At Akuna, Python is used for a range of things, from data analysis to infrastructure, trading systems to middle office (accounting stuff). This challenge is to assess your knowledge and experience with Python. Please approach this as you would a real software project: quality, easy to understand code, not necessarily the slickest or most sophisticated solution. Don't complicate things to add flexibility to handle situations that aren't included in this challenge. Appropriate comments should be used, keeping in mind that we do not know your coding style or techniques.

The basic idea behind this challenge is that you have been given access to some sort of Python based trading system, but it is very flawed. We want to be able to pull data from this system, analyze it, determine the "edge" (profitability of our strategy) for each stock in the trading system based on this analysis and calculate the total edge that we see. To be clear, the goal of this challenge isn't to implement your own TradingSystem class, but to understand the documentation below and try to interact with it. If you would like to implement a mock TradingSystem class to test your script against, we would be happy to review that as well.

For historical reasons, our trading system interface changes every time it is run, and the data it provides is most likely far from the clean data we would like to see. Each time a TradingSystem object is instantiated, it will have some function names with a unique suffix.

To access the interface, instantiate a TradingSystem object from the trading_system package:

```
from trading_system import TradingSystem

trd_sys = TradingSystem()
```

Everything described below is a method of this TradingSystem class (trd_sys.get_days('NYSE'), etc.).

Some functions are the same every run:

**get_days(exchange)** – returns a list of python datetimes in no specific order, which might contain weekends that should be removed

**set_total_edge(value)** – lets the trading system know the total edge we saw, where value is the sum of the mean differences calculated by the objects given to set_stock_param

**get_stocks(day)** -- returns a list of (stock symbol, exchange) tuples, but this list of symbols will contain duplicates.

**set_stocks(stocks)** – where stocks is a list of symbols, tells the trading system what stock symbols you will be requesting in the future

Some function names change every run (* indicates a random suffix):

**get_stock_data_*(symbol, day)** -- where day is an integer following YYYYMMDD, returns a dictionary with keys 'open', 'high', 'low' and 'close' of the symbol on that day, as well as 'exchange'. If the requested symbol exists on multiple exchanges, this function will pick which exchange's data to return randomly. There are several potential ways to fix this (like simply silently ignoring data from the wrong exchange), but patching TradingSystem.is_valid_symbol(exchange, symbol) to return False if the exchange is not the desired one is an interesting (hint: suggested) way of solving it.

Sometimes the data is bad (low > high), and should be requested again. This bad data can be returned any number of times; if it happens five times in a row (for data from the correct exchange), the most recently provided close price should be used, even if the data is bad.

**set_stock_param_*(stock)** -- takes an object with an interface defined below

> The object passed as the stock parameter needs to have one attribute: symbol, and one method: get_edge(current_price), which calculates the differences between the current price that is passed to it, and the close price for each of the last ten days. Beyond that, it can have anything else you need or desire to solve the rest of the challenge. It then returns both the mean of the differences, and the standard deviation of the differences.

Notes:

The exchange string is passed to your script as the second command line argument (where the first is the name of the script)

In case the function name suffix can't be ascertained, 'backup' can be used (i.e., get_stock_data_backup, set_stock_param_backup), but this should be avoided if at all possible (pretend it's hilariously bad and just tries get_stock_data_0001, get_stock_data_0002,...).

Consider the call to get_stock_data* as being expensive – try to limit calls to it if possible (i.e., call it just enough times to get the correct data)

There is no actual output from this program, outside of the number you pass to set_total_edge. We are looking to see how you interact with the system.

Goal:

Using the exchange that you're given on the command line:

Find all of the stocks on the given exchange that exist on the most recent available day and pass that to set_stocks

Obtain all of the necessary data for each of these stocks to calculate the analysis described in set_stock_param

Construct objects for each stock that will calculate the edge (get_edge) given the current price of the stock by the trading system

Calculate the total edge for all analyzed stocks and pass it to set_total_edge - the solution here does not have to be pretty, but try to avoid global variables. Note there is no other way to get the current price than when get_edge is called.