Name : _____

Grading TA: _____

- INTEGRITY: By taking this exam, you pledge that this is your work and you have neither given nor received inappropriate help during the taking of this exam in compliance with the Academic Honor Code of Georgia Tech. Do NOT sign nor take this exam if you do not agree with the honor code.
- DEVICES: If your cell phone, pager, PDA, beeper, iPod, or similar item goes off during the exam, you will lose 10 points on this exam. Turn all such devices off and put them away now. You cannot have them on your desk.
- ACADEMIC MISCONDUCT: Academic misconduct will not be tolerated. You are to uphold the honor and integrity bestowed upon you by the Georgia Institute of Technology.
  - Keep your eyes on your own paper.
  - Do your best to prevent anyone else from seeing your work.
  - Do NOT communicate with anyone other than a proctor for ANY reason in ANY language in ANY manner.
  - Do NOT share ANYTHING during the exam. (This includes no sharing of pencils, paper, erasers).
  - Follow directions given by the proctor(s).
  - Stop all writing when told to stop. Failure to stop writing on this exam when told to do so is academic misconduct.
  - Do not use notes, books, calculators, etc during the exam.
- TIME: Don't get bogged down by any one question. If you get stuck, move on to the next problem and come back once you have completed all of the other problems. This exam has 5 questions on 9 pages including the title page. Please check to make sure all pages are included. You will have 50 minutes to complete this exam.

I commit to uphold the ideals of honor and integrity by refusing to betray the trust bestowed upon me as a member of the Georgia Tech community. I have also read and understand the requirements outlined above.

Signature: _____

Question	Points	Score
1. Multiple Choice	4	
2. Movies	24	
3. Get Users	12	
4. Taxes Download	13	
5. Draw Tree	10	
Total:	63	

1. *(4 points)*
For each of the following multiple choice questions, indicate the single most correct answer by circling it!

(a) [1 pt] Which of the following Regular Expressions will ONLY match possible dollar amounts from $1 to $999,999,999? (e.g. it should NOT match $999,99,999 or $X99,999 or X9,999 or $9,123,999,999)

A. `\$(?:\d{1,3},?){0,3}`
B. `\$(?:[^0](?:[0-9]){0,2})(?:,[0-9]{3}){0,2}`
C. `.(?:(?:[1-9]\d{0,2},?)|(?:\d{0,2},?){0,2})`
D. `\$(?:[^0][0-9]{0,2})(?:(?:(?:,[0-9]{3}){1,2})*){0,2}`
E. `\$[1-9]\d{0,2}(?:,\d{3}){0,2}`

(b) [1 pt] Given this variable assignment: `var = "$WillieIsFunkMasterFlex$ (CASH MONEY)"` Which of the following returns exactly **TWO** matches?

A. `re.findall("[Willie]",var)`
B. `re.findall("\(.+\)",var)`
C. `re.findall("[$|i]",var)`
D. `re.findall("[()a-zA-Z]{7,15}",var)`

(c) [1 pt] Which of the following is the return type of a PyMySQL cursor object's execute method?

A. `None` **B.** `int` C. `float` D. `<pymysql.cursors.Cursor object>`

(d) [1 pt] Which of the following is **NOT** a syntactically correct way to change an xml element's attribute?

A. `myElement.attrib["Jake"] = "1"`
B. `myElement.set("Jake","1")`
C. `myElement.attrib["Jake"] = 1`
D. `myElement.attrib.update("Jake") = "1"`

2. *(24 points)*

A table has been created for you with the following command:
CREATE TABLE Movies (Title TEXT NOT NULL, Year INTEGER NOT NULL, Genre TEXT, Rating FLOAT) The database has contents such as the following (but with many more records):

Title	Year	Genre	Rating
Pulp Fiction	1994	Drama	9.0
300	2006	Action	7.7
The Iron Giant	1999	Action	7.9
Forgetting Sarah Marshall	2008	Comedy	7.2
American Psycho	2000	Drama	7.5

(a) [3 pts] Write an SQL statement that returns a listing of all movies where the Title begins with "The".

Solution: SELECT * FROM Movies WHERE Title LIKE "The%"
Grading:
+1 for SELECT * (or SELECT Title)
+1 for FROM Movies
+1 for WHERE Title LIKE 'The%'

(b) [4 pts] Write an SQL statement that will return the top 3 highest rated movie titles and their respective ratings:

Solution: SELECT Title,Rating FROM Movies ORDER BY Rating DESC LIMIT 3
Grading:
+1 for SELECT Title, Rating
+1 for ORDER BY Rating +1 for DESC
+1 for LIMIT 3

(c) [4 pts] Write an SQL statement that will return the rating and year for the highest rated movie each year. Label the columns RatingMax and Year. Order your results by rating, with the smallest number first.

Solution: SELECT MAX(Rating) as RatingMax, Year FROM Movies GROUP BY Year ORDER BY RatingMax
Grading:
+1 for SELECT MAX(Rating) AS RatingMax
+1 for GROUP BY Year +1 for ORDER BY RatingMax
+1 for everything else correct.

(d) [4 pts] Write an SQL statement that would add "Punch Drunk Love" into the table.

That movie was released in 2002, is a Romance, and has a rating of 7.3.

Solution: INSERT INTO Movies (Title, Year, Genre, Rating) VALUES ("Punch Drunk Love", 2002, 'Romance', 7.3)

Grading:
+1 for INSERT INTO Movies
+1 for column/field names (in any order) (could also omit!)
+1 for VALUES
+1 for data in same order as column names (or in same order as table!)!

(e) [3 pts] Write an SQL statement that would change 300's rating to 5.

Solution: UPDATE Movies SET Rating=5 WHERE Title = '300'

Grading:
+1 for UPDATE Movies
+1 for SET Rating=5
+1 for WHERE Title = '300' (or Title LIKE '300')

(f) [3 pts] Write an SQL statement that returns the average Rating for all movies. Name the returned column "AVGRATE".

Solution: SELECT AVG(Rating) AS AVGRATE FROM Movies

Note: could also do SUM(Rating)/ COUNT(Titles)

Grading:
+1 for SELECT AVG(Rating)
+1 for AS AVGRATE
+1 for FROM Movies

(g) [3 pts] Write an SQL statement that removes all movies released before 2000 from the table.

Solution: `DELETE FROM Movies WHERE Year < 2000`

Grading:
+1 for DELETE FROM Movies
+1 for `WHERE Year < 2000`
+1 for not having a DELETE *

3. *(12 points)*
A PizzaOrders table has been created with the following command:

```
CREATE TABLE PizzaOrders (
OrderID int(11) NOT NULL AUTOINCREMENT,
User  text, Cheese int(11), Pepperoni int(11), Veggie int(11));
```

Each record in this table represents a pizza order (for some number of cheese, pepperoni, or veggie pizzas) by a particular User. You have been asked to generate a report in CSV format that has each User listed once, with the total number of cheese, pepperoni and veggie pizzas that they have ever ordered (which may be a single order, or multiple orders in the database). For example, if the User "Bob" had ordered 2 Cheese pizzas in one order, and then made a second order for one Pepperoni pizza, his line in the CSV output would look like:

```
Bob, 2, 1, 0
```

Write a function named **exportPizzaTotals** that opens a connection to the academic-mysql.cc.gatech.edu database using the "cs2316db" database and the "cs2316" username with "SECRET" as the password. It should download the data and write the output to a CSV file named "pizzaExport.csv".

Solution:

```
import pymysql
import csv

def exportPizzaTotals():
  db = pymysql.connect(host='academic-mysql.cc.gatech.edu', user='cs2316',
                    db='cs2316db', passwd="SECRET")
  c = db.cursor()
  SQL = "SELECT User, SUM(Cheese), SUM(Pepperoni), SUM(Veggie)
             FROM PizzaOrders GROUP BY User"

  c.execute(SQL)

  f = open("pizzaExport.csv", "w")
  writer = csv.writer(f)
  for item in c:
     writer.writerow(item)
  f.close()
  c.close()
  db.close()
```

Grading:
+1 point for importing pymysql (-1 if they used csv without importing it)
+1 point for correct db connection with all parameters.
+1 for having a valid SQL statement that retrieves the data.
+1 for executing the SQL statement.
+2 points for summing up total of pizzas for each user, either in SQL or via some complicated python.
+1 for opening the correct output file in write mode.
+2 for writing each row of data out correctly.
+3 for closing the file, cursor, and database objects!

4. *(13 points)*

The hypothetical website http://www.allTheTaxes.com has a simple HTML structure that displays tax records as list items `<li>` of an unordered list `<ul>`. Here is an example of a few records in the webpage:

```
...
<ul>
<li>Address: 123 Maryweather Ave, Notown GA 30234<br/>$4,234.32</li>
<li>Address: 456 Suesucker Blvd, Someplace GA 24232<br/>$234</li>
<li>Address: 6 1/2 hard-street Rd NW, other GA 34323<br/>1323.40</li>
...
</ul>
...
```

Each address falls after a begin list item tag and the string "Address:" and ends with a break tag. The tax amount falls between the break tag and the end list item tag. Some (but not all) of the tax amounts have dollar signs in front of them, some are rounded to the whole dollar, and others have dollars and cents. Some dollar amounts have commas every three digits and others do not. Also, the string "Address:" without a begin list item tag may appear elsewhere in the webpage.

You are hired to write a function called **getTaxes** which accepts no parameters. Your funciton will go to that URL (place the url in your code) and download the webpage. Return a list of tuples, where each tuple is a tax record made up of a string and a floating point number. The string is the address, and the floating point number is the dollar amount. It is permissible to convert a decimal number such as 213.40 to 213.4 as part of the converion to a float.

Solution:

```
import urllib.request
from re import findall
def getTaxes():
   request = urllib.request.urlopen("http://www.allthetaxes.com")
   html = request.read()
   text = str(html)
   tempList = findall("<li>Address:([^<]+)<br/>\$?([.0-9,]+)</li>", text)
   aList = []
   for item in tempList:
      addr = item[0]
      dollarS = itemp[1]
      dollarS = dollarS.replace(",","")
      dollarF = float(dollarS)
      aList.append( (addr, dollarF)
```

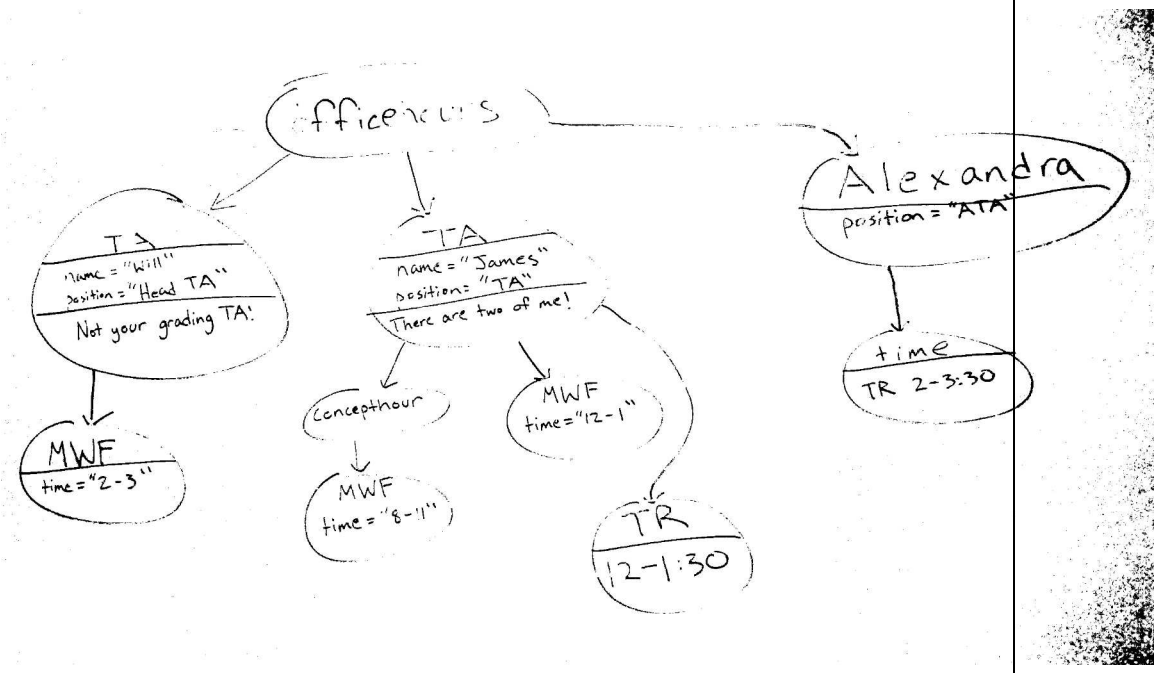
```
   return aList
```

Grading:
+1 imports urllib.request correctly.
+1 correct urlopen
+1 response.read()
+1 casts response to string
+3 correctly finds all address and dollar amounts (iteration/find or regex)
+2 correctly isolates the adddress string and dollar amount string
+1 removes commas in dollar amount.
+1 correctly converts to float
+1 makes tuple with correct data.
+1 returns list with correct data in it.

5. *(10 points)*
Draw the object tree hierarchy that is encoded by the following textual XML:

```
<officehours>
  <TA name="Will" position="Head TA">Not your grading TA!
     <MWF time="2-3" />
   </TA>
   <TA name="James" position="TA">There are two of me!
      <concepthour>
         <MWF time="8-11" />
      </concepthour>
      <MWF time="12-1"></MWF>
      <TR>12-1:30</TR>
   </TA>
   <Alexandra position="ATA">
      <time>TR 2-3:30</time>
   </Alexandra>
</officehours>
```

Draw each element indivdually as a circle that contains the tag name, attribute key/value pairs on separate lines, and any text, in that order. Also include lines between each element's circle showing parent/child relationships. Draw a line betwen each parent and (direct) child element, with an arrow head pointing towards the child element.



Solution:

Grading:
+1 for correct root element, with no parent
+1 for each TA element (and Alexandra)
+1 for each MWF/TR/time/conceptHour element, connected to the correct parent
-1 if any node is missing an arrow, maximum penalty of 2