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 4 questions on 10 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	7	
2. Functional Blanks	7	
3. Baseball	26	
4. fileMin	10	
Total:	50	

1. *(7 points)*
   For each of the following multiple choice questions, indicate the most correct answer! Indicate your selected answer by circling it.

   (a) [1 pt] Which of the following is an aggregate function in MySQL?
   A. SUM B. ABS C. NOW D. ROUND

   (b) [1 pt] Which of the following statements about SQL is true?
   A. The syntax "DELETE *" is valid.
   B. All keywords must be capitalized.
   C. If specified, WHERE must come before FROM.
   D. The % character is a wildcard when used with LIKE.

   (c) [1 pt] Which of the following is true of properly-formatted XML?
   A. There may be multiple root elements.
   B. All elements in the tree are direct subelements of the root.
   C. An element may have any number of attributes.
   D. An element may have no tag and no attributes.

   (d) [1 pt] The element at the top of the XML tree is called a
   A. child
   B. root
   C. branch
   D. sub-tree
   E. sub-element

   (e) [1 pt] Every child element in an XML tree can have only one parent.
   A. True B. False

   (f) [1 pt] Every parent element in an XML tree can have only one child.
   A. True **B. False**

   (g) [1 pt] Every element in an XML tree has at least one parent.
   A. True **B. False**

2. *(7 points)*

Fill in the following blanks so that the python code is correct:

```
 originalList = [-4,-3,-2,-1,1,2,3,4]
```

1) all elements are multiplied together to produce the product of all values in originalList

```
 >>> ___________(lambda x,y: x*y, originalList)
576
```

2) all elements in originalList are doubled

```
>>> list(____________(lambda x: 2*x, originalList))
[-8, -6, -4, -2, 2, 4, 6, 8]
```

3) only positive elements are retained

```
>>> list(_____________(lambda x: x > 0, originalList))
[1, 2, 3, 4]
```

4) only multiples of 3 and multiples of 5 are returned

```
>>> list(filter(_____________________________________________, originalList))
[-3, 3]
```

5) all elements in the list are printed out (A list of None's is returned)

```
>>> list(map(_________________________________________, originalList))
-4
-3
-2
-1
1
2
3
4
[None, None, None, None, None, None, None, None]
```

6) the expression sums all negative, odd numbers (note these numbers must be both odd and negative)

```
>>> reduce(________________________________________________,

         filter(________________________________________________, originalList)
          )
-4
```

Solution:

```
1) reduce(lambda x,y: x*y, originalList)

2) list(map(lambda x: 2*x, originalList))

3) list(filter(lambda x: x > 0, originalList))

4) list(filter(lambda x: x % 3 == 0 or x % 5 == 0, originalList))

5) list(map(print, originalList))

6) reduce(lambda x,y: x + y, list(filter(lambda x: x < 0 and x % 2 == 1, originalLi
```

3. *(26 points)*
A table has been created for you with the following command:
CREATE TABLE baseball (Name TEXT NOT NULL, Game_Date TEXT NOT NULL, AtBats INT(10), Hits INT(10))

The database has contents such as the following (but with many more records):

The baseball table:

Name	Game_Date	AtBats	Hits
Babe Ruth	11/15/1930	5	2
Willie Mays	07/18/1965	10	6
Albert Pujols	10/17/2011	6	4

(a) [10 pts] Write a function called addStats that takes in a single parameter. The parameter will be a list of tuples, where the tuples are in the following form:

(Name, Date of Game, Number of At Bats, Number of Hits)

So a possible parameter passed into your addStats function could be:

[("Barry Bonds", "09/14/2007", 3, 1), ("Prince Fielder", "05/16/2010", 7, 5)]

The list can be of any length. Your addStats function should take the data from each tuple and insert it into the baseball table using PyMySQL code. The table is located on a DB server "exam4.gt.edu" , "Test4" is the database name, "cs2316" is the username, and "yellowjackets" is the password. You may assume every tuple within the passed in list will have four entries in the correct format necessary to insert into the baseball table. You may also assume the internet connection will always work (You do not need to do error checking).

Solution:

```
import pymysql

def addStats(stats):
   db = pymysql.connect(host = "exam4.gt.edu", user = cs2316, passwd =  yellowj
   for player in stats:
      sql = INSERT INTO baseball (Name, Game_Date, AtBats, Hits) VALUES (%s, %s

      cursor = db.cursor()
      cursor.execute(sql, player)
      cursor.close()
      db.commit()
   db.close()
```

Grading:
+1: imports pymysql
+1: has correct function header
+2: connects to database (+1 for a decent attempt, +1 for doing it correctly)
+1: Inserts all tuples into the database

+2: uses SQL statement (+1 for a decent attempt, +1 for doing it correctly)
+2: uses cursor (+1 for a decent attempt, +1 for doing it correctly)
+1: Closes DB and cursor(s).
-1 if they forget to commit after updating the database.

(b) [16 pts] Write a function called calcAverage that does not take in any parameters. Your calcAverage function should return a list of tuples, where the tuples are in the following form:

(Name of Player, Total Batting Average)

The returned list should have as many tuples as there are unique players in the baseball table above. The table can have any number of entries in it. Your calcAverage function should use PyMySQL code to select every distinct player and their total batting average, where:

$totalBattingAverage = TotalNumberOfHits/TotalNumberOfAtBats$

Realize that the total number of hits is not necessarily from one game that player played, but from every game that player has played; as well as for the total number of at bats. The database your function should connect to is the same one used in Part 1. Here, you may also assume the internet connection will always work. (Hint: If you cannot remember the SQL code that will automatically return the batting average per player for you, you can use a simple query that will return every row and then use python code to calculate the data.)

Solution:

```
EASY SOLUTION:
import pymysql
def calcAverage():
   db = pymysql.connect(host = ``exam4.gt.edu'', user = ``cs2316'', passwd = `

   sql = ``SELECT Name, (SUM(Hits)/SUM(AtBats)) FROM baseball GROUP BY Name''

   cursor = db.cursor()
   cursor.execute(sql)

   stats = cursor.fetchall()
   cursor.close()
   db.close()
   finalList = []
   for player in stats:
      finalList.append(player)

   return finalList
```

Grading:
+1: imports pymysql
+1: has correct function header
+2: connects to database (+1 for a decent attempt, +1 for doing it correctly)
+2: uses SQL statement (+1 for a decent attempt, +1 for doing it correctly)
+2: uses cursor (+1 for a decent attempt, +1 for doing it correctly)

+1: uses .fetchall() or the equivalent to (basically attempts to use retrieved data in python)
+2: appends to the final list (+1 for appending something, +1 for appending the correct tuple)
+3: Correctly finds total batting average (SQL or Python). (-2 for forgetting sum of hits/at bats, -1 for minor errors)
+1: returns the final list
+1: Closes the DB/Cursor

4. *(10 points)*
Write a function named `fileMin` that accepts the name of a file to open as a string parameter. The function should open the file and read in each line. Each line will contain a single integer number. Your function should find the minimum (smallest) number in the file and return it (as an integer). You may assume that the file name is correct and will exist, that the file will have at least one number in it, and that all numbers are valid integers.

For example, calling the funtion fileMin on file test.txt containing:

```
5
32
-1
0
```

would return the integer -1.

Solution:

```
def fileMin(fileName):
   f = open(fileName,"r")
   data = f.readlines()
   min = int(data[0])
   for item in data[1:]:
      num = int(item)
      if num < min:
         min=num
   f.close()
   return min
```

or:

```
def fileMin( fileName):
   f = open(fileName, "r")
   aLine = f.readline()
   min = int(aLine)
   aLine = f.readline()
   while( len(aLine) > 0):
       num = int(aLine)
       if min > num:
          min = num
       aLine = f.readline()
   f.close()
   return min
```

Grading: 1 point for correct header,
2 points for opening file (in read mode)
2 points for converting strings to ints/floats before comparison
2 points for finding the correct minimum in most ”standard” cases.
1 point for working with a file that has a single really large number in it (such as 999999999999999999). [i.e. they use the first number in the file as the starting point for the minimum.]
1 point for closing the file.
1 point for returning the integer.
-2 for failing to read the data from the file.