
600;20180125:0930-1600;20180126:0930-1600;20180127:CLOSED;20180128:CLOSED;20180129:0930-1600;20180130:0930-1600;20180131:09
30-1600;20180201:0930-1600;20180202:0930-1600;20180203:CLOSED;20180204:CLOSED;20180205:0930-1600', mdSizeMultiplier=100, ag
gGroup=1)]
```

Key ib_insync commands (getting quotes):

ib.reqMktData(stock,"",False,True): getting snapshot data

Identify the stock:

```
In [9]: stock = Stock(stosym, 'SMART', 'USD')
```

Grab the Level 1 quote:

```
In [10]: l1_quote = ib.reqMktData(stock,"",False,True)
         l1_quote
Out[10]: Ticker(contract=Stock(symbol='SPY', exchange='SMART', currency='USD'), ticks=[], tickByTicks=[], domBids=[], domAsks=[], do
         mTicks=[])
```

ib.reqMktData(stock,"",False,False):

According to the documentation when you use the "False" value supposedly tick data rather than snapshot is being returned. It is not clear that this is true unless it is returning a snapshot of the most recent tick data, which according to how it works for options, it is not supposed to be doing (it is supposed to create an empty bucket, then drop the tick data in when it appears). For a stock like NATH that would introduce considerable latency, which we do not see here. If in doubt for just a level 1 inquiry use the True,False option.

```
In [40]: 1 s_quote2 = ib.reqMktData(nath,"",False,False)
         2 s_quote2
Out[40]: Ticker(contract=Stock(conId=272249, symbol='NATH', exchange='SMART', primaryExchange='NASDAQ', currency
         'NATH', tradingClass='NMS'), time=datetime.datetime(2018, 6, 8, 19, 45, 16, 930772, tzinfo=datetime.timezone.utc),
         5, bidSize=5, ask=89.5, askSize=3, last=89.5, lastSize=1, prevBidSize=3, volume=101, open=88.0, high=90.0, low=88.0,
         88.0)
```

Getting true tick data is not working properly at IBKR right now.

Key ib_insync commands (getting quotes 2):

ib.reqTickers(stock):

The first option considered is using ib.reqTickers(), a method defined in ib.py, to download the data arranged in a list by keyword. According to the documentation it returns a snapshot in a LIST! Note how the output of s_ticker_snap returns a list, but in our case because of how we have set up our definition of the contract to include only one contract, the list has only one element.

In [36]:
```
1 s_ticker_snap = ib.reqTickers(nath)
2 s_ticker_snap
```

Out[36]:
```
[Ticker(contract=Stock(conId=272249, symbol='NATH', exchange='SMART', primaryExchange='NASDAQ', currency='USD', localSymbol
='NATH', tradingClass='NMS'), time=datetime.datetime(2018, 6, 8, 19, 44, 30, 343290, tzinfo=datetime.timezone.utc), bid=89.
05, bidSize=3, ask=89.5, askSize=3, last=89.5, lastSize=1, volume=101, open=88.0, high=90.0, low=88.0, close=88.0)]
```

But, because this is a list of one element, when we download a attribute like "bid" we have to specify the list with the [0] symbol. See how this request for bid differs from the next example (s_quote).

In [37]:
```
1 s_ticker_snap[0].bid
```

Out[37]:
```
89.05
```

The tick access has not been working in late 2019 and 2020 for IBKR. This is left here as reference in case it is resumed,

The tick-latency issue (see program ibapioptions_slo_tick.ipynb).

Interactive Brokers API: detect options chain and extract Level 1 quote

This is the slow tick program.

Ditto the previous slide ...

This is ibapioptions_slo_tick, developed June 8, 2018 by Professor Evans.

This is the program that shows the example of (apparent) extreme latency because the program is asking for tick data rather than snapshot data for the puts and calls here (in a strangle setup). When tick data are requested, the program sets up an empty bucket and waits for new data to appear. It does not display the most recent tick. For that you use snapshot! Here also we are asking for best bid and best ask, so it will reflect a limit order only if it establishes a new best bid or best ask.

In [4]:
```
option = Option('IWM', exchange='SMART')
cds = ib.reqContractDetails(option)
```

In [6]:
```
call_contract = [c for c in contracts if
        c.lastTradeDateOrContractMonth == '20180615' and
        c.strike == 167.0 and
        c.right == 'C']
```

(stock was 166.20 or so)

In [15]:
```
put_contract = [c for c in contracts if
        c.lastTradeDateOrContractMonth == '20180615' and
        c.strike == 165.0 and
        c.right == 'P']
```

Stock Level 1 quote quick and dirty solution

First, look at the documentation at ibapi_experiment.ipynb or .html

1. All program initialize with these four sets of commands:

The handshake (and beginning of loop):

```
In [8]: import sys
        util.startLoop()
        ib = IB()
        ib.connect('127.0.0.1', 7496, clientId=29)

Out[8]: <IB connected to 127.0.0.1:7496 clientId=29>
```

Identify the stock:

```
In [9]: stock = Stock(stosym, 'SMART', 'USD')
```

(7496 for a paid account, 7497 for a paper account, clientId can be any integer and must be changed if you are running another IB program.)

2. These specifically will allow you to download level 1:

Grab the Level 1 quote:

```
In [10]: l1_quote = ib.reqMktData(stock,"",False,True)
         l1_quote

Out[10]: Ticker(contract=Stock(symbol='SPY', exchange='SMART', currency='USD'), ticks=[], tickByTicks=[], domBids=[], domAsks=[], do
         mTicks=[])

In [11]: s_last = l1_quote.last
         s_bid = l1_quote.bid
         s_ask = l1_quote.ask
         s_peg = (s_ask+s_bid)/2.0
         s_peg = round(s_peg,2)
         print("Last:",s_last," Bid: ",s_bid," Ask: ",s_ask,"Peg: ",s_peg)

         Last: 337.18  Bid:  337.18  Ask:  337.19 Peg:  337.19
```

and ask, bidsize etc.

The tick-latency issue (see program ibapioptions_slo_tick.ipynb).

(results):

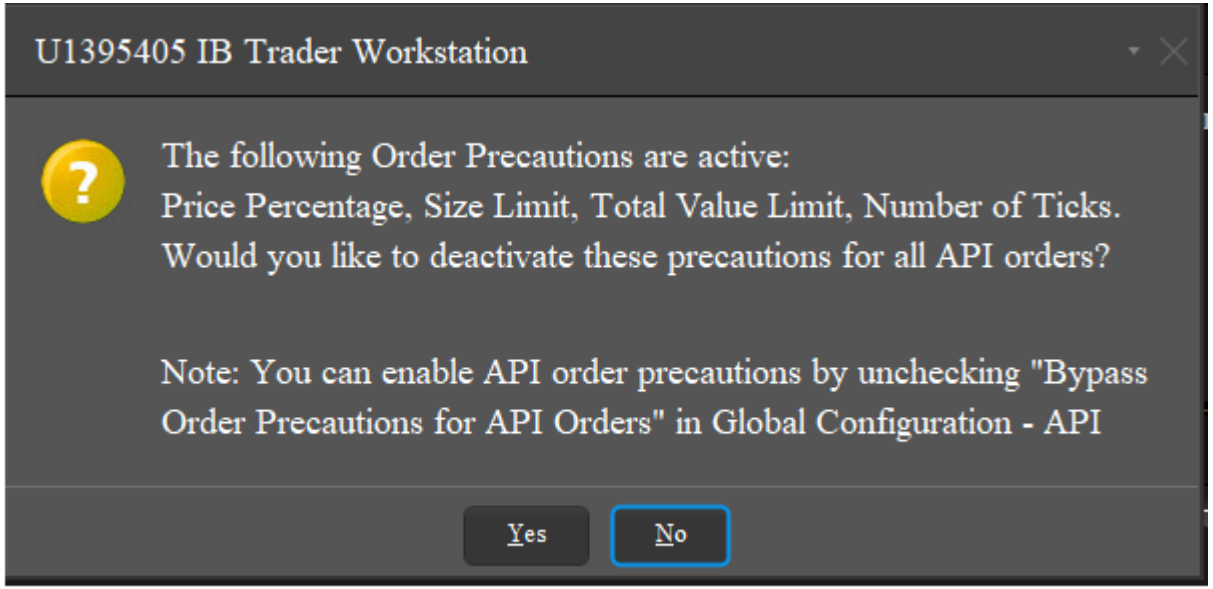
```
In [9]:  1 call_quote = ib.reqTickers(*call_contract)

In [10]: 1 call_quote[0].bid
Out[10]: 0.81

In [11]: 1 call_quote[0].bidSize
Out[11]: 68

In [12]: 1 call_quote[0].ask
Out[12]: 0.83

In [13]: 1 call_quote[0].askSize
Out[13]: 2138

In [14]: 1 end1 = time()
         2 print ('Elapsed time: {:.3f}'.format(end1-start))
         Elapsed time: 12.322
```

```
In [18]: 1 put_quote = ib.reqTickers(*put_contract)

In [19]: 1 put_quote[0].bid
Out[19]: 0.69

In [20]: 1 put_quote[0].bidSize
Out[20]: 1096

In [21]: 1 put_quote[0].ask
Out[21]: 0.71

In [22]: 1 put_quote[0].askSize
Out[22]: 699

In [23]: 1 end = time()
         2 print ('Elapsed time: {:.3f}'.format(end-start))
         Elapsed time: 24.331
```

(cumulative value)

That's unacceptable, but it is because we are asking for raw tick value with the ib_reqTickers command. We need to request a "snapshot," which will show the active bid!

The first market order, for 10 shares of AMD worked after we answered Yes to this:



+/-	Time	Fin Instrument	Action	Quantity	Price	Exch.	Commission	Account
	10:28:42	AMD	BOT	10	18.9489	DARK	1.00	U1395405

The first limit order, set to trigger a buy at 5 cents below current best bid!

Note: This has not been checked since summer ...

In [13]:
```
s_last = l1_quote.last
s_bid = l1_quote.bid
s_ask = l1_quote.ask
s_peg = (s_ask+s_bid)/2.0
print("Last:",s_last," Bid: ",s_bid," Ask: ",s_ask,"Peg: ",s_peg)
```

```
Last: 18.86  Bid:  18.86  Ask:  18.87 Peg:  18.865000000000002
```

Specifiy the limit order that you want to make. Use 'BUY' or 'SELL' followed by order size in shares, followed by the limit order price. In this example we are using the bid minus 5 cents.

In [15]:
```
order = LimitOrder('BUY',10, (s_bid - 0.05))
```

In [16]:
```
trade = ib.placeOrder(stock, order)
trade
```

	Action	Type	Details	Quantity	Fill Px	
AMD	BUY	LMT	LMT 18.81	0/10	-	Cancel
AMD	BUY	MKT		10	18.948	

10:57:10	AMD	BOT	10	18.81	ISLAND	1.00 U1395405
10:28:42	AMD	BOT	10	18.9489	DARK	1.00 U1395405

Call or put option quotes (this is complicated) ...

(page 1)

Grab the option strikes

In [12]:
```
option = Option(stosym, lastTradeDateOrContractMonth=expiry, right='C', exchange='SMART', currency='USD')
cds = ib.reqContractDetails(option)
```

To see the debugging commands that were once here, see version 1.8_working

In [13]:
```
length = len(cds)
length
```

Out[13]: 175

In [14]:
```
strikes = [0 for x in range(length)]
for i in range(length):
    strikes[i] = cds[i].contract.strike
```

Because the strikes are sometimes sorted but not always sorted, we have to sort them from low to high and then display them ... In Jupyter, this works better if you are using numpy arrays rather than Python lists.

In [15]:
```
u_strike_r = np.asarray(strikes)
strike_r = np.sort(u_strike_r)
```

In [16]:
```
stock_price = s_peg
stock_price
```

Out[16]: 337.19

Call or put option quotes ...

(page 2) ... this is an example of letting the program choose strikes that are ATM. Given that you now have an ordered list of all strikes, you can use any sorting or choice criteria that you want ...

Choose the strikes ...

Note: First we simply chose the call and put that are both out of the money but closest to the money ...

In [17]:
```
c_str_elem = int(0)
while strike_r[c_str_elem] < stock_price:
    c_str_elem +=1
call_strike = strike_r[c_str_elem]
call_strike
```

Out[17]: 337.5

In [18]:
```
p_str_elem = c_str_elem - 1
put_strike = strike_r[p_str_elem]
put_strike
```

Out[18]: 337.0

In [19]:
```
spread = call_strike - put_strike
spread
```

Out[19]: 0.5

Call or put option quotes ...

(page 3)

Get the option quotes from IB:

In [26]:
```
call_option = Option(stosym, lastTradeDateOrContractMonth=expiry, strike=call_strike, right='C', exchange='SMART',
currency='USD')
```

In [27]:
```
call_option
```

Out[27]:
```
Option(symbol='SPY', lastTradeDateOrContractMonth='20200221', strike=337.5, right='C', exchange='SMART', currency='USD')
```

In [28]:
```
put_option = Option(stosym, lastTradeDateOrContractMonth=expiry, strike=put_strike, right='P', exchange='SMART',
currency='USD')
```

In [29]:
```
put_option
```

Out[29]:
```
Option(symbol='SPY', lastTradeDateOrContractMonth='20200221', strike=337.0, right='P', exchange='SMART', currency='USD')
```

For months this program ran successfully with these two commands: call_quote = ib.reqMktData(call_option,"",False,True) put_quote = ib.reqMktData(put_option,"",False,True), then on 4/22 these generated errors that indicated no permissions??? When I realized that the inpos program used these permissions below, I just swapped and they worked. I haven't figured out why. This change was the basis for upgrade v1.7.

In [30]:
```
# call_quote = ib.reqMktData(call_option,"",False,True)   [old method]
call_quote = ib.reqMktData(call_option,"",True,False)
```

In [31]:
```
# put_quote = ib.reqMktData(put_option,"",False,True)    [old method]
put_quote = ib.reqMktData(put_option,"",True,False)
```

In [32]:
```
# call_last = call_quote.last
call_bid = call_quote.bid
call_ask = call_quote.ask
call_peg = (call_ask+call_bid)/2.0
call_peg = round(call_peg, 2)
print("Call Bid: {:.2f}, Ask: {:.2f},  Peg: {:.2f}.".format(call_bid,call_ask,call_peg))
```

```
Call Bid: 1.66, Ask: 1.68,  Peg: 1.67.
```

In [33]:
```
# put_last = put_quote.last
put_bid = put_quote.bid
put_ask = put_quote.ask
put_peg = (put_ask+put_bid)/2.0
put_peg = round(put_peg,2)
print("Put Bid: {:.2f}, Ask: {:.2f},  Peg: {:.2f}.".format(put_bid,put_ask,put_peg))
```

```
Put Bid: 1.64, Ask: 1.66,  Peg: 1.65.
```

In [34]:
```
position_cost = put_peg + call_peg
```

Call option limit order ...

Mudd (Palm Island Traders) finance

The order_spread_coefficient sets the limit order price as some percent between Bid and Ask.

```
In [4]: expiry = 20200221
        call_strike = 337.5
        action = 'BUY'
        order_size = 1
        order_spread_coefficient = 0.70
```

No user information is added below here unless you want to over-ride the limit price algo.

```
In [5]: call_option = Option(stosym, lastTradeDateOrContractMonth=expiry, strike=call_strike, right='C', exchange='SMART',
        currency='USD')
        cds = ib.reqContractDetails(call_option)

In [6]: call_option
Out[6]: Option(symbol='SPY', lastTradeDateOrContractMonth=20200221, strike=337.5, right='C', exchange='SMART', currency='USD')

In [7]: call_quote = ib.reqMktData(call_option,"",True,False)
```

Warning .. this is where the actual order is sent.

If in doubt, check all buy and sell, order size and price variables are set at the top.

```
In [8]: call_last = call_quote.last
        call_bid = call_quote.bid
        call_ask = call_quote.ask
        # call_peg = (call_ask+call_bid)/2.0
        call_spread = call_ask - call_bid
        call_spread = round(call_spread,2)
        call_peg_coeff = order_spread_coefficient
        call_peg = call_bid + (call_peg_coeff*call_spread)
        call_peg = round(call_peg,2)
        print("Call Last:",call_last," Bid: ",call_bid," Ask: ",call_ask, " Bid/Ask spread: ",call_spread, " Peg:",call_peg)

        Call Last: 1.8  Bid:  1.77  Ask:  1.79  Bid/Ask spread:  0.02  Peg: 1.78

In [9]: limit_order = LimitOrder(action,order_size,call_peg)
        limit_trade = ib.placeOrder(call_option, limit_order)
        limit_trade
```

.. resetting the price

```
In [11]: new_price = 1.83  # or whatever price you want to put in here.
         limit_order.lmtPrice = new_price
         limit_trade = ib.placeOrder(call_option, limit_order)
         limit_trade
```

... cancel the limit order

```
In [10]: ib.cancelOrder(limit_order)
```

Disconnect when done

```
In [10]: ib.disconnect()
```

```
In [5]: put_option = Option(stosym, lastTradeDateOrContractMonth=expiry, strike=put_strike, right='P', exchange='SMART',
        currency='USD')
        cds = ib.reqContractDetails(put_option)
```

... and for a put option

... as of Feb 2020, a nagging bug ...


```
67   # Do the IB handshake
68   #
69   ib = IB()
70   ib.connect('127.0.0.1', 7496, clientId=82)
71   print(ib.connect)
72   #
73   # Start the asyncio loop
74   #
75   util.patchAsyncio()
76   #
77   # Get the stock quote
78   #
79   # stock = Stock(stosym, 'SMART', 'USD') [the old command - what follows is new].
80   stock = Stock(symbol=stosym, exchange='SMART', currency='USD')
81   ib.qualifyContracts(stock)
82   l1_quote = ib.reqMktData(stock,"",True,False)
83   #
84   # This asyncio command below prevents the quotes from sending "nan"s instead of
85   # data.
86   #
87   ib.sleep(0.5)
88   s_last = l1_quote.last
89   s_bid = l1_quote.bid
90   s_bid_size = l1_quote.bidSize
91   s_ask = l1_quote.ask
92   s_ask_size = l1_quote.askSize
93   s_peg = (s_ask+s_bid)/2.0
94   s_peg = round(s_peg,2)
95   #
96   print("  {}, Last: {:.2f}, Bid: {:.2f}, Bid Size: {:d}, Ask: {:.2f}, Ask Size: {:d}, Peg: {:.2f}"
97       .format(stosym,s_last,s_bid,s_bid_size,s_ask,s_ask_size,s_peg))
98   print ("  Expiry: {}, days to expiry: {}".format(expiry,days))
99   print ("  Call strike: {:.2f}, Put strike: {:.2f}".format(call_strike,put_strike))
100  #
```

Starting asyncio, but calling it effectively from an ib-insync method already initializing it ... this is fine.

This is the problem ... this usage, although it works 90% of the time, is not correct.

asyncio is the primary Python utility designed to overcome I/O-bound latency (and other latency as well, but most applications are dealing with I/O latency). It is designed to overcome Python's very restrictive Global Interpretor Lock (GIL), which forces Python to be sequential. With GIL, when I/O bound, the program has to wait for a data bucket to fill before any other steps can be executed (this was why Go was developed by Google). Asyncio goes around the GIL.

References to the new API:

This was added in January 2019, at a time when it was clear that a new API was being developed for IB. These are mostly reference articles:

https://qoppac.blogspot.com/2017/03/interactive-brokers-native-python-api.html
Interactive Brokers native python api

For help on using ibinsync:

https://groups.io/g/insync/topics