ALGO TRADING IN PYTHON

#1 : TRADING STRATEGY AND BACKTESTING

Long Tran – Snap Innovations Pte Inc

COURSE OUTLINE

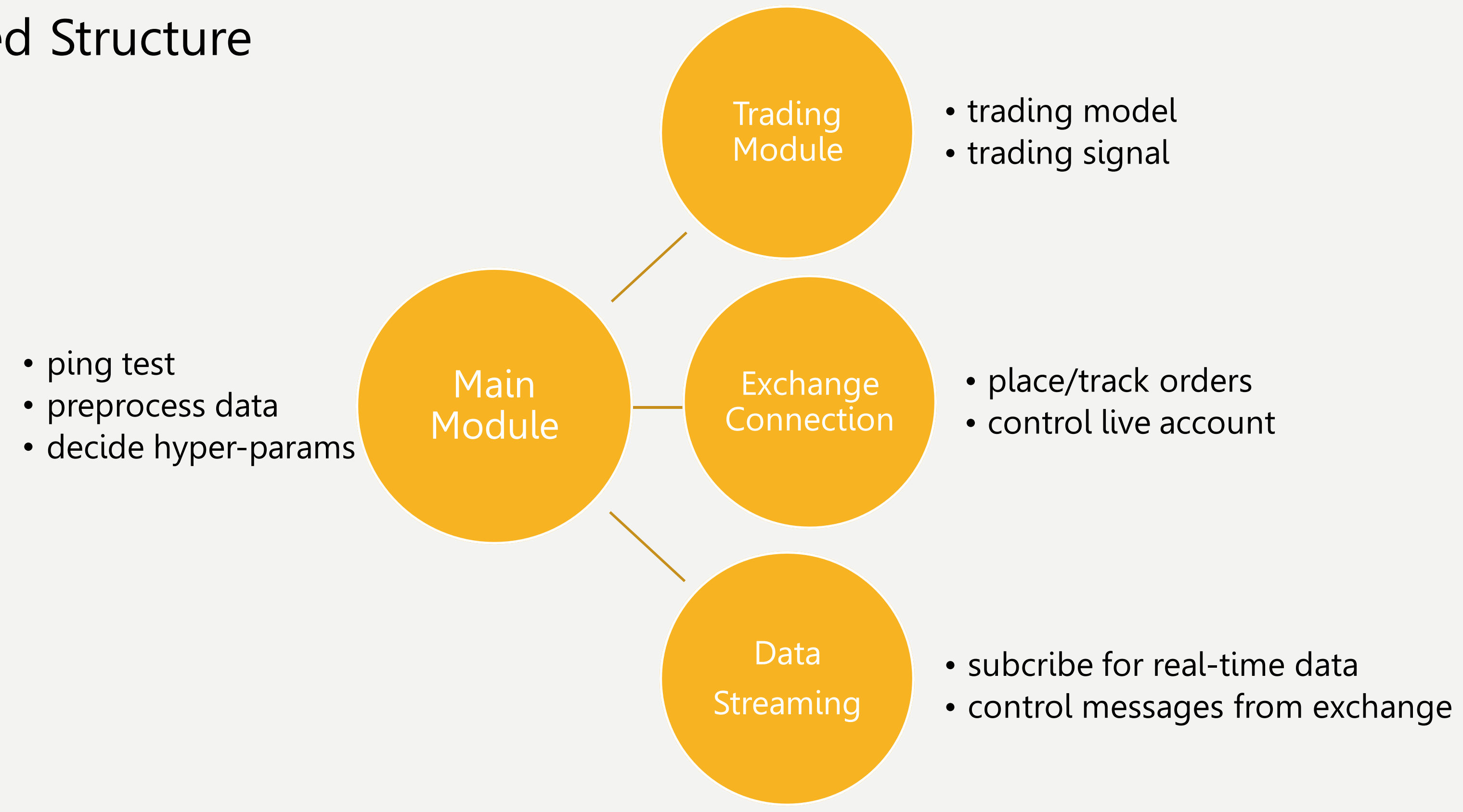
- Session/Week 1 : Trading strategy and backtesting in Python
- Session/Week 2 : Connect to the exchange (REST api)
- Session/Week 3 : Real-time data streaming (websocket)
- Session/Week 4 : Errors handling and Q&A

LAYOUT : SESSION #1

1. Modules Structure
2. Data Structure
3. Trading Model / Signal / Backtester Classes
4. Python tricks and tips
5. Coding exercises

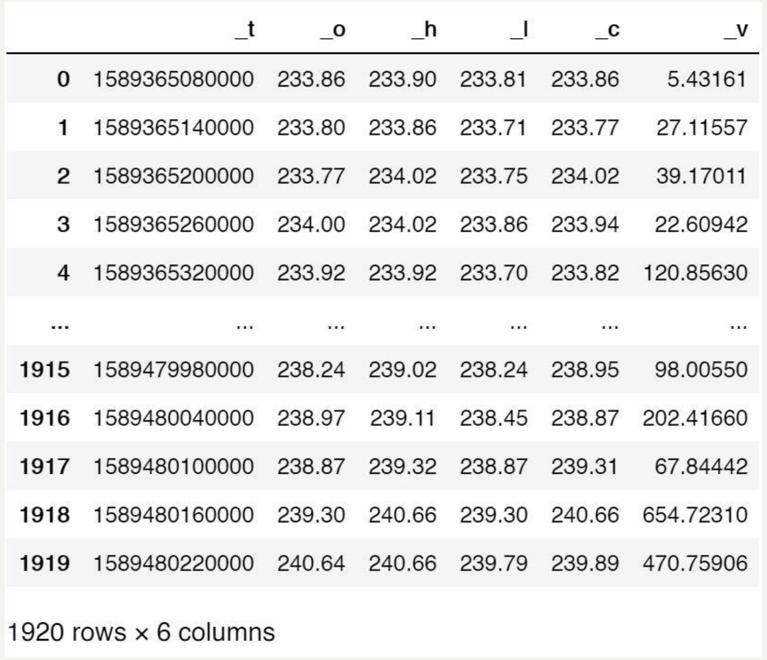
MODULES STRUCTURE

- Why?
  - Easier for debugging
  - Integration with new trading strategy
- Suggested Structure



DATA STRUCTURE

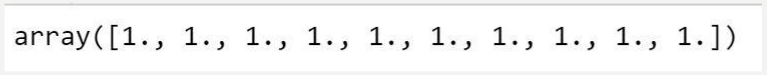
- pandas.DataFrame()



	_t	_o	_h	_l	_c	_v
0	1589365080000	233.86	233.90	233.81	233.86	5.43161
1	1589365140000	233.80	233.86	233.71	233.77	27.11557
2	1589365200000	233.77	234.02	233.75	234.02	39.17011
3	1589365260000	234.00	234.02	233.86	233.94	22.60942
4	1589365320000	233.92	233.92	233.70	233.82	120.85630
...	...	...	...	...	...	...
1915	1589479980000	238.24	239.02	238.24	238.95	98.00550
1916	1589480040000	238.97	239.11	238.45	238.87	202.41660
1917	1589480100000	238.87	239.32	238.87	239.31	67.84442
1918	1589480160000	239.30	240.66	239.30	240.66	654.72310
1919	1589480220000	240.64	240.66	239.79	239.89	470.75906

1920 rows × 6 columns

- numpy.array()

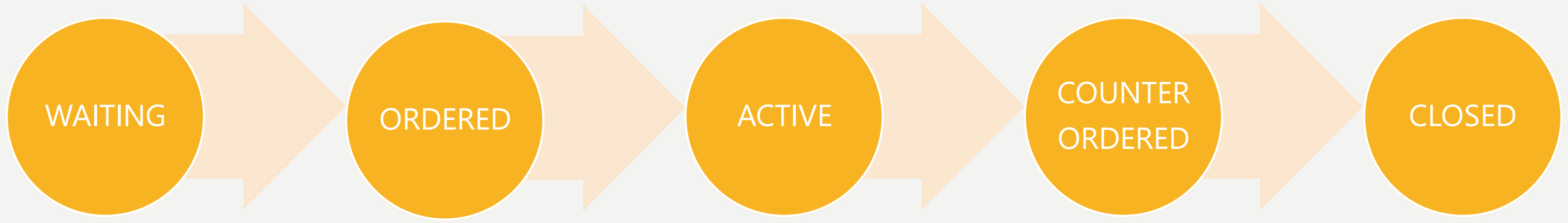


```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

- other packages: matplotlib, statsmodels, scipy, seaborn, sklearn, keras,…

TRADING SIGNAL (CLASS)

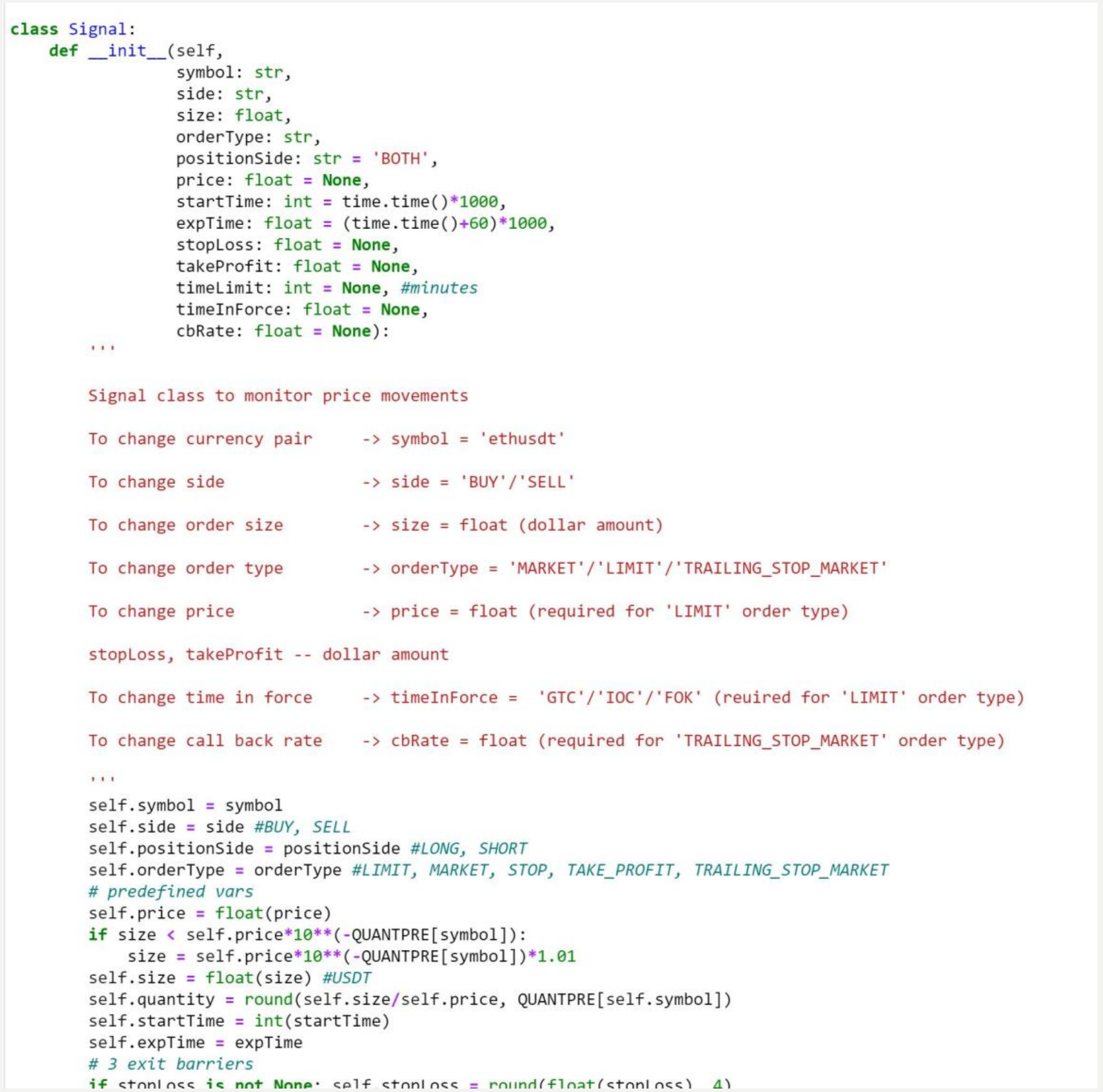
- Life span of a signal:



- Class properties:
  - symbol, side, size, orderType, excPrice, excTime, clsPrice, clsTime,...
  - pricePath, stopLoss, takeProfit, timeLimit,...
  - STATUS : waiting / ordered / active / cnt_ordered / closed
- Class functions:
  - check_status()
  - update_path()
  - exit_trigger()
  - ...

TRADING SIGNAL (CLASS)

- Example:



```python
class Signal:
    def __init__(self,
                 symbol: str,
                 side: str,
                 size: float,
                 orderType: str,
                 positionSide: str = 'BOTH',
                 price: float = None,
                 startTime: int = time.time()*1000,
                 expTime: float = (time.time()+60)*1000,
                 stopLoss: float = None,
                 takeProfit: float = None,
                 timeLimit: int = None, #minutes
                 timeInForce: float = None,
                 cbRate: float = None):
        '''

        Signal class to monitor price movements

        To change currency pair     -> symbol = 'ethusdt'

        To change side              -> side = 'BUY'/'SELL'

        To change order size        -> size = float (dollar amount)

        To change order type        -> orderType = 'MARKET'/'LIMIT'/'TRAILING_STOP_MARKET'

        To change price             -> price = float (required for 'LIMIT' order type)

        stopLoss, takeProfit -- dollar amount

        To change time in force     -> timeInForce =  'GTC'/'IOC'/'FOK' (reuired for 'LIMIT' order type)

        To change call back rate    -> cbRate = float (required for 'TRAILING_STOP_MARKET' order type)

        '''
        self.symbol = symbol
        self.side = side #BUY, SELL
        self.positionSide = positionSide #LONG, SHORT
        self.orderType = orderType #LIMIT, MARKET, STOP, TAKE_PROFIT, TRAILING_STOP_MARKET
        # predefined vars
        self.price = float(price)
        if size < self.price*10**(-QUANTPRE[symbol]):
            size = self.price*10**(-QUANTPRE[symbol])*1.01
        self.size = float(size) #USDT
        self.quantity = round(self.size/self.price, QUANTPRE[self.symbol])
        self.startTime = int(startTime)
        self.expTime = expTime
        # 3 exit barriers
        if stopLoss is not None: self.stopLoss = round(float(stopLoss), 4)
```

TRADING MODEL (CLASS)

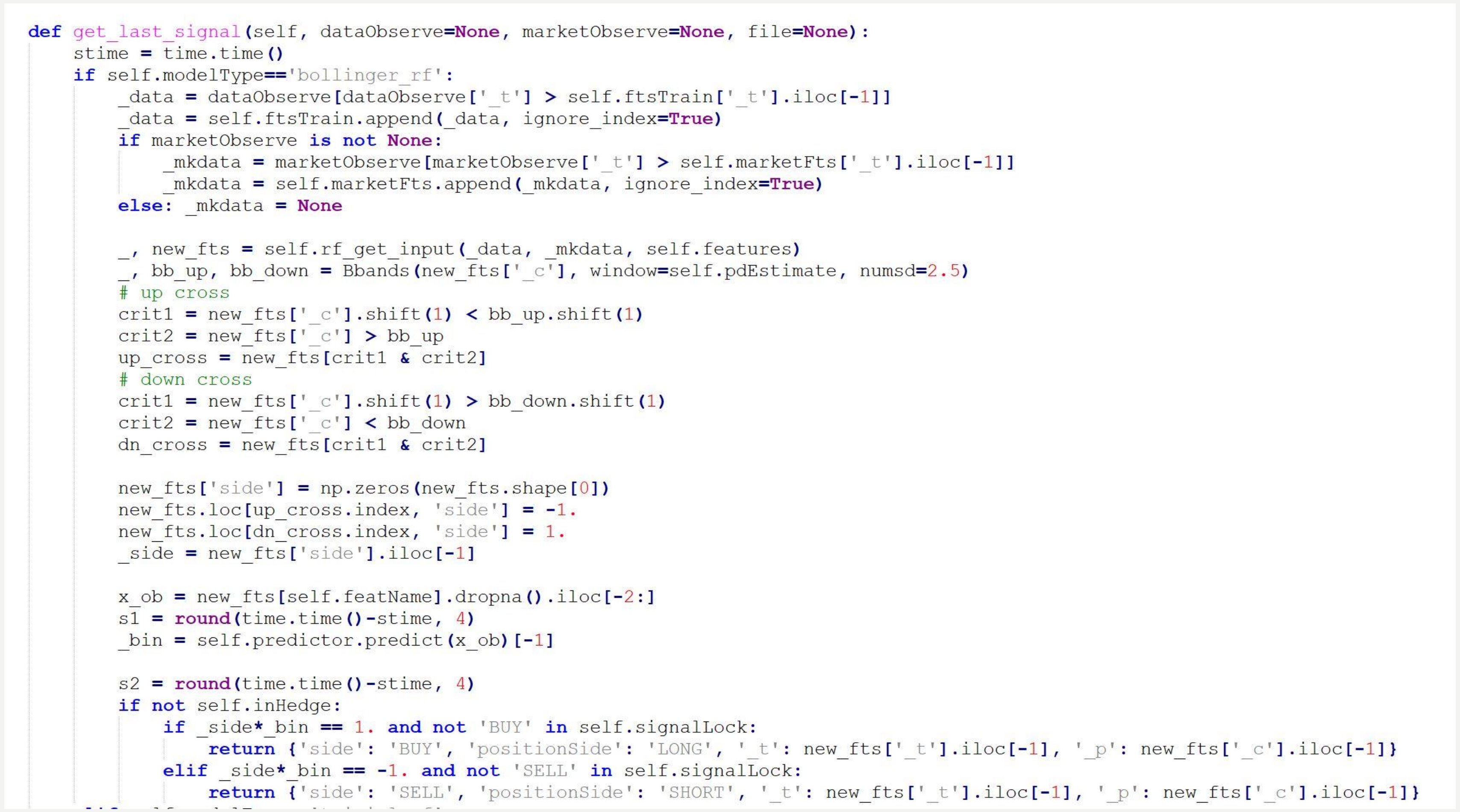
- Implement the logic:

  Input: Market Data → **Output**: BUY/SELL signals

- Class properties:
  - dataTrain, dataObserve
  - predefined hyper-params
- Class functions:
  - indicator_generator() (RSI, EMA, Stoch, Williams, OBV,…)
  - params_optimize()
  - get_last_signal()
  - …

TRADING MODEL (CLASS)

- Example:



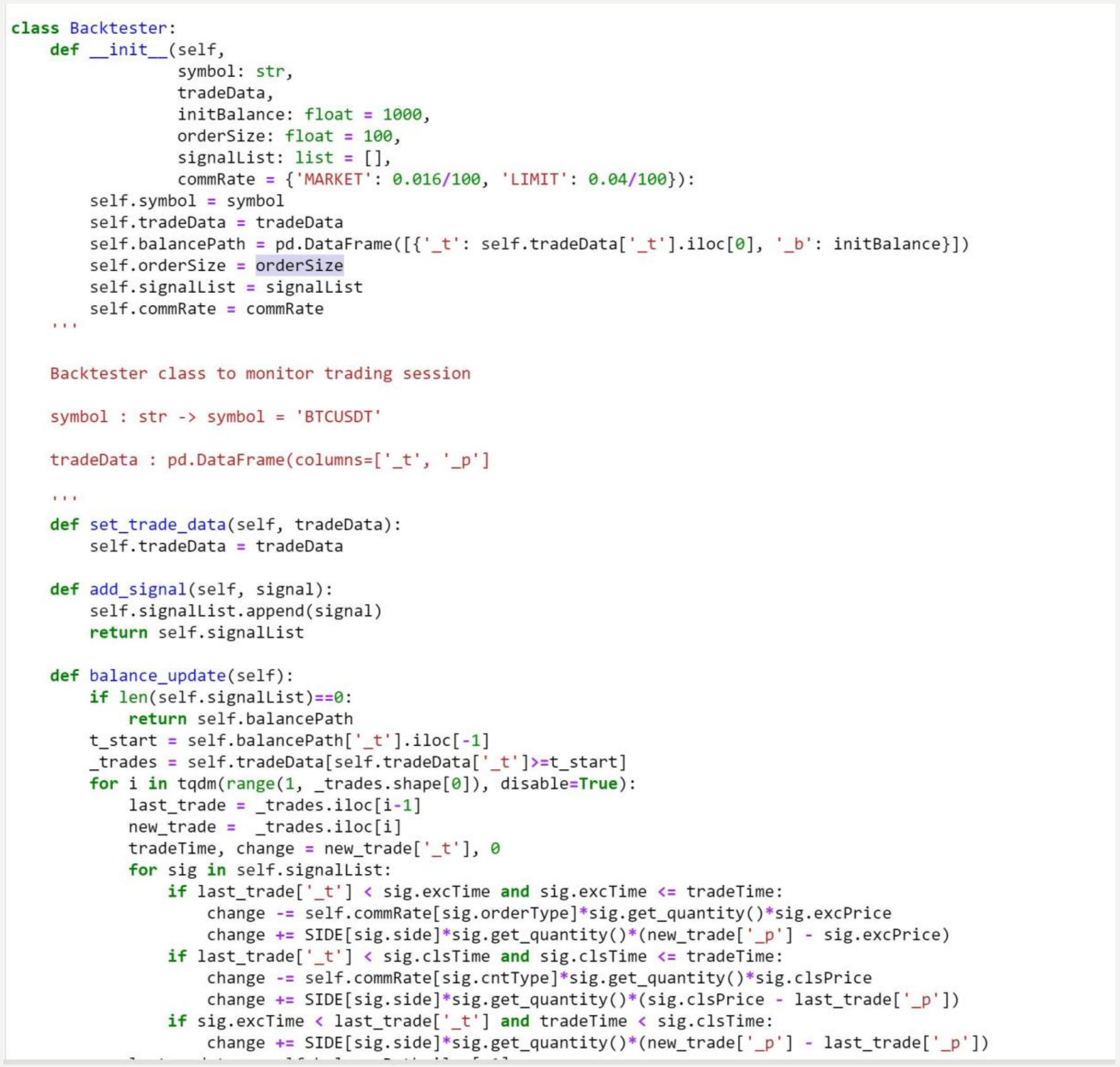
```python
def get_last_signal(self, dataObserve=None, marketObserve=None, file=None):
    stime = time.time()
    if self.modelType=='bollinger_rf':
        _data = dataObserve[dataObserve['_t'] > self.ftsTrain['_t'].iloc[-1]]
        _data = self.ftsTrain.append(_data, ignore_index=True)
        if marketObserve is not None:
            _mkdata = marketObserve[marketObserve['_t'] > self.marketFts['_t'].iloc[-1]]
            _mkdata = self.marketFts.append(_mkdata, ignore_index=True)
        else: _mkdata = None

        _, new_fts = self.rf_get_input(_data, _mkdata, self.features)
        _, bb_up, bb_down = Bbands(new_fts['_c'], window=self.pdEstimate, numsd=2.5)
        # up cross
        crit1 = new_fts['_c'].shift(1) < bb_up.shift(1)
        crit2 = new_fts['_c'] > bb_up
        up_cross = new_fts[crit1 & crit2]
        # down cross
        crit1 = new_fts['_c'].shift(1) > bb_down.shift(1)
        crit2 = new_fts['_c'] < bb_down
        dn_cross = new_fts[crit1 & crit2]

        new_fts['side'] = np.zeros(new_fts.shape[0])
        new_fts.loc[up_cross.index, 'side'] = -1.
        new_fts.loc[dn_cross.index, 'side'] = 1.
        _side = new_fts['side'].iloc[-1]

        x_ob = new_fts[self.featName].dropna().iloc[-2:]
        s1 = round(time.time()-stime, 4)
        _bin = self.predictor.predict(x_ob)[-1]

        s2 = round(time.time()-stime, 4)
        if not self.inHedge:
            if _side*_bin == 1. and not 'BUY' in self.signalLock:
                return {'side': 'BUY', 'positionSide': 'LONG', '_t': new_fts['_t'].iloc[-1], '_p': new_fts['_c'].iloc[-1]}
            elif _side*_bin == -1. and not 'SELL' in self.signalLock:
                return {'side': 'SELL', 'positionSide': 'SHORT', '_t': new_fts['_t'].iloc[-1], '_p': new_fts['_c'].iloc[-1]}
```

BACKTESTER (CLASS)

- Combine the Market Data and list of Trades excecuted during the backtesting period
- Class properties:
  - tradeData, signalList
  - commisionRate, orderSize
- Class functions:
  - balance_update()
  - gross_profit() / gross_loss()
  - total_trades()
  - time_in_position()
  - summary()
  - …

BACKTESTER (CLASS)

- Example:



```python
class Backtester:
    def __init__(self,
                 symbol: str,
                 tradeData,
                 initBalance: float = 1000,
                 orderSize: float = 100,
                 signalList: list = [],
                 commRate = {'MARKET': 0.016/100, 'LIMIT': 0.04/100}):
        self.symbol = symbol
        self.tradeData = tradeData
        self.balancePath = pd.DataFrame([{'_t': self.tradeData['_t'].iloc[0], '_b': initBalance}])
        self.orderSize = orderSize
        self.signalList = signalList
        self.commRate = commRate
    '''

    Backtester class to monitor trading session

    symbol : str -> symbol = 'BTCUSDT'

    tradeData : pd.DataFrame(columns=['_t', '_p']

    '''
    def set_trade_data(self, tradeData):
        self.tradeData = tradeData

    def add_signal(self, signal):
        self.signalList.append(signal)
        return self.signalList

    def balance_update(self):
        if len(self.signalList)==0:
            return self.balancePath
        t_start = self.balancePath['_t'].iloc[-1]
        _trades = self.tradeData[self.tradeData['_t']>=t_start]
        for i in tqdm(range(1, _trades.shape[0]), disable=True):
            last_trade = _trades.iloc[i-1]
            new_trade =  _trades.iloc[i]
            tradeTime, change = new_trade['_t'], 0
            for sig in self.signalList:
                if last_trade['_t'] < sig.excTime and sig.excTime <= tradeTime:
                    change -= self.commRate[sig.orderType]*sig.get_quantity()*sig.excPrice
                    change += SIDE[sig.side]*sig.get_quantity()*(new_trade['_p'] - sig.excPrice)
                if last_trade['_t'] < sig.clsTime and sig.clsTime <= tradeTime:
                    change -= self.commRate[sig.cntType]*sig.get_quantity()*sig.clsPrice
                    change += SIDE[sig.side]*sig.get_quantity()*(sig.clsPrice - last_trade['_p'])
                if sig.excTime < last_trade['_t'] and tradeTime < sig.clsTime:
                    change += SIDE[sig.side]*sig.get_quantity()*(new_trade['_p'] - last_trade['_p'])
```

PYTHON TRICKS AND TIPS

#form up a pd.DataFrame from a dictionary:

df = pd.DataFrame({‘a’: 1, ‘b’: 2, ‘c’: 3, ‘d’: 4})

#point to element nth in a pd.Series:

df[<*column_name* >].iloc[n]

#operation that makes a copy of pd.DataFrame:

df = df.dropna() #.copy()

#return the index of the first element that is greater than a value:

index = df[<*column_name*>].searchsorted(<*some_value*>)

#assign a new value to an element given it’s index:

df.loc[<*some_index*> , <*column_name*>] = <*new_value*>

#return index of min/max value in a pd.Series:

idx = df [<*column_name*>].idxmin(axis=0)

CODING EXERCISES

1. Complete the code for function exit_trigger() in Signal class. This function should check for the last price in the pricePath and return whether the lastest prices hit the stopLoss or takeProfit levels.
2. In any algos, we would need to print out the infomation of our signals. The function __str__() is to return a string respresentation of the class. The first case of WAITING/EXPIRED signals is provided. Finish the rest of the code to show the following information:
   - symbol, status, side, quantity
   - orderId, orderType, excTime, excPrice (if ORDERED / ACTIVE)
   - cntorderId, cntType, clsTime, clsPrice (if CNT_ORDERED / CLOSED)
   - timeInForce (if orderType/cntType == 'LIMIT')
   - stopLoss, takeProfit, timeLimit
3. Combine the 2 'for loop' in 2.a and 2.b into a single 'while'/'for loop' that does the same backtester.
4. (Retraced stopLoss) Modify exit_trigger() to satisfy the logic:
   - if the price hits takeProfit, exit right away
   - If the price hits stopLoss, wait until the price comes back to 0.5*stopLoss, and then exit

*** Instructions:**

- modify tradingpy.py for Problem 1, 2 and 4. Run Bollinger_Band_Backtester notebook to verify the results.
- Add block(s) of code to the notebook for Problem 3