CSE 1321L: Programming and Problem Solving I Lab

Lab 9

Sequence Types (Part 1)

What students will learn

- Sequence types.
- Creating lists and tuples.
- Adding and removing elements from lists.
- Accessing a particular element in a sequence type.

Content

Overview

One of the data types available in Python is the Sequence Type. Unlike integers or floats, Sequence Types can **hold more than one "element"; or no element at all**. We have been using one such Sequence Type thus far: the **string**. Strings can hold any number of characters, from 0 to many characters. We can access a particular character in a string using the square brackets. Strings are 0-indexed, meaning that the first element in the string is in **position 0**.

```
name = "Alice"

# prints "A"
print(name[0])
```

Another very helpful sequence type is the List. Lists can also store any number of elements, from 0 to many. Unlike strings, lists can store any data type, including other lists:

```
# declares an empty list
mylist = []

# adds "Alice" to the end of the list
mylist.append("Alice")
mylist.append(30)
mylist.append(12.3)
mylist.append(True)

# prints the list, which will appear as ["Alice", 30, 12.3, True]
print(mylist)
```

Much like strings, we can access a particular element in a list using the square brackets. Lists are also 0-indexed.

```
print(mylist[2]) # prints 12.3
```

Unlike with strings, we can remove elements from the list, by either deleting at a particular index on the list or by using the remove() method.

```
del mylist[0] # Removes “Alice”

# Removes the first True entry in the list
mylist.remove(True)

# This will crash your program, as there is no “Bob” in the list!
mylist.remove(“Bob”)
```

Finally, we have the sequence type called Tuple. Tuples are created using parenthesis and can also store any data type like lists. Elements in a tuple are also accessed using square brackets. **Unlike lists, however, tuples cannot be edited**: once they are created, a tuple is permanent.

```
mytuple = (“Alice”, 30, 12.3, True)

# This will crash your program
del mytuple[0]
```

Like lists, tuples can also store other lists or tuples.

As with previous weeks, all labs should have the appropriate file names:

- Lab9A.py
- Lab9B.py

Lastly, make sure you review the sample output and make sure the output of your program follows the exact same format including the input statements, print statement, etc. As always, user input is shown in **red** and **bold**.

Lab9A: Sign me up

You might have been enrolled into a mailing list at some point in your life. While some mailing lists are very helpful with the emails they send, some are simply used to send ads or spam. We'll code a very simple mailing list which allows the adding or removal of emails to it.

Requirements:

- **This program must feature a list** which remembers every entry that is added to it.
- The program should also feature a loop with the prompts for the following options:
  - Add email
  - Delete email
  - List all emails
  - Quit
- The user should select the appropriate option from the list using an integer input.
- For adding an email into the list, the program should prompt the user for an email address and then add it into the list.
  - There is no need to verify that the input value is an actual email address.
- For deleting an email, the program should prompt for the to-be deleted email address and verify it exists in the email list. If the email exists in the list the program should remove it, if not output an error message.
- For listing all the emails, the program should output and list all the emails in the email list,
- For quitting, the program should terminate the loop.

Sample output #1:

```
[Mailing List]

1 - Add email
2 - Delete email
3 - List all emails
4 - Quit
Make your selection: 1

Enter the email to be added: alice@email.com
Email added to mailing list.

1 - Add email
2 - Delete email
3 - List all emails
4 - Quit
Make your selection: 1

Enter the email to be added: bob@email.com
Email added to mailing list.

1 - Add email
2 - Delete email
3 - List all emails
4 - Quit
```

```
Make your selection: 1

Enter the email to be added: charlie@email.com
Email added to mailing list.

1 - Add email
2 - Delete email
3 - List all emails
4 - Quit
Make your selection: 2

Enter the email to be removed: david@email.com
No such email in mailing list: david@email.com

1 - Add email
2 - Delete email
3 - List all emails
4 - Quit
Make your selection: 2

Enter the email to be removed: bob@email.com
bob@email.com has been removed from the mailing list.

1 - Add email
2 - Delete email
3 - List all emails
4 - Quit
Make your selection: 3

alice@email.com
charlie@email.com

1 - Add email
2 - Delete email
3 - List all emails
4 - Quit
Make your selection: 4

Shutting down...
```

Lab9B: Remember me

We'll write a simple application which allows for the storage of the names and ages of the user's friends. A person's information will be stored in a tuple, with the first index holding their name and the second index holding their age (you may store their age as strings). These tuples will be stored in a list. As such, if we stored "Alice" with the age of "30", "Bob" with the age of "40", and "Charlie" with the age of "50" in our list, and then tried to print out the list, we would get the following:

```
[("Alice", "30"), ("Bob", "40"), ("Charlie", "50")]
```

Requirements:

- **The program must feature a List which will contain Tuples**.
- The Tuples should contain two values, a Name and an Age.
    - This is an example of how the list:
      `[ ("Alice", "30"), ("Bob", "40"), ("Charlie", "50") ]`
- The program should also feature a loop with the prompts for the following options:
    - Add friend
    - List friends
    - Quit
- The user should select the appropriate option from the list using an integer input.
- For adding a friend, the program should prompt the user for the name and age of the friend then add these values as a single tuple into the list.
- For listing friends, the program should output and list all the friends using the following format:
  `` `"Name: {NAME}, Age: {AGE}"` ``
- For quitting, the program should terminate the loop

Sample Output #1:

```
[Friend List]

1 - Add friend
2 - List friends
3 - Quit
Make your selection: 1

Enter your friend's name: Alice
Enter your friend's age: 30
Friend added

1 - Add friend
2 - List friends
3 - Quit
Make your selection: 1

Enter your friend's name: Bob
Enter your friend's age: 40
Friend added

1 - Add friend
```

```
2 - List friends
3 - Quit
Make your selection: 1

Enter your friend's name: Charlie
Enter your friend's age: 50
Friend added

1 - Add friend
2 - List friends
3 - Quit
Make your selection: 2

Name: Alice, Age: 30
Name: Bob, Age: 40
Name: Charlie, Age: 50

1 - Add friend
2 - List friends
3 - Quit
Make your selection: 3

Shutting down...
```

Submission Instructions:

- Programs must follow the output format provided. This includes each blank line, colons (:), and other symbols.
- Programs must be working correctly.
- Programs must be written in Python.
- Programs must be submitted with the correct **.py** format.
- Programs must be saved in files with the correct file name:
  - Lab9A.py
  - Lab9B.py
- Programs (source code files) must be uploaded to Gradescope by the due date.