Carleton University
Department of Systems and Computer Engineering
SYSC 1005 - Introduction to Software Development - Fall 2013

Lab 10 - Lists, Tuples, and Sets

Attendance/Demo

To receive credit for this lab, you must make an effort to complete the exercises, and demonstrate your work.

When you have finished all the exercises, call a TA, who will review the code you wrote. For those who don't finish early, a TA will ask you to demonstrate whatever code you've completed, starting about 30 minutes before the end of the lab period. Finish any exercises that you don't complete by the end of the lab on your own time.

Reference

While working on this exercises, you may want to refer to Chapter 10 of *Think Python*:

http://greenteapress.com/thinkpython/thinkpython.pdf

This chapter describes the operators, functions and methods that work with lists.

Maintaining A Club Membership List

Suppose we want to store information about the members of a club. Individual memberships can be represented as a tuple containing three values: a string containing the name of the member, an integer representing the month in which the member joined the club (1 = January, 2 = February, etc.), and an integer representing the year in which the member joined. For example, here is the tuple for a club member who joined in February, 2011: ("Jack Harkness", 2, 2011).

The membership list can be maintained as a list of these tuples.

Exercise 1

Open a new file in Wing IDE 101, and save it with the name club.py.

Define a function named join, which adds a new member to the club's membership list. The function header is:

```
def join(members, name, month, year):
```

- `members` is the membership list (a Python list)
- `name` is the name of the member (a string)
- `month` is the month in which the member joined (an integer between 1 and 12)
- `year` is the year in which the member joined (an integer)

Remember that the information about an individual member is to be stored in a tuple.

This function does not return anything. Instead, it modifies the list bound to `members`.

If the `month` parameter is outside the valid range of 1-12, the function should not change the membership list, but should instead print a descriptive error message.

Use the shell to test your function. Create an empty membership list, and call `join` a few times to add members to the list. Each time you call `join`, verify that the new member was added to the list. For example:

```
>>> members = []
>>> join(members, "Gwen", 4, 2010)
>>> members     # the membership list will be displayed
>>> len(members)
1
```

Exercise 2

Define a function named `build_club`, which returns a membership list containing the membership information for a small club. The function header is:

```
def build_club():
```

This function must call your `join` function from Exercise 1 to add each member to the membership list. The function should return the list shown in this example:

```
>>> members = build_club()
>>> members
[("Jack", 2, 2011), ("Ianto", 2, 2009), ("Gwen", 4, 2010), ("Rhys",
5, 2008), ("Owen", 2, 2010), ("Toshiko", 3, 2011)]
```

Exercise 3

Define a function named `joined_in_month` that determines how many members joined in a given month. The function header is:

```
def joined_in_month(members, month):
```

- `members` is the membership list (a Python list)
- `month` is the month in which the member joined (an integer between 1 and 12)

This function returns the number of members who joined in the month we are interested in.

If the `month` parameter is outside the valid range of 1-12, the function should print a descriptive error message and return -1.

Use the shell to test your function. Use the membership list returned by your `build_club` function. You'll need several test cases; e.g., develop tests in which `joined_in_month` returns 0 (no members joined in the specified month), 1 (one member joined in the specified month), and a value greater than 1 (multiple members joined in the specified month). You'll also need test cases to

demonstrate the your function correctly handles out-of-range values for the `month` parameter.

Exercise 4

For this exercise, use the Online Python Tutor, not Wing IDE 101.

Point your web browser to the OPT Web site:

http://www.pythontutor.com/visualize.html

Delete the example code in the OPT editor.

Define a function named `remove_selected` that has two parameters: a list of integers, `lst`, and an integer value, `val`. The function modifies `lst` by removing all occurrences of `val`. The function does not return or print anything; in other words, it does not require any `return` or `print` statements. The function header is:

```
def remove_selected(lst, val):
```

Don't use a `for` loop to iterate over the list; e.g.,

```
for num in lst:
     # do something with num
```

When a `for` loop iterates over the elements in a list, code in the loop body shouldn't cause the list to grow (by inserting objects into the list) or shrink (by removing objects from the list). If the loop body modifies the list, integers in the list may be skipped over (never bound to `num`) or the same integer may be bound to `num` more than once. This means that the loop may work for some lists, but not for others.

Instead of a `for` loop, use a `while` loop.

Hint: there is no need for this function to copy integers between lst and a secondary list created by the function. The list bound to parameter lst is the only one the function requires.

Try these test cases (type these statements below your function definition, then click the Visualize Execution button and step through the execution of your function):

```
lst = []
remove_selected(lst, 3)
# lst should be bound to an empty list

lst = [3]
remove_selected(lst, 3)
# lst should now be bound to an empty list

lst = [5, 3, 7]
remove_selected(lst, 3)
```

```
# lst should now be bound to the list [5, 7]

lst = [5, 7, 9, 10]
remove_selected(lst, 3)
# lst should be bound to the list [5, 7, 9, 10]

lst = [3, 5, 3, 3, 7, 9, 3]    # Check your results carefully for
remove_selected(lst, 3)        # this test case
# lst should now be bound to the list [5, 7, 9]
```

Exercise 5

In club.py, define a function named `purge` that removes from the club's membership list the membership information for all members who joined in a given month, and returns this information in a separate list. The function header is:

```
def purge(members, month):
```

- `members` is the membership list (a Python list)
- `month` is the month in which the member joined (an integer between 1 and 12)

The function returns a new list containing the membership information for every member who joined in the given month; that is, all the tuples that were removed from `members`.

If the `month` parameter is outside the valid range of 1-12, the function should print a descriptive error message and return an empty list.

Hint: this function is similar to the `remove_selected` function you wrote for Exercise 4, in that both functions must modify a list while iterating over it. The `purge` function will modify the membership list (remove tuples, causing the remaining tuples to "shift to the left") while it traverses the list. Remember to take this into account while designing your function.

Use the shell to test your function. Use the membership list returned by your `build_club` function. You'll need several test cases; e.g., develop cases where purge returns an empty list (no members joined in the specified month), a list of size 1 (one member joined in the specified month), and a list whose size is greater than 1 (several members joined in the specified month). For each test case, verify that membership list passed to `purge` is modified correctly. You'll also need test cases to demonstrate the your function correctly handles out-of-range values for the `month` parameter.

Wrap-up

1. Remember to have a TA review your club.py file and give you a grade (Satisfactory, Marginal or Unsatisfactory) before you leave the lab.
2. Remember to save a copy of your club.py file (copy it to a flash drive, or email a copy to yourself, or store it on your M: drive - remember, files left on your desktop or in your Documents folder are not guaranteed to be there the next time you log in).
3. Log in to cuLearn, click the Submit Lab 10 link and submit club.py. After you click the Add submission button, drag the file to the File submissions box. After the icon for the file appears in the box, click the Save changes button. At this point, the submission status for

your files is "Draft (not submitted)". You can resubmit the file by clicking the Edit my submission button. After you've finished uploading the file, remember to click the Submit assignment button to change the submission status to "Submitted for grading".

Challenge Exercise

My nephew's math textbook contains this exercise, which stumped the parents of his Grade 4 math class:

> Letters e, n, o, s, u, y represent integers between 0 and 9, inclusive. Each letter represents a different integer; for example, if e represents 2, then n cannot be 2. Determine all values of e, n, o, s, u, and y that satisfy the sum:
>
> ```
>   see
> + you
> ----
> soon
> ```

A bit of analysis revealed that this problem has more than a few solutions, so I decided to write a Python program to crank out the numbers.

Challenge: write a Python program that calculates all integers e, n, o, s, u, y such that `see + you` equals `soon`. Remember, each letter represents a different integer between 0 and 9. Print the these numbers using the format `see` + `you` = `soon`, sorted in ascending order. The output should look like this:

```
99 + 124 = 223
99 + 125 = 224
99 + 126 = 225
    ...
199 + 803 = 1002
    ...
199 + 807 = 1006
```

This problem can be solved with less than 30 lines of code, if you use lists, sets and tuples. I'll post my solution later in the term.