[Index | Exercise 1.1 | Exercise 1.3]

Exercise 1.2

Objectives:

- Manipulate various built-in Python objects

Files Created: None

Part 1 : Numbers

Numerical calculations work about like you would expect in Python. For example:

```
>>> 3 + 4*5
23
>>> 23.45 / 1e-02
2345.0
>>>
```

Be aware that integer division is different in Python 2 and Python 3.

```
>>> 7 / 4      # In python 2, this truncates to 1
1.75
>>> 7 // 4     # Truncating division
1
>>>
```

If you want Python 3 behavior in Python 2, do this:

```
>>> from __future__ import division
>>> 7 / 4
1.75
>>> 7 // 4      # Truncating division
1
>>>
```

Numbers have a small set of methods, many of which are actually quite recent and overlooked by even experienced Python programmers. Try some of them.

```
>>> x = 1172.5
>>> x.as_integer_ratio()
(2345, 2)
>>> x.is_integer()
```

```
False
>>> y = 12345
>>> y.numerator
12345
>>> y.denominator
1
>>> y.bit_length()
14
>>>
```

Part 2 : String Manipulation

Define a string containing a series of stock ticker symbols like this:

```
>>> symbols = 'AAPL IBM MSFT YHOO SCO'
```

Now, let's experiment with different string operations:

(a) Extracting individual characters and substrings

Strings are arrays of characters. Try extracting a few characters:

```
>>> symbols[0]
'A'
>>> symbols[1]
'A'
>>> symbols[2]
'P'
>>> symbols[-1]        # Last character
'O'
>>> symbols[-2]        # 2nd from last character
'C'
>>>
```

Try taking a few slices:

```
>>> symbols[:4]
'AAPL'
>>> symbols[-3:]
'SCO'
>>> symbols[5:8]
'IBM'
>>>
```

(b) Strings as read-only objects

Strings are read-only. Verify this by trying to change the first character of `symbols` to a lower-case 'a'.

```
>>> symbols[0] = 'a'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
>>>
```

(c) String concatenation

Although string data is read-only, you can always reassign a variable to a newly created string.
Try the following statement which concatenates a new symbol "GOOG" to the end of `symbols`:

```
>>> symbols += ' GOOG'
>>> symbols
... look at the result ...
```

Now, try adding "HPQ" to the beginning of `symbols` like this:

```
>>> symbols = 'HPQ ' + symbols
>>> symbols
... look at the result ...
```

It should be noted in both of these examples, the original string `symbols` is *NOT* being modified "in place." Instead, a completely new string is created. The variable name `symbols` is just bound to the result. Afterwards, the old string is destroyed since it's not being used anymore.

(d) Membership testing (substring testing)

Experiment with the `in` operator to check for substrings. At the interactive prompt, try these operations:

```
>>> 'IBM' in symbols
True
>>> 'AA' in symbols
True
>>> 'CAT' in symbols
False
>>>
```

Make sure you understand why the check for "AA" returned `True`.

(e) String Methods

At the Python interactive prompt, try experimenting with some of the string methods.

```
>>> symbols.lower()
'hpq aapl ibm msft yhoo sco goog'
>>> symbols
'HPQ AAPL IBM MSFT YHOO SCO GOOG'
```

Remember, strings are always read-only. If you want to save the result of an operation, you need to place it in a variable:

```
>>> lowersyms = symbols.lower()
>>> lowersyms
'hpq aapl ibm msft yhoo sco goog'
>>>
```

Try some more operations:

```
>>> symbols.find('MSFT')
13
>>> symbols[13:17]
'MSFT'
>>> symbols = symbols.replace('SCO','')
>>> symbols
'HPQ AAPL IBM MSFT YHOO  GOOG'
>>>
```

Part 3 : List Manipulation

In the first part, you worked with strings containing stock symbols. For example:

```
>>> symbols = 'HPQ AAPL IBM MSFT YHOO  GOOG'
>>>
```

Define the above variable and split it into a list of names using the `split()` operation of strings:

```
>>> symlist = symbols.split()
>>> symlist
['HPQ', 'AAPL', 'IBM', 'MSFT', 'YHOO', 'GOOG' ]
>>>
```

(a) Extracting and reassigning list elements

Lists work like arrays where you can look up and modify elements by numerical index. Try a few lookups:

```
>>> symlist[0]
'HPQ'
>>> symlist[1]
'AAPL'
>>> symlist[-1]
'GOOG'
>>> symlist[-2]
'YHOO'
>>>
```

Try reassigning one of the items:

```
>>> symlist[2] = 'AIG'
>>> symlist
['HPQ', 'AAPL', 'AIG', 'MSFT', 'YHOO', 'GOOG' ]
>>>
```

(b) Looping over list items

The `for` loop works by looping over data in a sequence such as a list. Check this out by typing the following loop and watching what happens:

```
>>> for s in symlist:
        print('s =', s)

... look at the output ...
```

(c) Membership tests

Use the `in` operator to check if `'AIG'`,`'AA'`, and `'CAT'` are in the list of symbols.

```
>>> 'AIG' in symlist
True
>>> 'AA' in symlist
False
>>>
```

(d) Appending, inserting, and deleting items

Use the `append()` method to add the symbol `'RHT'` to end of `symlist`.

```
>>> symlist.append('RHT')
>>> symlist
```

```
['HPQ', 'AAPL', 'AIG', 'MSFT', 'YHOO', 'GOOG', 'RHT']
>>>
```

Use the `insert()` method to insert the symbol `'AA'` as the second item in the list.

```
>>> symlist.insert(1,'AA')
>>> symlist
['HPQ', 'AA', 'AAPL', 'AIG', 'MSFT', 'YHOO', 'GOOG', 'RHT']
>>>
```

Use the `remove()` method to remove `'MSFT'` from the list.

```
>>> symlist.remove('MSFT')
>>> symlist
['HPQ', 'AA', 'AAPL', 'AIG', 'YHOO', 'GOOG', 'RHT']
```

Try calling `remove()` again to see what happens if the item can't be found.

```
>>> symlist.remove('MSFT')
... watch what happens ...
>>>
```

Use the `index()` method to find the position of `'YHOO'` in the list.

```
>>> symlist.index('YHOO')
4
>>> symlist[4]
'YHOO'
>>>
```

(e) List sorting

Want to sort a list? Use the `sort()` method. Try it out:

```
>>> symlist.sort()
>>> symlist
['AA', 'AAPL', 'AIG', 'GOOG', 'HPQ', 'RHT', 'YHOO']
>>>
```

Want to sort in reverse? Try this:

```
>>> symlist.sort(reverse=True)
>>> symlist
['YHOO', 'RHT', 'HPQ', 'GOOG', 'AIG', 'AAPL', 'AA']
>>>
```

Note: Sorting a list modifies its contents "in-place." That is, the elements of the list are shuffled around, but no new list is created as a result.

(f) Lists of anything

Lists can contain any kind of object, including other lists (e.g., nested lists). Try this out:

```
>>> nums = [101,102,103]
>>> items = [symlist, nums]
>>> items
[['YHOO', 'RHT', 'HPQ', 'GOOG', 'AIG', 'AAPL', 'AA'], [101, 102, 103]]
```

Pay close attention to the above output. `items` is a list with two elements. Each element is list.

Try some nested list lookups:

```
>>> items[0]
['YHOO', 'RHT', 'HPQ', 'GOOG', 'AIG', 'AAPL', 'AA']
>>> items[0][1]
'RHT'
>>> items[0][1][2]
'T'
>>> items[1]
[101, 102, 103]
>>> items[1][1]
102
>>>
```

Part 4 : Dictionaries

In last few parts, you've simply worked with stock symbols. However, suppose you wanted to map stock symbols to other data such as the price? Use a dictionary:

```
>>> prices = { 'IBM': 91.1, 'GOOG': 490.1, 'AAPL':312.23 }
>>>
```

A dictionary maps keys to values. Here's how to access:

```
>>> prices['IBM']
91.1
>>> prices['IBM'] = 123.45
>>> prices['HPQ'] = 26.15
>>> prices
{'GOOG': 490.1, 'AAPL': 312.23, 'IBM': 123.45, 'HPQ': 26.15}
>>>
```

To get a list of keys, use this:

```
>>> list(prices)
['GOOG', 'AAPL', 'IBM', 'HPQ']
>>>
```

To delete a value, use `del`

```
>>> del prices['AAPL']
>>> prices
{'GOOG': 490.1, 'IBM': 123.45, 'HPQ': 26.15}
>>>
```

[Solution | Index | Exercise 1.1 | Exercise 1.3]
