|  |  |  |
| --- | --- | --- |
| **College of Engineering**  Computer Science & Eng. Dept.  **Course:** CMP 321L Programminglanguages Lab | A picture containing logo  Description automatically generated | **Course Professor:** Dr. Michel Pasquier  **Lab Instructor:** Praveena Kolli  **Office:** EB2-126  **Phone**: 971-6-5152352  **e-mail**: pkolli@aus.edu  **Semester**: Summer 2022 |

**Lab 2 – Python Data Structures**

**Objectives:**

* Understand and practice with lists, tuples, and dictionaries.

**Due date: End of the lab. (**Only one team member needs to submit.)

**Rules:**

(1) Usage: **You should explore and make good use of the Python features you learned in class.** (2) Scope: **You should only use those features that have been explained in detail in class.**

(3) Style: Follow standard Python programming style and conventions.

(4) Logic: Add appropriate comments to your code to explain your solution.

(Code / answers that do not follow the above specifications will be penalized.)

***Warning:* You need to come to the lab properly prepared i.e.**

(1) Make sure you have studied and understood the class material.

(2) Read the lab doc, think about the problems, and prepare questions as needed.

If you do not, completing the lab in 2.45 hours may become too much of a challenge!

**Useful resources:**

* <https://docs.python.org/3/tutorial/datastructures.html>

**Exercise 1: Tuples and printing format [2 Marks]**

Write a script that defines a list of (5 or more) 3D points, where the coordinates of each point are stored in a tuple. The script should then:

* Remove the last point
* Insert a new point before the last one in the list (do not assume the length is known)
* Print the points in the following format (where all columns are right-aligned):

|  |  |  |
| --- | --- | --- |
| X | Y | Z |
| 10 | 8 | 12 |
| 111 | 64 | 45 |
| … | … | … |

**Code:**

points = [(1,2,3) , (4,5,6) , (7,8,9) , (10, 11, 12), (13,14,15)]

points.pop() #Last element is now (10,11,12)

points.insert(-1, (17,18,19)) #Inserts before element with index -1 (before last element)

print ("{:>4} {:>4} {:>4}".format('X', 'Y', 'Z'))

for i in points:

print ("{:>4} {:>4} {:>4}".format(str(i[0]), str(i[1]), str(i[2])))

**Screenshot:**

**A picture containing text, device, meter

Description automatically generated**

**Exercise 2: Dictionary [2 Marks]**

Write a script that defines a string containing the following text:

The best programs are written so that computing machines can perform them quickly and so that human beings can understand them clearly. A programmer is ideally an essayist who works with traditional aesthetic and literary forms as well as mathematical concepts, to communicate the way that an algorithm works and to convince a reader that the results will be correct...

The script uses a dictionary to count letter frequencies, where the key is a letter and the value is the number of times it appears. Your script will have to traverse the text character by character to update the dictionary appropriately. Once done it should print the letter frequencies.

**Code:**

text = "The best programs are written so that computing machines can perform them quickly and so that human beings can understand them clearly. A programmer is ideally an essayist who works with traditional aesthetic and literary forms as well as mathematical concepts, to communicate the way that an algorithm works and to convince a reader that the results will be correct..."

text = text.lower()

letters = {}

for i in text:

if i.isalpha() :

if i in letters.keys():

letters[i] += 1

else:

letters[i] = 1

#Could just print(letters), but this is just to print nicely:

for key in sorted(letters.keys()):

print(f'{key} appears {letters[key]} times')

**Screenshot:**

Text

Description automatically generated

**Exercise 3: String functions and dictionary [3 Marks]**

Computer users’ information ID’s, user names, surnames, and roles are recorded as a list of strings, as follows:

                serverInfo = ( "id=0;role=admin;username=joe;surname=naysmith",

                         "surname=suffi;username=sam;role=guest;id=421",

                        "id=33;surname=lee;username=mia;role=staff"  )

A “user database” is a dictionary of computer user records: the key will be an ID and the value another (nested) dictionary, which comprises the key/value pairs of the user’s info.

1. Write a function that takes server info as input and returns the corresponding “database”, as per the example below. Hint: Use string functions such as split/partition/join, etc. as needed.

                db = { 0 : { 'username': "joe", 'surname': "naysmith", 'role': "admin" },

          33 : { 'username': "mia", 'surname': "lee", 'role': "staff" },

          421 : { 'username': "sam", 'surname': "suffi", 'role': "guest" } }

1. Perform a “database query” that retrieves and print all user names and surnames (all capitalized) as well as their roles:

            Naysmith, Joe - admin

Lee, Mia - staff

Suffi, Sam - guest

**Code:**

serverInfo = ( "id=0;role=admin;username=joe;surname=naysmith",

"surname=suffi;username=sam;role=guest;id=421",

"id=33;surname=lee;username=mia;role=staff" )

#a)

def makeDatabase():

database={}

for record in serverInfo:

fields=record.split(';')

fields=sorted(fields,reverse=True)

currID=fields[-1].split('=')[1] #The last field is the ID (since we sorted in descending order)

innerDict={} # The record for each ID

for field in fields[:-1]:#Omit the ID

innerDict[field.split('=')[0]]=field.split('=')[1]

database[currID]=innerDict

return database

db=makeDatabase()

print('db =',db)

#b)

for key in sorted(db.keys()):

record=db[key]

print(f"{record['surname'].title()}, {record['username'].title()} - {record['role']}")

**Screenshot:**

Text

Description automatically generated

**Exercise 4: Dictionary and random class [3 Marks]**

Coding a random cipher

1. Write a *RandomCipher* function that builds a dictionary where each of the 26 letters of the alphabet is randomly matched to another (all lowercase). Each of the matching letters should be unique (in other words, your function creates a one-to-one mapping). Example:

{'a': 's', 'b': 'y', 'c': 'h', 'd': 'i', 'e': 'o', 'f': 'm', 'g': 'n', 'h': 'b', 'i': 'c', 'j': 'a', 'k': 'q', 'l': 'l', 'm': 'g', 'n': 'u', 'o': 'j', 'p': 'x', 'q': 'w', 'r': 'z', 's': 'r', 't': 'k', 'u': 'v', 'v': 'p', 'w': 't', 'x': 'e', 'y': 'f', 'z': 'd'}

**Hint:** use random class

1. Write an *Encode* function that takes a plain text string as input and returns a new string encrypted using your cypher in part (a). Make sure to replace all letters once only; leave punctuation and other non-alphabetical characters as is. Example: if using the above mapping, the text "hello, world!" would be encoded as " bollj, tjzli!”
2. Write a *Decode* function that takes an encrypted text string as input and returns a new string decrypted using your cypher in part (a). Create any intermediate data structure as required to make your code efficient. Example: " bollj, tjzli!!" should be decoded as "hello, world!" (Encoding followed by decoding should always produce the same string.)

**Code:**

import random

#a)

lowercase=[chr(i+97) for i in range(26)]

random.shuffle(lowercase)

randomCipher = {}

for i in range(26):

randomCipher[chr(i+97)]=lowercase[i]

print("Cipher:")

print(randomCipher)

#b)

def encode(s):

ans=''

for ch in s:

if ch.isalpha():

ans+=randomCipher[ch]

else:

ans+=ch

return ans

#c)

def decode(s):

decoder={v:k for k,v in randomCipher.items()}

ans=""

for ch in s:

if ch.isalpha():

ans+=decoder[ch]

else:

ans+=ch

return ans

print('-'\*95)

print("String to be encoded: 'hello world!' ")

encoded=encode('hello world!')

print("Encoded string:",encoded)

print("Decoded string:",decode(encoded))

**Screenshot:**

A screenshot of a computer

Description automatically generated with medium confidence