**Lab 7 - Copying, Tuples, List Comprehension and Dictionaries**

**Questions:**

**1. Show the output for the following program. To make it easier, draw a diagram similar to the ones in lecture 9 pdf.**

**str\_list = ['hi', 'mom', 'dad', ['grandma', 'grandpa']]**

**new\_list = str\_list**

**copy\_list = str\_list[:]**

**str\_list[0] = 'bye'**

**new\_list[1] = 'mother'**

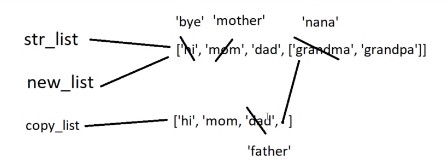
**copy\_list[2] = 'father'**

**copy\_list[-1][0] = 'nanna'**

**print(str\_list) *# Line 1***

**print(new\_list) *# Line 2***

**print(copy\_list) *# Line 3***



**(a) What output is produced by Line 1 when the program is executed?**

['bye', 'mother', 'dad', ['nanna', 'grandpa']]

**(b) What output is produced by Line 2 when the program is executed?**

['bye', 'mother', 'dad', ['nanna', 'grandpa']]

**(c) What output is produced by Line 3 when the program is executed?**

['hi', 'mom', 'father', ['nanna', 'grandpa']]

**2. Consider:**

ListA = [1, 2, 3, 4, 5]

ListB = ListA

ListA[2] = 10

**What is the value of ListB[2]?**

ListA = [1, 2, 3, 4, 5]

ListB = ListA

ListA[2] = 10

print(ListB[2])

**Result:** 10

**3. Consider the following code:**

list1 = [1, 2, 99]

list2 = list1

list3 = list2

list1 = list1.remove(1)

print(list3)

**(a) What is printed?**

[2, 99]

**(b) How can you change the code so list3 is unchanged?**

list1 = [1, 2, 99]

list2 = list1

list3 = list2

list1 = list1.remove(1)

list3 = tuple(list3)

**4. Given a list L = [1,2,3,4], we want to convert the list to the string '1234'. We tried ''.join([i for i in L]), but it didn't work. Fix it.**

L = [1, 2, 3, 4]  
''.join([str(i) for i in L])

**OR**

L = [1, 2, 3, 4]

res = ""

for i in range(len(L)):

res += str(L[i])

**5. Fractions: You can express a fraction as a tuple: (numerator, denominator).**

**(a) Write a function that adds two fractions that are passed as tuples.**

**(b) Write a function that multiplies two fractions that are passed as tuples.**

import math  
  
  
def add\_fraction(operand\_1, operand\_2):  
  
 lcm\_int = find\_lcm(operand\_1[1], operand\_2[1])  
  
 numerator\_1\_int = operand\_1[0] \* (lcm\_int // operand\_1[1])  
 numerator\_2\_int = operand\_2[0] \* (lcm\_int // operand\_2[1])  
  
 fraction\_sum = numerator\_1\_int + numerator\_2\_int, lcm\_int  
  
 gcd = math.gcd(fraction\_sum[0], fraction\_sum[1])  
 return fraction\_sum[0] // gcd, fraction\_sum[1] // gcd  
  
  
def find\_lcm(first\_int, second\_int):  
  
 gcd = math.gcd(first\_int, second\_int)  
 return first\_int // gcd \* second\_int // gcd \* gcd  
  
  
def mult\_fraction(operand\_1, operand\_2):  
  
 numerator = operand\_1[0] \* operand\_2[0]  
  
 denominator = operand\_1[1] \* operand\_2[1]  
  
 factor = math.gcd(numerator, denominator)  
 return numerator // factor, denominator // factor

**6. Write a Python program to sort a tuple by its float element.**

**Sample data: [('item1', '12.20'), ('item2', '15.10'), ('item3', '24.5')]**

**Expected Output: [('item3', '24.5'), ('item2', '15.10'), ('item1', '12.20')]**

price = [('item1', '12.20'), ('item2', '15.10'), ('item3', '24.5')]

print(sorted(price, key=lambda x: float(x[1]), reverse=True))

**7. Using list comprehension**

**(a) Generate a list of square numbers**

**(b) Convert a list of colors = ['Red', 'Blue', 'Green', 'Black', 'White'] to upper case  
(c) Find all of the numbers from 1-1000 that are divisible by 7**

**(d) Find all of the numbers from 1-1000 that have a 3 in them**

**(d) Count the number of spaces in a string**

**(e) Remove all of the vowels in a string**

**(f) Find all of the words in a string that are less than 4 letters**

**(g) Challenge! Use a nested list comprehension to find all of the numbers from 1-1000 that are divisible by any single digit besides 1 (2-9). The first part is given below. You need to find out the second list comprehension**

[number **for** number **in** range(1, 1001) **if True in** [second list comprehension]]

(a) Generate a list of square numbers:

squares = [x\*\*2 for x in range(10)]

print(squares) # prints [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

(b) Convert a list of colors = ['Red', 'Blue', 'Green', 'Black', 'White'] to upper case:

colors = ['Red', 'Blue', 'Green', 'Black', 'White']

upper\_colors = [color.upper() for color in colors]

print(upper\_colors) # prints ['RED', 'BLUE', 'GREEN', 'BLACK', 'WHITE']

(c) Find all of the numbers from 1-1000 that are divisible by 7:

divisible\_by\_7 = [x for x in range(1, 1001) if x % 7 == 0]

print(divisible\_by\_7) # prints [7, 14, 21, ..., 986, 993, 1000]

(d) Find all of the numbers from 1-1000 that have a 3 in them:

numbers\_with\_3 = [x for x in range(1, 1001) if '3' in str(x)]

print(numbers\_with\_3) # prints [3, 13, 23, ..., 983, 993]

(e) Count the number of spaces in a string:

my\_string = "The quick brown fox jumps over the lazy dog"

num\_spaces = sum([1 for char in my\_string if char == ' '])

print(num\_spaces) # prints 8

(f) Remove all of the vowels in a string:

my\_string = "Hello, World!"

no\_vowels = ''.join([char for char in my\_string if char not in 'aeiouAEIOU'])

print(no\_vowels) # prints "Hll, Wrld!"

(g) Find all of the words in a string that are less than 4 letters:

my\_string = "The quick brown fox jumps over the lazy dog"

words\_less\_than\_4 = [word for word in my\_string.split() if len(word) < 4]

print(words\_less\_than\_4) # prints ['The', 'fox', 'over', 'the', 'dog']

(h) Challenge! Use a nested list comprehension to find all of the numbers from 1-1000 that are divisible by any single digit besides 1 (2-9):

divisible\_by\_single\_digit = [x for x in range(1, 1001) if any([x % i == 0 for i in range(2, 10)])]

print(divisible\_by\_single\_digit) # prints [2, 3, 4, 5, 6, 7, 8, 9, 10, 12, ..., 989, 990, 992, 994, 996, 998, 999, 1000]

**8.** **If my\_dict = {'a':15 , 'c':35, 'b':20}, write Python code:**

**(a) to print all the keys.  
(b) to print all the values.  
(c) to print all the keys and values pairs.  
(d) to print all the keys and values pairs in order of key.  
(e) to print all the keys and values pairs in order of value.**

my\_dict = {'a': 15, 'c': 35, 'b': 20}

for k, v in my\_dict.items():

print(k, v)

print(sorted(my\_dict.items(), key=lambda x: x[1]))

print(sorted(my\_dict.items(), key=lambda x: x[1], reverse=True))

**9. Write a Python program to combine these two dictionaries adding values for  
common keys.**

**d1 = {'a': 100, 'b': 200, 'c':300}  
d2 = {'a': 300, 'b': 200, 'd':400}**

**Sample output: d3 = {'a': 400, 'b': 400, 'd': 400, 'c': 300}**

d1 = {'a': 100, 'b': 200, 'c': 300}  
d2 = {'a': 300, 'b': 200, 'd': 400}  
  
d3 = dict(d1)  
  
d3.update(d2)  
  
for i, j in d1.items():  
  
 for x, y in d2.items():  
  
 if i == x:  
 d3[i] = (j+y)  
  
print(d3)

**Or**

from collections import Counter

d1 = {'a': 100, 'b': 200, 'c':300}

d2 = {'a': 300, 'b': 200, 'd':400}

d = Counter(d1) + Counter(d2)

print(d)

**10. Make a word cloud. A word cloud is a visual representation for text data typically used to depict keyword metadata on websites, or to visualize free form text. An example is given below:**

Text

Description automatically generated

**The text of the HTML page is provided below:**

**<!DOCTYPE html>  
<html>  
<head lang="en">  
<meta charset="UTF-8">  
<title>Tag Cloud Generator</title>  
</head>  
<body>  
<div style="text-align: center; width: 15%; vertical-align: middle; font-family: arial; color: white; background-color:black; border:1px solid black">  
\*\* Your SPAN elements should be inserted here \*\*  
</div>  
</body>  
</html>**

**The format of a span element is**

**<span style="font-size: XXpx"> WORD </span>**

**where XX is the size in pixels and WORD is the word being represented.  
So for example, <span style="font-size: 20px"> our </span>**

1. **The first part will be to read the story from the hare\_and\_tortoise.txt file in brightspace and populate a dictionary using each word as a key, and value equals the frequency of the word. Your dictionary may look something like this:**

**{'tortoise': 5, 'hare': 4, 'stopped': 2, …}**

1. **For each word in the dictionary of frequencies you’ll need to write a SPAN tag to your HTML file. The font size will vary depending on the frequency of the word – for example if you use count\*10 to calculate the words’ size, then words that appear once will be size 10px, words that appear twice 20px, etc.**
2. **You should then create the completed HTML page and write it as a .html file. To manipulate html files just use the .html extension to write/read a file instead of the usual .txt. For instance: fo = open("output.html", "w"). You can open a html file to test it using a browser of your choice.**
3. **Modify your program to exclude common words from showing in the cloud. You can get a file of stopwords (e.g. a, the, this, there, etc) in brightspace.**

import string

file = open("hare\_and\_tortoise.txt", "r")

stop\_words\_file = open("stopwords.txt", "r")

stop\_words = []

for line in stop\_words\_file:

# Add only lower case stop words without punctuation and new lines.

stop\_words.append(line.strip(string.punctuation + "\n").lower())

# Dictionary of word:key pairs

word\_frequency = {}

for line in file:

words = line.split()

for word in words:

word = word.strip(string.punctuation).lower()

# Don't add stop words

if word in stop\_words:

continue

# Add word in the dictionary with frequency 1 or update frequency

if word not in word\_frequency:

word\_frequency[word] = 1

else:

word\_frequency[word] += 1

# html string to write in the html file

html = '<!DOCTYPE html><html><head lang="en"><meta charset="UTF-8"><title>Tag Cloud Generator</title></head><body><div style="text-align: center; width: 15%; vertical-align: middle; font-family: arial; color: white; background-color:black; border:1px solid black">'

count = 0 # Count to add new line every 4 words

for word, frequency in word\_frequency.items():

html += '<span style="font-size: {}px">{}</span>'.format(frequency\*10, word)

count += 1

if count % 4 == 0:

html += '<br>'

html += '</div></body></html>'

file = open("word\_cloud.html", "w")

file.write(html)

file.close()