Q1	Q2	Q3	Q4	Q5	Tot

CEng 445 - Software Development with Scripting Languages

Fall 2016-2017

MT, Open notes (6 pages, 5 questions, 110 points, 110 minutes)

Name: _____ **No:** _____

QUESTION 1.(20 points)

a) You are asked to write a iterator class Cart that will take two lists and iterates on all elements of the cartesian product of the lists. For example, `Cart([1,2],["a","b","c"])` will return an iterator having 6 iterations as
`(1,"a") (1,"b") (1,"c") (2, "a") (2, "b") (2, "c")`
The order of iterations is not important, but if you construct a list with cartesian product of lists and iterate over it you will loose 5 points. Just keep current state in terms of list indexes and calculate the next element.

```
class Cart:
        def __init__(self, a, b):
```

```
for i in Cart([1,2], ["a","b"]):
        print i
# will output (1,"a"), (1,"b"), (2,"a"), (2,"b), each on a seperate line
```

b) Write same iterator as a generator function.

```
def cart(a, b):
```

```
for i in cart([1,2], ["a","b"]):
        print i
```

QUESTION 2.(20 points)

Assume you like to get rid of repeated lines of an input file and write only unique lines on a sorted output file. Assume you are allowed to use two useful command line tools of Linux, sort and uniq. sort will read standard input from file input.txt. It will write sorted output to a pipe. uniq will read this pipe as standard input and eliminate all repeated lines and keep only one unique line. It will write resulting output on output.txt. You do not need to know anything about how sort and uniq works. Briefly you need to execute the commands as shell executes:

```
/usr/bin/sort <input.txt | /usr/bin/uniq >output.txt
```

You are asked to start these processes from Python subprocess module get same task done. Fill in the following function taking input and output file names as parameters and executing sort and uniq as described above.

```
from subprocess import *

def sortuniq(inpfilepath, outfilepath):
```

QUESTION 3.(20 points)

You are asked to get an already implemented Logger object and replace some patterns in its output for highlighting or other purposes. Since different replacement combinations may be required, each highlighter will encapsulate previous one as a chain (see code below).

a) Fill in nextline() methods of two highlighter classes, HLDate and HLIP. Only give bodies of these two methods. Use re module for replacing patterns. Read descriptions in the code.

```
import re

class Logger:
    ''' Assume logger already implemented and has many methods.
            We are only interested in nextline() method'''
    # many other methods here
    def nextline(self):
        ''' returns a new line of log. assume already implemented'''

class Highlighter:      #Base class
    def __init__(self, t):
        self.obj = t

    def __getattr__(self, name):
        return getattr(self.obj, name)

class HLDate(Highlighter):
    def nextline(self):
        ''' get next line of log from member object and return
        with replacing all date occurences of 02/12/2015 into **02/12/2015** '''
```

```
class HLIP(Highlighter):
    def nextline(self):
        ''' get next line of log from member object and return
        with replacing all IP occurences like 144.12.1.121 into 144.12.*.* '''
```

```
a = HLIP(HLDate(Logger(...)))
b = HLIP(Logger(...))

l = a.nextline()
while l != '':
    print l,
    l = a.nextline()
```

b) Which design pattern Highlighter uses?

QUESTION 4.(25 points)

Assume you need a concurrent game room implementation where a game can start only after all of the 4 players sit in their places. Similarly, the game room will be occupied until all players leave their seats. After they left, a new turn can start with new set of players.

The rules are as follows:

- Players call `enter()` method to enter the game room.
- An entering player waits for the next turn if a game is in progress, i.e. last player leaves the room.
- An entering player sits on an empty seat if game is not in progress.
- When fourth player sit on an empty seat, `enter()` call of all four players return, indicating game started and players are inside. Otherwise `enter()` will block.
- Entered players call `exit()` method to leave the room whenever they like.
- A game turn remains in progress until all four players leaves the room. New players cannot sit on empty seats until last player leaves.
- When last player leaves, a new turn can start and players waiting to be seated can start sitting on empty seats.

Complete the following GameRoom class definition in threading environment. Note that you implement this class for existing threads sharing the same GameRoom instance, you shall not start a thread. Existing fields are defined for convenience, you can add any other definitions and fields.

```
from threading import *


class GameRoom:
        def __init__(self):
                self.nempty = 4         # empty seats
                self.ninside = 0        # players started playing ninside>0 -> game i



        def enter(self):









        def exit(self):
```

QUESTION 5.(25 points)

You are asked to write a TCP/IP server that implements a simple application for selling items. A list of prices for items are given as input to server. Clients connect and query prices of items and buy them. Server keeps track of the availability of the item (already sold or not) and confirms the sell.

You are asked to implement this server in multiprocessing environment, where a process is created per accepted connection. The clients can send one of two commands. Server process commands line by line interactively:

- `PRICE` *i*
  i is the item number. Server replies (writes on the connection) the price of item *i*.
- `BUY` *i*
  Server replies either `OK` or `SOLD` depending on if item *i* is already sold or not. If not , reply will be `OK` and all future buy requests of item *i* will get `SOLD` reply.

Assume server will be started by the function startserver (host, port, pricelist). If pricelist is `[10.0, 6.0, 9.0, 3.0]`, there are 4 items in the server and clients can request `PRICE` and `BUY` requests with values 0 to 3.

Only provide essential parts of your code, do not make any error handling. Note that connections need to update a shared sold/not sold information.

```
from multiprocesing import *
from socket import *

def parsecommand(line):          # parse client input provided for convenience
        (req, num) = line.rstrip('\n').split(' ')
        return (req, int(num))

class Server(          ):                # per process code is here
        def __init__(self,                                  ):
```

```
def startserver( host, port, pricelist ):
```