Response to Feedback

I have read the book "Byte of Python" and have learnt a lot. Based on my understanding on Python, I have modified the code according to your feedback. The code is now more complete with unit testing. We can also flexibly reuse some blocks of the code in other tasks. Also, I think I can understand the way Python works better after having a look at the production standard that you guys cope with such tasks.

1. The reusability of the code. The whole code is in one big block when it should be factorized into individual components.

Response: Thanks for the advice. In the new version, I created three modules, i.e., *dlyc*, *sqlyc* and *macdyc*, which are responsible for "downloading data", "maintaining data in MySQL database" and "data analysis" respectively. Within each module, I have built functions for various types of sub-tasks. Therefore, in the *main program*, we only need to call the corresponding functions from the ready-to-use modules to complete the task! For example, we can simply use the command "cnsql=sqlyc.link()" to connect the python to MySQL database. While the default database is mine, the user can customize their own database location!

```
In [5]: #Write it to MySQL database.
        cnsql=sqlyc.link()
        sqlyc.writesymbol(symbollist,exchangemarket,cnsql)
        print('Congratulations! All available symbols in',exchangemarket.upper(),\
              'has been stored/updated in MySQL database!')

        Successfully connected to MySQL database!
        Uploading the data to MySQL database, please wait!
        Congratulations! All available symbols in AMEX has been stored/updated in MySQL database!

In [6]: ##Write stock data to MySQL database.
        cnsql=sqlyc.link()
        sqlyc.writedetail(datafile,stockname,cnsql)
        print('Congratulations! The historical data of',stockname.upper(),'has been stored/updated in MySQL database.')
```

Through separating the code into functions and individual blocks, we can easily reuse them in other tasks and this will be more effective.

2. A quirk we found in your code is the use of increments flags rather than standard for loops which we had no idea why.

Response: Thanks for indicating this issue. In this new version, I employed the standard "for … in …" loop in all situations except for a specific function.

3. Also, there are no unit tests available. Making sure that it works is not the same as having tests in place.

Response: Sorry for this. I misunderstood the meaning of testing due to lack of experience. In this new version, I have performed the standard unit tests for the functions with the standard *unittest* library. The test will further guarantee the correctness of the model. For example, the test for functions in *macdyc* module is as follows:

In [3]:
```
##From now we begin the unit testing of some blocking with unittest module.
class macdTestCase(unittest.TestCase):
    '''Tests for macdyc module.'''
    def test_MA(self):
        '''Test for the Moving Average (MA) fuction.'''
        self.assertEqual(list(macdyc.MA([1,2,3],0)),[1,2,3])
        self.assertEqual(list(macdyc.MA([2,2,2],2)),[2,2,2])
    def test_crossover(self):
        '''Test for the crossover fuction, which is used to compute diff-sig.'''
        self.assertEqual(list(macdyc.crossover([1,2,3],0,0,0)),[0,0,0])
        self.assertEqual(list(macdyc.crossover([1,1,1],2,2,2)),[0,0,0])
    def test_ratio(self):
        '''Test for the ratio fuction, which is used to indicate the change of an array!'''
        self.assertEqual(macdyc.ratio([2,2,3,4]),[1,1,1.5,2])
        self.assertEqual(macdyc.ratio([5,6,7,4]),[1,1.2,1.4,0.8])
    def test_actions(self):
        '''Test for the actions fuction, which can return the Buy-Sell sequence!'''
        self.assertEqual(macdyc.actions([0,1,2,3,2,1,-1]),['nochange','buy','nochange','nochange',\
                                                           'nochange','nochange','sell',])

    def test_valuedetail(self):
        '''Test valuedetail, which return revenue details with the given Buy-Sell sequence!'''
        action=['nochange','buy','nochange','nochange','sell']
        data=[1,1.5,3,6,1.5]
        initialvalue=1000000
        commissionrate=0
        self.assertEqual(macdyc.valuedetail(action,data,initialvalue,commissionrate)[0],[1000000,1000000,\
                                                                                         2000000,4000000,1000000])
unittest.main(argv=['first-arg-is-ignored'], exit=False)
```