**Lab: List, Dictionaries and Files**

Work with your group – by now you have gone through the powerpoint – done the activities and done the List, Tuples and Dictionary Lab. Now work on these problems, paste your code here, and submit to the Lab List, Dictionaries and Files dropbox at the end of class, you should submit the work you have done, whether you have finished or not.

**Problem 1**

Use the visualizer: <http://www.pythontutor.com/visualize.html#mode=edit>

Predict what will happen. Then add this code to the visualizer and step through it:

for i in range(10, 13):

for j in range(1, 5):

print(i, j)

**Problem 2**

Let

villains = ["Voldemort", "Darth Vader", "Sauron"]

What type (or class) of data structure is this? If necessary ask Python >>>Type (object)

(Is it mutable or immutable – and what does that mean again?) List; mutable; means it can be changed

find the length of villains? 3

Use a method (or two) to add ‘The White Witch’, ‘Cruella Deville’ and ‘Maleficent’ to the list. Villains.append(), and villains.insert(index, object)

find the length of villains again? 6

Check if "Darth Vader" **in** villains: True

Also check if Lex Luthor is **in** list? What is returned? If not add him. False

Did you edit or change villains? Does the original list still exist? Explain. Changed villains[], the original does not exist anymore. Lists are mutable and we added to an already existing list.

Which method can be used to preserve the original list? At what point of the process should you use this step? Villains.copy(), at the beginning before any changes to villains were made.

Use a for loop to print each item on a line

>>> for item in villains:

print (item)

slice the list to get the female villians in 4 ways

>>> villains[:1]

['Cruella Deville']

**>>> villains[-2:-1]**

**['The White Witch']**

**>>> villains[3:-3]**

**['Maleficet']**

**>>> villains[3:4]**

**['Maleficet']**

**Problem 3**

Lists can contain items of any type, including other lists. These are called *nested lists*.

Here is an example.

grades = [['Assignment 1', 80], ['Assignment 2', 90], ['Assignment 3', 70]]

**Find**

>>> grades[0]

['Assignment 1', 80]

>>> grades[1]

[‘Assignment 2’, 90]

>>> grades[2]

[‘Assignment 3’, 70]

To access a nested item, first select the sublist, and then treat the result as a regular list.

For example, to access 'Assignment 1', we can first get the sublist and then use it as we would a regular list:

sublist = grades[0]

sublist

['Assignment 1', 80]

**Find**

>>> sublist[0]

'Assignment 1'

>>> sublist[1]

80

Both sublist and grades[0] contain the memory address of the ['Assignment 1', 80] nested list.

We can access the items inside the nested lists like this:

>>> grades[0][0]

'Assignment 1'

>>> grades[0][1]

80

**Find**

>>> grades[1][0]

‘Assignment 2’

>>> grades[1][1]

90

>>> grades[2][0]

‘Assignment 3’

>>> grades[2][1]

70

**Problem 4**

Write a function calculate\_averages, in this function the outer for loop iterates through each sublist in grades. We then calculate the average of that sublist using a nested, or inner, loop, and add the average to the accumulator (the new list, averages). Write a main() function that calls and prints the results of calculate\_averages for an list of scores.

calculate\_average description and example:

Return a new list in which each item is the average of the grades in the

inner list at the corresponding position of grades.

>>> calculate\_averages([[70, 75, 80], [70, 80, 90, 100], [80, 100]])

[75.0, 85.0, 90.0]

def calculate\_averages(grades):

averages = []

for obj in grades:

count = 0

sum = 0

for itm in obj:

sum += itm

count += 1

averages.append(sum/count)

return averages

def main():

calculate\_average([[70, 75, 80], [70, 80, 90, 100], [80, 100]])

print(averages)

main()