CSE 1321L: Programming and Problem Solving I Lab

Lab 10

Sequence Types

What students will learn:

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

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’ve 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”
print(name[0]) # prints “A”

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**:

mylist = [] # declares an empty list
mylist.append(“Alice”) # adds “Alice” to the end of the list
mylist.append(30)
mylist.append(12.3)
mylist.append(True)
print(mylist) # prints the list, which will appear as [“Alice”, 30, 12.3, True]

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

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 created, a tuple is permanent.

mytuple = (“Alice”, 30, 12.3, True)
del mytuple[0] # This will crash your program

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

**Lab10A**: 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.

Note: You do not need to verify that the information being entered is a valid email address.

Your program will feature a list which remembers every entry that is added to it. It will also contain a loop with the following options:

* Add entry: prompts the user for an email address, and then adds it to the list.
* Remove entry: prompts the user for an email address. If the email exists in the list, remove it. Otherwise, print out an error message.
* List entries: List all emails in the mailing list.
* Quit: Quits the program.

Sample output (user input in **bold**):

[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...

**Lab10B:** 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”)]

Notice that our list of friends has 3 elements, all of which are tuples. Each tuple has two elements, with the first one being the person’s name and the second one being their age.

Write a loop which prompts the user to add a friend, list all friends, or quit the program.

* Adding a friend prompts the user for their name and age, and stores the resulting tuple in the list.
* List all friends prints all friends in the following format, one friend per line:
	+ Name: NAME, Age: AGE
	+ Thus, our example list above would print out the following:

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

* If the user selects “quit”, stop the program.

Sample output (user input in **bold**):

[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...