**COSC 1336, Lab 9 Instructions, Dictionaries, Sets and Pickling**

**Dictionary and Set**

The more tools available, the less work required to complete a task. With a list of lists, you can create pairs of abbreviations with their meaning. To decipher an abbreviation, you can look up the first item to find the second, like this:

**abbrev=[ ['NRG', 'Northridge'], ['AUS', 'Austin'], ['TX', 'Texas'], ['US', 'United States'] ]  
target=input('Enter the abbreviation, I’ll tell you what it stands for: ')**

**for pair in abbrev:  
 if pair[0]==target:  
 print(pair[1])**

This works, but there is a better way. It is easier to use dictionaries, like this:

**abbrev={'NRG':'Northridge', 'AUS':'Austin', 'TX':'Texas', 'US':'United States'}  
target=input('Enter the abbreviation, I’ll tell you what it stands for: ')  
print(abbrev.get(target)) # or print(abbrev[target])**

Consider another situation. Say you need a collection of one of each (unique) letter used in a sentence:

**sentence='this is a fine day to go to the park and have a picnic with lemons and zebras'  
letters=[ ]  
for letter in sentence:  
 if not letter in letters:  
 letters+=letter  
print(letters)**  
  
This works, but it is easier to create a set:

**sentence='this is a fine day to go to the park and have a picnic with lemons and zebras'  
letters=set(sentence)  
print(letters)**

Different objects (containers) have various advantages and disadvantages. Many originated in mathematics (sets and set theory) and migrated into programming languages, where they could be useful.

By using **dictionary** and **set**, instead of list, you can write smaller, faster, better code.

**Pickling**

Here is another powerful tool: **pickle**. Say you want to save the list (or dictionary) ***abbrev*** and the string ***sentence*** to a file; and then later, read that data back. How would you do this? To save these objects to a text file and then restore is awkward. Saving many different objects: list, dictionary, set, string, etc. to a text file and reloading them is awkward, because every item must be converted to and from a string. Using pickle to “dump” and “load” objects, it’s easy!

This lab has three parts. The first two parts try out dictionaries and sets. You don’t have to learn every operation these provide; but you should learn they can be better than list, tuple and string. The third part is to try out pickle operations.

**Part 1: (10 points)** Write a program that uses a dictionary. The dictionary will be similar to the **abbrev** dictionary mentioned above. Each key is a two-letter abbreviation for a state; each value is the full state name. Create a dictionary; fill it with data from the same file used in lab 6: **StateCensus2010.txt**. Read full name and abbreviation for each of the 50 states into the dictionary. Ignore the population data. With the dictionary loaded, enter a loop to ask the user for the state abbreviation, and display the full state name from the dictionary. The main loop will look similar to the dictionary lookup loop above. Allow any case for the two letter abbreviation entered by the user. (TX, Tx, tx and tX should find Texas). To do this, convert the letters entered to uppercase-before the dictionary lookup. Do not let the lookup crash if the letters are not in the dictionary. Instead, say: “not found”. Loop until the user enters Q or q to quit. If desired, save this temporarily to: **DDHH\_L9\_Lastname\_dictionary.py**

**Part 2: (10 points)** For this part, make several sets, then perform some operations on these sets to make other sets. Display the derived sets.

Start by making the following “startup” sets or strings:

* **alpha** set of all lowercase letters: a..z
* **digit** set of all digits: 0..9
* **even** set of even digits: 0,2,4,6,8
* **vowel** set of all lowercase vowels: aeiou
* **punct** set of punctuation: ~`!@#$%^&\*()\_-+={[}]|\:;”’<,>.?/
* **match** set of matching punctuation characters: {}[]()<>
* **advice** string: “treat others kindly”
* **address** string:“11928 stonehollow dr., austin, tx (us [of] a)”

Given these sets, use set operations to compute and display the following:

* **consonants** set of all alpha that are **not** in vowel
* **odd** set of digits that are **not** even
* **advice\_consonant** set of consonants found in advice
* **odd\_address** set of odd digits in address
* **punct\_address** set of punctuation in address
* **no\_match** set of punctuation in address which is not in match

To save you time, code is provided to make the startup sets.

Save the If desired, save this temporarily to: **DDHH\_L9\_Lastname\_set.py**

**Part 3: (10 points)** Try pickle. Write code to load all the objects from a .dat file, and display the objects to the screen. Provided is the file: **secret.dat**. This file was created using pickle.dump(object, outfile) statements. See chapter 9, Program 9-4, page 476 (4ed); page 407 (3ed); chapter 10, program 10-4, page 409 (2ed) for code that writes data (using **pickle.dump()**) to a file. To read data (using **pickel.load()**) from a pickled file, see the next program in the textbook: 9-5 unpickle\_objects.py (3ed & 4ed); 10-5 (2ed). Your program will look like unpickle\_objects.py, but, instead of calling display\_data(person), you can just print the object. You don’t know in advance what the objects are, or how many objects are in the file.   
If desired, save this temporarily to: **DDHH\_L9\_Lastname\_pickle.py**

**Extra credit:** Put the code for each part in a separate function. In main, provide a menu-driven loop. Allow the user the select and call either: part1, part2, part3 or quit.

**Summary: Combine all three parts into one file. Submit the following file for this lab: DDHH\_L9\_Lastname.py**