CS 1301 Homework 9

Functional Programming, Classes, and Recursion!

Due: Monday, April 20th before 11:55 pm

THIS IS AN INDIVIDUAL ASSIGNMENT!
Students are expected to abide by the Georgia Tech Honor Code. Collaboration at a reasonable level will not result in substantially similar code. Students may only collaborate with other students currently in CS 1301, TAs, and the lecturer.

Files to Submit:
hw9.py (out of 100 points)

For Help
- TA Helpdesk – Schedule posted on class website.
- Email the TAs
- Jay's Office Hours

Notes:
- Do not forget to include required comments and collaboration statement
- Do not wait until the last minute to start the assignment
- If you find any significant error(s) in the assignment, please notify a TA immediately
- Have fun :)

Part 1 – taco cat (15 pts)

Parameters: a string
Return Value: a Boolean

For this problem, you will write a function called *isPalindrome,* which takes one argument and returns whether or not the string is a palindrome. **You must use recursion to solve the problem!** No credit will be given to iterative solutions. If the characters do not match case it is not a palindrome. (e.g. Dood is not a palindrome, while dood is.)

Example:

```
isPalindrome(“dood”)
> True
isPalindrome(“racecars”)
> False
isPalindrome(“racecar”)
> True
```

Part 2 – Bank Account (85 pts)

You’ve learned about classes from lecture, so for this problem, you will be making a BankAccount class. Please read everything carefully before implementing any code.

2.1 BankAccount (15 pts)

The bank class will have the following attributes:

- balance
- numDeposits
- numWithdraws

It will also have the following methods:

- getBalance()
- getWithdraws()
- getDeposits()
- deposit()
- withdraw()
- batchDeposit()

CONTINUED ON NEXT PAGE

During creation, the user has the option to make an initial deposit (balance defaults to 0.0).

```
account1 = BankAccount()
account2 = BankAccount(500.0)
```

2.2 Deposit (10 pts)

Parameters: Amount to deposit
Return value: None

This method should allow you to add money to your account.   Remember to keep track of the number of times a deposit as occurred.

Example:

```
baller = BankAccount(1000000.0)
baller.deposit(1000000.0)
stacks = baller.getBalance()
print( stacks )
>> 2000000.0
print( baller.getDeposits() )
>> 1.0
```

2.3 Withdraw (15 pts)

Parameters: The amount to withdraw
Return value: the amount withdrawn (-1 if you withdrew too much!)

This method should allow you to withdraw money from your account. If there is not enough money, you should not be able to withdraw anything! In this case, reject and return -1! Remember to increment the number of withdraws ONLY if the withdraw is successful.

Example:

```
myAccount = BankAccount()
money = myAccount.withdraw(5.0)
print( money )
>> -1.0
myWithdraws = myAccount.getWithdraws()
print( myWithdraws )
>> 0.0
```

```
myAccount.deposit(1.0)
money = myAccount.withdraw(1.0)
print ( myAccount.getWithdraws() )
>> 1.0
```

2.4 batchDeposit (35 pts)

Parameters – a string representing a text file
Return Value - None

This function will represent a string representing a text file. There's one problem: not all items on the list are numbers! For example, a text file may look something like this:

```
deposits.txt
10
12
#bandz
10000
hax
130
```

This function must use **filter** and **reduce** to add all numbers to the account balance (It would be good to use map to convert strings into floats too!). Remember to keep track of the number of deposits.

Example:

```
myAccount = BankAccount()
myAccount.batchDeposit("deposits.txt")
print( myAccount.getBalance() )
>> 10152.0
print( myAccount.getDeposits() )
>> 4.0
```

2.5 getBalance/getWithdraws/getDeposits (10 pts)

Parameters: None
Return Value: float representing the corresponding information

In each of the above functions, a call must simply return the corresponding attribute. Each function should only be one line of code. You may wonder why we don't just return that attribute. This idea is generally used to protect information from others while still allowing them to access it. This concept is commonly referred to as *encapsulation* or *information hiding*.

Example:

```
myAccount = BankAccount(100.0)
print( myAccount.getBalance() )
>> 100.0
print( myAccount.getDeposits() )
>> 0.0
money = myAccount.withdraw(50.0)
print( myAccount.getWithdraws() )
>> 1.0
myAccount.deposit(5.0)
print( myAccount.getDeposits() )
>> 1.0
print( myAccount.getBalance() )
>> 55.0
```

Grading

Part 1: taco cat (15 pts)

Function header is correct	1
Uses recursion to produce correct result	14

Part 2: Bank Account (85 pts)

2.1 BankAccount (15 pts)

Class named correctly	1
All attributes are named correctly	5
Class allows optional initial deposit	8
All methods are present	1

2.2 deposit (10 pts)

Function header is correct	1
Correctly adds to balance	5
Properly increments numDeposits	4

2.3 withdraw (15 pts)

Function header is correct	1
Correctly removes from balance	5
Properly increments numWithdraws	4
Returns withdrawn amount	5

2.4 batchDeposit (35 pts)

Function header is correct	1
Reads and closes file correctly	5
Uses filter	10
Uses reduce	10
Correctly correctly adds to balance	5
Correctly increments numDeposits	4

2.5 getBalance/getWithdraws/getDeposits (10 pts)

Functions each named correctly	5
Functions return corresponding attributes	5