Q1	Q2	Q3	Q4	Q5	Q6	Tot

CEng 445 - Software Development with Scripting Languages

Fall 2016-2017

Final, Open notes (7 pages, 6 questions, 120 points, 110 minutes)

Name: ______________________________ **No:** __________

QUESTION 1.(20 points)

Assume you have a concurrent four person game where the four players are required to keep the game going and the game should continue forever. So that a player can only leave the game when there is a new player wiling to enter, and a new player can enter if an existing player exits.
The rules are as follows:

- New player call enter() method to enter the game.
- Existing players call exit() method to leave the game.
- An entering player waits until an existing player decides to leave.
- A leaving player waits until a new player tries to enter.
- For simplicity assume four players are already inside.

Complete the following Game class definition in threading environment. Note that you implement this class for existing threads sharing the same Game instance, you shall not start a thread. Existing variables are defined for convenience, you do not have to use all, and can add any other variables.

```
from threading import *
class Game:
        def __init__(self):
                self.waiting = 0        # players waiting to enter
                self.leaving = 0        # players waiting to leave
                self.lck = Lock()




        def enter(self):
                self.lck.acquire()
                self.waiting += 1
                self.toleave.notify()
                while self.leaving == 0:
                        self.toenter.wait()
                self.waiting -= 1
                self.lck.release()




        def exit(self):
                self.lck.acquire()
                self.leaving += 1
                self.toenter.notify()
                while self.leaving == 0:
                        self.toleave.wait()
                self.leaving -= 1
                self.lck.release()
```

QUESTION 2.(20 points)

Complete the missing blocks in the code. Descriptions are in the comments. Assume HTTP query encoding is the POST data of a form sent to an HTTP server. No escape handling is required.

```
class Loader:   # abstract class
        def load(self,store, str): pass
class Saver:    # abstract class
        def save(self,store): pass
class Store:
        def __init__(self, ldr, svr):
                self.store, self.ldr, self.svr = ({}, ldr, svr)
        def parse(self, fname):
                self.ldr.load(self.store, open(fname,"r").read())
        def dump(self):
                return self.svr.save(self.store)
class XMLLoader(Loader):
        def load(self, store, str):
                '''assume this parses str as xml and puts on store'''
class XMLSaver(Saver):
        def save(self, store):
                '''assume this generates store contents as XML string'''
class HTMLQLoader(Loader):
        def load(self, store, str):
                '''str is an HTTP query. parse it simply (no escapes)
                   (do not use modules) put values in store dictionary'''
                for t in str.split("&"):
                    k,v = t.split("=")
                    store[k] = v

class HTMLQSaver(Saver):
        def save(self, store):
                '''return store dictionary content as a HTTP query.
                   (do not use modules). Assume simple dump to string.'''
                lst = ["=".join((k,v)) for (k,v) store.items()]
                return "&".join(lst)


def Generator(inp, out):
        if inp == 'xml': loader=XMLLoader()
        else:            loader=HTMLQLoader()
        if out == 'xml': saver=XMLSaver()
        else:            saver=HTMLQSaver()
        return Store(loader,saver)
```

a) Generator is an example of Factory design pattern

b) Store, Loader, Saver classes form an example of Strategy design pattern.

For the remaining questions assume the following scenario. You have a publisher site for books. Each book has one or more authors. Also books have categories where each book belonging to exactly one category. Author is entered as comma seperated list of author names by the user.

A sample HTML form for a book is:

```
<form id="bookform" method="post" action="add">
<input type="text" name="title"/>
<input type="text" name="author"/>
<input type="text" name="year"/>
<select id="catselect" name="category">
<option value="child">Children</option>
<option value="scifi">Science Fiction</option> ...
</select>
<input type="submit">
</form>
```

QUESTION 3.(20 points)

write the add function for a WSGI application that will convert this posted form into a dictionary, call savebook(rec) and return a "Success" word in HTML result. Sample dictionary value given below. Complete the following code — respecting comments — for a simple version of add.

```
rec = { title = 'Asterix and Cleopatra',
        authors = ['Rene Goscinny','Albert Uderzo'],
        year = '2004', category = 'child'}

import cgi

def add(                         ):
        # read form data as string on variable 'poststr'
        poststr =
        # parse post data on variable 'posted'




        # compose 'rec' dictionary from 'posted' as given above










        # call
        savebook(rec)

        # compose a success and simple HTML page "success"
        succstr='<html><body>Success</body></html>'



        return
```

QUESTION 4. (20 points)

Assume you write this application using Django and need to design the model.
a) Complete the following model for Book assuming it only has the fields specified in the form in Page 3. Also model its relation with Author and Category classes.

```
from django.db import models
class Author(models.Model):
        name = models.CharField(max_length=100, primary_key = True)

class Category(models.Model):
        name = models.CharField(max_length=100, primary_key = True)

class Book(models.Model):
        title = models.CharField(max_length=200)
        year = models.CharField(max_length=8)
        category = models.ForeignKey(Category)
        authors = models.ManytoManyField(Author)
```

Write the following queries in this model:
b) The book with title 'Django Tutorial':

```
Book.objects.get(name='Django Tutorial')
```

c) All books published in 2016 and containing 'python' in their titles:

```
Book.objects.filter( title__contains ='python').filter(year=2016)
```

d) All books in category with name 'Sci-Fi':

```
Book.objects.filter(category=Category.objects.get(name='Sci-Fi'))
```

e) All books of author 'Carl Sagan':

```
Author.get(name='Carl Sagan').book_set()
```

f) All distinct authors that wrote at least one 'Sci-Fi' category book (hint: `v in lst`) :

```
added=[]
for b in Book.objects.filter(category=Category.objects.get(name='Sci-Fi')):
    for a in b.authors.all():
        if not (a in added):
            added.append(a)
```

QUESTION 5.(20 points)

a) Write a Django view function that gets the posted data and compose a dictionary value as specified in Page 3, then render template "added.html" with this dictionary.

```
# assume everything necessary is imported here

def add(                ):
        # read form data as string on variable 'poststr'
        poststr =
        # parse post data on variable 'posted'
        # compose 'rec' as given above
        # this is duplicate of previous question, so skip it
        # assume 'rec' has the correct value

        #set 'rec' as the template context and show 'added.html'
```

b) The following template for "added.html" is missing the author information. Complete it as each author is given as a `<li>` element.

```
<html><body>You added the book:<br/>
<b>Title</b>{{ title }}</br/>
<b>Author</b><div>
<ul>{#fill here #}









</ul>
</div>
<b>Year</b>{{ year }}</br/>
<b>Category</b>{{ category }}</br/>
</body></html>
```

QUESTION 6.(20 points)

Assume you have a Javascript code that is used by the HTML page containing the form in Page 3. The form contains a selection element for category names that is better be dynamic instead of a fixed list. As list of categories updated in the system, the `<select>` element should have a different set of options. Assume category data can be retrieved by an AJAX GET request to relative URL /categories. This request returns a JSON object as:

```
[{"id":"scifi", "name":"Sience Fiction"},
 {"id":"child", "name":"Children"},...]
```

Complete the following Javascript code that will dymanicall set content of the `<select>` element that is initialy does not contain any `<option>`. You can assume JQuery is loaded. Both solutions with or without JQuery is accepted.

```
function fillcategories() {
        // make the AJAX call get JSON in 'retval'













        // parse JSON into Array 'catlist'




        // get <select> element in variable 'sel'
        var sel =



        // fill in the inner HTML of the select object
        // with values from 'catlist'














}
```