Important

1. Due Date: **Monday, April 17th**
2. This homework is graded out of 100 points.
3. This is an Individual Assignment. You may collaborate with other students in this class. Collaboration means talking through problems, assisting with debugging, explaining a concept, etc. Students may only collaborate with fellow students currently taking CS 1301, the TA's and the lecturer. You should not exchange code or write code for others. For individual assignments, each student must turn in a unique program. Your submission must not be substantially similar to another student's submission. Collaboration at a reasonable level will not result in substantially similar code.
4. For Help:
    - TA Helpdesk (Schedule posted on class website.)
    - Email TA's or use Piazza Forums Notes:
    - How to Think Like a Computer Scientists [http://openbookproject.net/thinkcs/python/english3e/]
    - CS 1301 Python Debugging Guide [http://www.cc.gatech.edu/classes/AY2016/cs1301_spring/CS-1301-Debugging-Guide/index.html]
5. Don't forget to include the required collaboration statement (outlined on the syllabus). Failing to include the Collaboration Statement will result in no credit.
6. Do not wait until the last minute to do this assignment in case you run into problems.
7. Comment or delete all your function calls. When your code is run, all it should do is build without any errors. Having function calls or extraneous code outside the scope of functions will result in a lower grade. (import statements and global variables are okay to be outside the scope of a function)
8. **Read the entire specifications document before starting this assignment.**
9. **IF YOUR CODE CANNOT RUN BECAUSE OF AN ERROR, IT IS A 0%.**

Object-Oriented Programming

Python is an **object-oriented programming language**. That means it provides features that support object-oriented programming (**OOP**). Object-oriented programming has its roots in the 1960s, but it wasn't until the mid 1980s that it became the main programming paradigm used in the creation of new software.

In object-oriented programming the focus is on the creation of objects, which contain both data and functionality together. Usually, each object definition corresponds to some object or concept in the real world and the functions that operate on that object correspond to the ways real-world objects interact. We've already seen classes like `Turtle`, `Math`, `Random` and many others. We are now ready to create our own user-defined class.

[http://openbookproject.net/thinkcs/python/english3e/classes_and_objects_I.html]

Introduction

The goal of this homework is for you to showcase your knowledge about how to properly use OOP. For this assignment you will need to implement 2 classes. Refer to the rubric to see how points will be rewarded for each function. You have been given `bank_account.py` to fill out with instructions in the docstrings. However, below you will find more detailed information to complete your assignment. Read it thoroughly before you begin. Don't forget to include your name and your collaboration statement. Re-download your submission from T-Square after you submit it to make sure that your code runs successfully.

BankAccount

The BankAccount class will represents a real life Bank Account in which the account holder can do different transactions to his or her account. This user is an instance of the Person class. Depending on the type of account, the account holder will be able to deposit or withdraw money to and from the account. This class has other useful methods that allow the holder to deposit many amounts at once. Given a file with a batch of money amounts, we would want to automatically add all these amounts to the account.

```
| Attributes:
|      acct_holder: A Person object representing the account holder
|      acct_type: A string consisting of: "Savings" or "Checking" and
|                 optional "Youth" Prefix
|      balance: An int representing balance of account. Can be passed
|               in, and if it is not, should be defaulted to 0.
| Methods:
|     __init__(self, acct_holder, acct_type, balance = 0)
|          Initialize a BankAccount obj whose acct_holder:
|          *acct_holder*, acct_type: *acct_type*, and balance:
           *balance*
|     changeAccountType(self, newType)
|          Update the acct_type attribute to newType. Return None.
|     deposit(self, amt)
|          Deposit should update the balance of the account.
|          Returns True if successful, False if amt is negative
|     withdraw(self, amt)
|          Update the balance of the account if applicable. Don't
           withdraw if there's not enough money/ amt is negative.
|          Return True if successful, False otherwise
|     batchDeposit(self, file)
|          Deposits money into account.
|          All Lines with only numbers (and whitespace/newline chars)
|          Should be deposited into account, see sample test files.
|          Returns true if successful, False if no deposits were made
|     transfer(self, other, amt)
|          Updates balances of both accounts. Transfer if withdraw
           from self is valid and deposit to other is valid.
|          eg: amt is positive, other has enough balance in account.
|          Returns True if successful, False if otherwise
|     __gt__(self, other)
|          Return True if self’s balance is greater, otherwise False
|     __lt__(self, other)
|          Return True if self’s balance is less, otherwise False
|     __eq__(self, other)
|          Return True if all attributes are equal, otherwise False
|     giveToCharity(self, list_charities, amount):
|          list_charities are BankAccount objects
|          Transfer amount to the charity with the smallest balance.
|          Split the amount if there are multiple smallest charities.
|          Return True if successful, False if amount cannot be
|          withdrawn or there are no charities.
```

Person

A Person Object represents a person that has a Bank Account (a BankAccount object). A Person can start an account, and has related methods for their account. For this assignment, a Person will only ever have a single account, which is None before they start an account.

```
| Attributes:
|      name: A string representing the name of the Person.
|      age: An int representing age of the person.
|      wallet: An int representing amount in a person’s wallet. Can be
|          passed in, and if it is not, should be defaulted to 0.
|       account: A BankAccount object.
| Methods:
|     __init__(self, name, age, wallet = 0)
|          Initialize a Person object whose name:*name*
|          age: *age*, wallet *wallet*, and account to None
|     startAccount(self, a_type, balance = 0)
|          Start a new BankAccount with balance or 0 if not given.
|          Make account type “Youth” if under 18. (eg: “Youth
Savings”)
|     birthday(self)
|          Increment age by 1
|          If Person turns 18, update account type to not have “Youth”
|          Assume a person has an account if birthday is called.
|     emptyWallet(self)
|          Deposits money into account from wallet.
|          Update amount in wallet.
|          Assume a person has an account if emptyWallet is called.
|          Return None
```

Grading Rubric

BankAccount	
- `__init__`	5
- `deposit`	5
- `withdraw`	5
- `batchDeposit`	15
- `transfer`	15
- `giveToCharity`	20
- `__lt__`	5
- `__gt__`	5
- `__eq__`	5
Person	
- `__init__`	5
- `startAccount`	5
- `birthday`	5
- `emptyWallet`	5

Provided

The following file(s) have been provided to you. There are several, but you will only edit one of them:

1. `bank_account.py`

This is the file you will edit and implement. All instructions for what the methods should do are in the docstrings.

2. `hw_9_test.py`

In this file we have created a series of tests for your usage. We understand you probably have never been exposed to testing code so you are not expected to do anything to this file or even use this file if you don't want to. However, we encourage you to use it as it will be highly beneficial in testing your code. Feel free to add your own tests to the file to cover any additional cases you would like to test.
If you do desire to test your code, all you have to do is have the bank_account.py and the hw_9_test.py files in the same directory. Open and run hw_9_test.py. After running the test, you should see the results. Check the results and start debugging if needed. Read more about unittest here: [https://docs.python.org/3/library/unittest.html]

Disclaimer: **The tests found in hw_9_test.py are not intended to be an exhaustive list of all test cases and does not guarantee any type of grade. Write your own tests if you wish to ensure you cover all edge cases.**

3. `file.txt`

A test file that is used in the autograder. You are encouraged to use it in your own testing, but do not modify it or the autograder could fail.

4. `file2.txt`

Same as above.

Deliverables

You must submit all of the following file(s). Please make sure the filename matches the filename(s) below. Be sure you receive the confirmation email from T-Square, and then download your uploaded files to a new folder and run them.

1. `bank_account.py`

If your file is named something else, you will get a 0%. If this file does not run (if it encounters an error while trying to run), you will get no credit on the assignment.