Collections

Objectives

To understand the use and syntax of containers in Python 2. We will also compare different ways to access a list.

Reference Material

Chapter 5 Collections, and Chapter 3 Flow Control.

Questions

1. What is wrong with this?

```
cheese = ['Cheddar', 'Stilton', 'Cornish Yarg']
cheese += 'Oke'
```

How should `'Oke'` be added to the end of the cheese list?

2. What is going on here? Can you explain the output?

```
tup = 'Hello'
print len(tup)
```

Prints 5

```
tup = 'Hello',
print len(tup)
```

Prints 1

3. We need to do some maintenance on a list of machines:

```
machines = {'user100': 'yogi',
            'user1'  : 'booboo',
            'user2'  : 'rupert',
            'user3'  : 'teddy',
            'user4'  : 'care',
            'user5'  : 'winnie',
            'user6'  : 'sooty',
            'user7'  : 'padders',
            'user8'  : 'polar',
            'user9'  : 'grizzly',
            'user10' : 'baloo',
            'user11' : 'bungle',
            'user12' : 'fozzie',
```

```
            'user13' : 'huggy',
            'user14' : 'barnaby',
            'user15' : 'hair',
            'user16' : 'greppy'}
```

Don't type this in! It should be available for you to edit in Ex5.3.py in the `labs` directory.
Without altering the initial definition of the dictionary, write code that will implement the following changes:

a) user14 no longer has a machine assigned

b) The name of user15's machine is changed to 'cinnamon'

c) user16 is leaving the company, and a new user, user17, will be assigned his machine

d) user4, user5, and user6 are all leaving at exactly the same time, but their machine names are to be stored in a list called `unallocated`. Hint: pop in a loop.

e) user8 gets another machine called 'kodiak' in addition to the one they already have.

f) Print a list of users, each with their machine, in any order. Print each user/machine pair on a separate line.

g) Print a list of unallocated machines, sorted alphabetically.

4. This exercise will compare various ways of accessing a list. The script **Ex5.4.py** creates a list called '**`words`**' from a file of the same name. Each item in the list contains a single word. The script calls user-written functions to find the position of the word '**Zulu**' in the list, each function using a different searching technique.

   Each function is called a number of times in a loop, and timings are displayed. If you run the script right now it should display zero times - *your task is to write the search technique code*.

   Do not worry about the syntax for functions (**`def`**), we will cover that later in the course, but make sure that the **`global`** statements are intact.

   IMPORTANT: Your statements within the functions should, nay, **must**, have consistent indentation (which is four spaces).

   a) Open the script **Ex5.4.py** using a text editor. Look for the comments marked **`### TODO`**, these indicate where you are asked to add code.

      The first function to complete is **`brute_force()`**. In this function use a counting '**`for`**' loop to search sequentially for the word 'Zulu'. When found, break out of the loop and return the word's position, plus 1 (which is the line number in the file).

      Run the script to test it. If you are unsure about the result, uncomment the print statement to display the line number returned, it should be 45400. That applies to all the other parts to this exercise as well.

b) The next function to complete is rather less code. Recall the **`index()`** method which may be called on a list. It returns the position of the first item found with the given value. Call this in the **`index()`** function, returning the position plus 1. Test the script, is **`index()`** faster than brute force?

c) Just for fun (!) we will now time the `in` keyword. We cannot get the line number from this, but it will be constructive in our comparisons of search methods. Return 1 if 'Zulu' is in `words`, and 0 if it is not.

d) Here we would like you to create a dictionary called **`words_dict`** where the keys are the words, with each value being the position in the `words` list. A simple `for` loop will suffice for this. Look for:

```
### TODO: Create a dictionary called words_dict
```

Notice that the dictionary creation should be after the `start_timer()` call. Now implement the **`dictionary()`** function to return the value for the key 'Zulu', plus 1.

Which technique wins? You might like to run the script several times to get more meaningful comparisons, or increase the internal loop count (a global called `LOOP_COUNT` near the top of the script).

If time allows...

Try the timing test again but search for a word near the front of the list, for example "aback". What difference does it make?

You should see that the search mechanisms are faster, but using a dictionary should take (or or less) the same time.

Further, if time allows…

5. This exercise is a development of a previous optional exercise to show the trajectory of a projectile, ignoring air resistance. If you did not complete that exercise, take the code from `02 Fundamental variables/traj.py`.

   We are going to plot the trajectory of a projectile onto a graph. You will need two lists, one called `x_axis` and one called `y_axis`.

   Write a loop for values of `x` (horizontal distance travelled) from zero in increments of 0.5. Append each value of `x` to `x_axis`.

   Within the loop, calculate a value for `y` (height of the projectile) for each increment of `x`, and append each value of `y` to `y_axis`.

   Exit the loop when y drops to zero or lower.

   a. We are going to use the `matplotlib` module to plot the graph. You will need to ensure that `numpy` (a pre-requisite) and `matplotlib` are installed.

      Here is the code to plot the graph:

```
import matplotlib.pyplot as pyplot

pyplot.ylabel('Height m')
pyplot.xlabel('Distance m')
pyplot.plot(x_axis, y_axis)
pyplot.show()
```

b. The graph is not very realistic because the x and y units are not the same base. Take a look at the pyplot method ylim() and figure out a limit for y that will show a realistic trajectory plot.

6. This exercise gives an example of iterating through a dictionary.

   Write a program to calculate random share prices for some fictional companies and display them all continually, with a two second delay between each display.

   Here is a skeleton, as in **`Shareprices.py`**:

```
import time
import random

SharePrices = {'Global Motors':50,
               'Big Blue Inc.':50,
               'Gates Software':50,
               'Banana Computers':50}

# Update stock prices with random price changes

while True:

    # TODO: Iterate through the dictionary,
    # updating each share price (sp) to:
    # max(1.0,
    #   sp * ( 1 + ((random.random() - 0.5)/0.5) * 0.05))

    # TODO: print each company and its share price

    # Print a blank line between
    print()

    # pause for 2 seconds
    time.sleep( 2 )
```

   Printing the company name and share price is an opportunity to practice your `format` statements!

.

Solutions

1. What is wrong with this?

```
cheese = ['Cheddar', 'Stilton', 'Cornish Yarg']
cheese += 'Oke'
```

If we print the variable `cheese` we see:

```
['Cheddar', 'Stilton', 'Cornish Yarg', 'O', 'k', 'e']
```

The string 'Oke' is a sequence, and using += on a list has broken it down into its constituent parts. It should have been:

```
cheese.append('Oke')
```

2. What is going on here? Can you explain the output?

```
tup = 'Hello'
print len(tup)
```

Prints 5

That should be no surprise, 5 is the number of characters in 'Hello'.

```
tup = 'Hello',
print len(tup)
```

Prints 1

This is rather more surprising, but did you notice the trailing comma? That extra comma meant that we created a tuple. The `len()` built-in then reported the number of items in that tuple, which is one.

3. There are several solutions, here is one:

```
machines = {'user100': 'yogi',
            'user1'  : 'booboo',
            'user2'  : 'rupert',
            'user3'  : 'teddy',
            'user4'  : 'care',
            'user5'  : 'winnie',
            'user6'  : 'sooty',
            'user7'  : 'padders',
            'user8'  : 'polar',
            'user9'  : 'grizzly',
            'user10' : 'baloo',
            'user11' : 'bungle',
            'user12' : 'fozzie',
            'user13' : 'huggy',
            'user14' : 'barnaby',
            'user15' : 'hair',
            'user16' : 'greppy'}
```

a) user14 no longer has a machine assigned
`machines['user14'] = None`

b) The name of user15's machine is changed to 'cinnamon'

```
machines['user15'] = ' cinnamon'
```

c) user16 is leaving the company, and a new user, user17, will be assigned his machine

```
machines['user17'] = machines['user16']
del machines['user16']
```

d) user4, user5, and user6 are all leaving at exactly the same time, but their machine names are to be stored in a list called unallocated.

```
unallocated = []
for user in ('user4', 'user5', 'user6'):
    unallocated += [machines.pop(user)]
```

e) user8 gets another machine called 'kodiak' in addition to the one they already have.

```
machines['user8'] = [machines['user8'],'kodiak']
```

f) Print a list of all the users, with their machines, in any order.

```
for kv in machines.items():
    print kv
```

g) Print a list of unallocated machines, sorted alphabetically.

```
print "Unallocated machines:",sorted(unallocated)
```

4. These are the results we got, yours will be different because you are almost certainly running on different hardware.

```
Brute_force : 0.938 seconds
Index       : 0.203 seconds
Dictionary  : 0.016 seconds
```

Each method offers an improved access time. Remember though that using a sorted list and a dictionary requires a preamble overhead which may only be worth the effort with many searches. The figures are also data dependant, longer search keys will have an effect on performance.

Further, if time allows…

Question 5

```
from math import pi, tan, cos

g     = 9.81          # Acceleration due to gravity
                      #    m/s squared
v0    = 44            # The initial velocity m/s
theta = 80 * pi/180   # elevation angle in radians
x     = 0.5           # the horizontal distance travelled
y0    = 1             # height of the barrel (m)

# Initial values for the graphic
y = y0
x = 0.0
x_axis = []
y_axis = []

while y > 0:
    x = x + 0.1
    y = y0 + x*tan(theta) - \
        (g*x**2)/(2*((v0*cos(theta))**2))

    print('x = %.1f m, y     = %.1f m' % (x,y))
    x_axis.append(x)
    y_axis.append(y)

# Graph
import matplotlib.pyplot as pyplot

pyplot.ylabel('Height m')
pyplot.xlabel('Distance m')

# Optional realism
pyplot.ylim(-1,max(max(x_axis),max(y_axis)))

pyplot.plot(x_axis, y_axis)
pyplot.show()
```

Question 6

```
import time
import random

SharePrices = {'Global Motors':50,
               'Big Blue Inc.':50,
               'Gates Software':50,
               'Banana Computers':50}

# Update stock prices with random price changes

while True:

    for key,sp in SharePrices.items():
        SharePrices[key] = max(1.0,
                               sp * ( 1 + ((random.random() –
```

```
                                0.5)/0.5) * 0.05))
            print("{:<18s} ${:05.2f}".
                  format(key,SharePrices[key]))

        print()
        time.sleep( 2 )
```

The code is as follows:

```
...
##########################################################
# TODO: USER FUNCTIONS
# Each function should return the line number
def brute_force():
    global words
    for pos in range(0,len(words)):
        if words[pos] == 'Zulu':
            break

    # return the line number
    return pos+1

def index():
    global words
    return words.index('Zulu') + 1

def dictionary():
    global words_dict
    return words_dict['Zulu'] + 1

...

# Create a dictionary from the words list
i = 0
start_timer()

# TODO: Create a dictionary called words_dict
for key in words:
    words_dict[key] = i
    i = i + 1

for i in range(0,LOOP_COUNT):
    line = dictionary()

end_timer("Dictionary")
print "Dictionary line number:",line
line = 0
```