module4

January 24, 2024

0.1 Module 4

In this assignment, you will implement an online banking system. Users can sign up with the system, log in to the system, change their password, and delete their account. They can also update their bank account balance and transfer money to another user's bank account.

You'll implement functions related to File I/O and dictionaries. The first two functions require you to import files and create dictionaries. User information will be imported from the "users.txt" file and account information will be imported from the "bank.txt" file. Take a look at the content in the different files. The remaining functions require you to use or modify the two dictionaries created from the files.

Each function has been defined for you, but without the code. See the docstring in each function for instructions on what the function is supposed to do and how to write the code. It should be clear enough. In some cases, we have provided hints to help you get started.

[1]:
```
############################################################
### EXECUTE THIS CELL BEFORE YOU TO TEST YOUR SOLUTIONS ###
############################################################

import nose.tools as tools
```

[2]:
```
def import_and_create_bank(filename):
    '''
    This function is used to create a bank dictionary.  The given argument is␣
↪the filename to load.
    Every line in the file should be in the following format:
        key: value
    The key is a user's name and the value is an amount to update the user's␣
↪bank account with.  The value should be a
    number, however, it is possible that there is no value or that the value is␣
↪an invalid number.

    What you will do:
    - Create an empty bank dictionary.
    - Read in the file.
    - Add keys and values to the dictionary from the contents of the file.
    - If the key doesn't exist in the dictionary, create a new key:value pair.
```

```
   - If the key does exist in the dictionary, increment its value with the␣
↪amount.
   - You should also handle the following cases:
   -- When the value is missing or invalid.  If so, ignore that line and don't␣
↪update the dictionary.
   -- When the line is completely blank.  Again, ignore that line and don't␣
↪update the dictionary.
   -- When there is whitespace at the beginning or end of a line and/or␣
↪between the name and value on a line.  You
   should trim any and all whitespace.
   - Return the bank dictionary from this function.

   For example, here's how your code should handle some specific lines in the␣
↪file:
   The 1st line in the file has a name and valid number:
       Brandon: 5
   Your code will process this line and add the extracted information to the␣
↪dictionary.  After it does,
   the dictionary will look like this:
       bank = {"Brandon": 5}

   The 2nd line in the file also has a name and valid number:
       Patrick: 18.9
   Your code will also process this line and add the extracted information to␣
↪the dictionary.  After it does,
   the dictionary will look like this:
       bank = {"Brandon": 5, "Patrick": 18.9}

   The 3rd line in the file has a name but invalid number:
       Brandon: xyz
   Your code will ignore this line and add nothing to the dictionary.  It will␣
↪still look like this:
       bank = {"Brandon": 5, "Patrick": 18.9}

   The 4th line in the file has a name but missing number:
       Jack:
   Your code will ignore this line and add nothing to the dictionary.  It will␣
↪still look like this:
       bank = {"Brandon": 5, "Patrick": 18.9}

   The 5th line in the file is completely blank.
   Your code will ignore this line and add nothing to the dictionary.  It will␣
↪still look like this:
       bank = {"Brandon": 5, "Patrick": 18.9}
```

```
    The 8th line in the file has a name and valid number, but with extra␣
↪whitespace:
        Brandon:       10
    Your code will process this line and update the value associated with the␣
↪existing key ('Brandon') in the dictionary.
    After it does, the value associated with the key 'Brandon' will be 10:
        bank = {"Brandon": 15, ...}

    After processing every line in the file, the dictionary will look like this:
        bank = {"Brandon": 115.5, "Patrick": 18.9, "Sarah": 827.43, "Jack": 45.
↪0, "James": 128.87}
    Return the dictionary from this function.
    '''

    bank = {}

    # your code here
    with open(filename, 'r') as file:
        for line in file:
            lst = line.strip().split(':')
            if len(lst) <= 1:
                continue
            username = lst[0].strip()
            value = lst[1].strip()
            try:
                value = float(value)
                bank[username] = bank.get(username,0) + value
            except:
                continue

    return bank
```

[3]:
```
############################
### TEST YOUR SOLUTION ###
############################
bank = import_and_create_bank("bank.txt")

tools.assert_false(len(bank) == 0, "The bank dictionary is not supposed to be␣
↪empty after reading 'bank.txt'.")
tools.assert_almost_equal(115.5, bank.get("Brandon"), msg="The total amount for␣
↪Brandon is incorrect. It should be 115.5.")
tools.assert_almost_equal(128.87, bank.get("James"), msg="The total amount for␣
↪James is incorrect. It should be 128.87.")
tools.assert_is_none(bank.get("Joel"), "Joel should not be in the bank␣
↪dictionary.")
```

```
tools.assert_is_none(bank.get("Luke"), "Luke should not be in the bank␣
↪dictionary.")
tools.assert_almost_equal(827.43, bank.get("Sarah"), msg="The total amount for␣
↪Sarah is incorrect. It should be 827.43.")
print("Success!")
```

```
Success!
```

[4]:
```
def signup(user_accounts, log_in, username, password):
    '''
    This function allows users to sign up.
    If both username and password meet the requirements:
    - Updates the username and the corresponding password in the user_accounts␣
↪dictionary.
    - Updates the log_in dictionary, setting the value to False.
    - Returns True.

    If the username and password fail to meet any one of the following␣
↪requirements, returns False.
    - The username already exists in the user_accounts.
    - The password must be at least 8 characters.
    - The password must contain at least one lowercase character.
    - The password must contain at least one uppercase character.
    - The password must contain at least one number.
    - The username & password cannot be the same.

    For example:
    - Calling signup(user_accounts, log_in, "Brandon", "123abcABCD") will␣
↪return False
    - Calling signup(user_accounts, log_in, "BrandonK", "123ABCD") will return␣
↪False
    - Calling signup(user_accounts, log_in, "BrandonK","abcdABCD") will return␣
↪False
    - Calling signup(user_accounts, log_in, "BrandonK", "123aABCD") will return␣
↪True. Then calling
    signup(user_accounts, log_in, "BrandonK", "123aABCD") again will return␣
↪False.

    Hint: Think about defining and using a separate valid(password) function␣
↪that checks the validity of a given password.
    This will also come in handy when writing the change_password() function.
    '''

    # your code here
    def valid(password):
        if len(password) < 8:
```

```
            return False
        if password.isalpha() or password.isdigit():
            return False
        if password.lower() == username.lower():
            return False
        if not any(char.islower() for char in password):
            return False
        if not any(char.isupper() for char in password):
            return False
        if not any(char.isdigit() for char in password):
            return False
        return True
    if username in user_accounts:
        return False

    if not valid(password):
        return False

    user_accounts[username] = password
    log_in[username] = False

    return True
```

[5]:
```
def import_and_create_accounts(filename):
    '''
    This function is used to create an user accounts dictionary and another␣
↪login dictionary.  The given argument is the
    filename to load.
    Every line in the file should be in the following format:
      username - password
    The key is a username and the value is a password.  If the username and␣
↪password fulfills the requirements,
    add the username and password into the user accounts dictionary.  To make␣
↪sure that the password fulfills these
    requirements, be sure to use the signup function that you wrote above.

    For the login dictionary, the key is the username, and its value indicates␣
↪whether the user is logged in, or not.
    Initially, all users are not logged in.

    What you will do:
    - Create an empty user accounts dictionary and an empty login dictionary.
    - Read in the file.
    - If the username and password fulfills the requirements, adds the username␣
↪and password
    into the user accounts dictionary, and updates the login dictionary.
    - You should also handle the following cases:
```

```
   -- When the password is missing.  If so, ignore that line and don't update␣
↪the dictionaries.
   -- When there is whitespace at the beginning or end of a line and/or␣
↪between the name and password on a line.  You
   should trim any and all whitespace.
   - Return both the user accounts dictionary and login dictionary from this␣
↪function.

   For example, here's how your code should handle some specific lines in the␣
↪file:
   The 1st line in the file has a name and password:
     Brandon - brandon123ABC
   Your code will process this line, and using the signup function, will add␣
↪the extracted information to the
   dictionaries.  After it does, the dictionaries will look like this:
     user_accounts = {"Brandon": "brandon123ABC"}
     log_in = {"Brandon": False}

   The 2nd line in the file has a name but missing password:
     Jack
   Your code will ignore this line.  The dictionaries will still look like␣
↪this:
     user_accounts = {"Brandon": "brandon123ABC"}
     log_in = {"Brandon": False}

   The 3rd line in the file has a name and password:
     Jack - jac123
   Your code will process this line, and using the signup function, will not␣
↪add the extracted information to the
   dictionaries because the password is invalid.  The dictionaries will still␣
↪look like this:
     user_accounts = {"Brandon": "brandon123ABC"}
     log_in = {"Brandon": False}

   The 4th line in the file has a name and password:
     Jack - jack123POU
   Your code will process this line, and using the signup function, will add␣
↪the extracted information to the
   dictionaries.  After it does, the dictionaries will look like this:
     user_accounts = {"Brandon": "brandon123ABC, "Jack": "jack123POU"}
     log_in = {"Brandon": False, "Jack": False}

   After processing every line in the file, the dictionaries will look like␣
↪this:
     user_accounts = {"Brandon": "brandon123ABC, "Jack": "jack123POU", "James":
↪ "100jamesABD", "Sarah": "sd896ssfJJH"}
```

```
      log_in = {"Brandon": False, "Jack": False, "James": False, "Sarah": False}
    Return the dictionaries from this function.
    '''

    user_accounts = {}
    log_in = {}

    # your code here
    with open(filename, 'r') as file:
        for line in file:
            line = line.strip()
            if not line:
                continue

            if '-' not in line:
                continue

            username, password = line.split('-')
            username = username.strip()
            password = password.strip()

            if len(password) < 8:
                continue

            success = signup(user_accounts, log_in, username, password)
            if success:
                log_in[username] = False

    return user_accounts,log_in
```

[6]:
```
############################
### TEST YOUR SOLUTION ###
############################
user_accounts, log_in = import_and_create_accounts("user.txt")

tools.assert_false(len(user_accounts) == 0, "The user_accounts dictionary is␣
 ↪not supposed to be empty after reading 'user.txt'." )
tools.assert_false(len(log_in) == 0, "The login dictionary is not supposed to␣
 ↪be empty after reading 'user.txt'.")
tools.assert_equal("brandon123ABC", user_accounts.get("Brandon"), "The password␣
 ↪associated with username 'Brandon' is incorrect.")
tools.assert_equal("jack123POU", user_accounts.get("Jack"),"The password␣
 ↪associated with username 'Jack' is incorrect.")
tools.assert_is_none(user_accounts.get("Jennie"), "Jennie should not appear in␣
 ↪user_accounts, since the associated password was invalid.")
tools.assert_false(log_in["Sarah"], "'Sarah' has not logged in yet, so this␣
 ↪should be false initially.")
```

```
print("Success!")
```

Success!

[7]:
```
##########################
### TEST YOUR SOLUTION ###
##########################
bank = import_and_create_bank("bank.txt")
user_accounts, log_in = import_and_create_accounts("user.txt")

tools.assert_false(signup(user_accounts,log_in,"Brandon","123abcABCD"), "When␣
 ↪signing up, if the username already exists in user_accounts, return False.")

tools.assert_false(signup(user_accounts,log_in,"BrandonK","12abCD"), "When␣
 ↪signing up, if the password does not have at least 8 characters, return␣
 ↪False.")
tools.assert_false(signup(user_accounts,log_in,"BrandonK","1234ABCD"), "When␣
 ↪signing up, if the password does not have at least one lowercase character,␣
 ↪return False.")
tools.assert_false(signup(user_accounts,log_in,"BrandonK","abcdABCD"), "When␣
 ↪signing up, if the password does not have at least one number, return False.
 ↪")
tools.assert_false(signup(user_accounts,log_in,"BrandonK","1234abcd"), "When␣
 ↪signing up, if the password does not have at least one uppercase character,␣
 ↪return False.")

tools.assert_false(signup(user_accounts,log_in,"123abcABCD","123abcABCD"),␣
 ↪"When signing up, if the username & password are the same, return False.")

tools.assert_true(signup(user_accounts,log_in,"BrandonK","123aABCD"), "The user␣
 ↪should be able to sign up with username 'BrandonK' and password '123aABCD'.")
tools.assert_false(signup(user_accounts,log_in,"BrandonK","123aABCD"), "Since␣
 ↪'BrandonK' was able to sign up already, 'BrandonK' should not be able to␣
 ↪signup again.")
tools.assert_true("BrandonK" in user_accounts, "BrandonK is not in␣
 ↪user_accounts.")
tools.assert_equal("123aABCD",user_accounts["BrandonK"], "The password␣
 ↪associated with'BrandonK' is incorrect." )
tools.assert_false(log_in["BrandonK"], "'BrandonK' has not logged in yet, so␣
 ↪this should be false initially.")
print("Success!")
```

Success!

[8]:
```
def login(user_accounts, log_in, username, password):
    '''
```

```
    This function allows users to log in with their username and password.
    The user_accounts dictionary stores the username and associated password.
    The log_in dictionary stores the username and associated log-in status.

    If the username does not exist in user_accounts or the password is␣
↪incorrect:
    - Returns False.
    Otherwise:
    - Updates the user's log-in status in the log_in dictionary, setting the␣
↪value to True.
    - Returns True.

    For example:
    - Calling login(user_accounts, "Brandon", "123abcAB") will return False
    - Calling login(user_accounts, "Brandon", "brandon123ABC") will return True
    '''

    # your code here
    if username not in user_accounts or user_accounts[username] != password:
        return False

    log_in[username] = True
    return True
```

[9]:
```
############################
### TEST YOUR SOLUTION ###
############################
bank = import_and_create_bank("bank.txt")
user_accounts, log_in = import_and_create_accounts("user.txt")

tools.assert_false(login(user_accounts, log_in,"Brandon","123abcAB"), "If the␣
↪password is incorrect, the user should not be able to log in.")
tools.assert_true(login(user_accounts, log_in,"Brandon","brandon123ABC"), "If␣
↪the password is correct, the user should be able to log in.")
tools.assert_false(login(user_accounts, log_in,"BrandonK","123abcABC"), "If the␣
↪user is not in user_accounts, return False.")
print("Success!")
```

```
Success!
```

[10]:
```
def update(bank, log_in, username, amount):
    '''
    In this function, you will try to update the given user's bank account with␣
↪the given amount.
    bank is a dictionary where the key is the username and the value is the␣
↪user's account balance.
```

```
    log_in is a dictionary where the key is the username and the value is the␣
↪user's log-in status.
    amount is the amount to update with, and can either be positive or negative.

    To update the user's account with the amount, the following requirements␣
↪must be met:
    - The user exists in log_in and his/her status is True, meaning, the user␣
↪is logged in.

    If the user doesn't exist in the bank, create the user.
    - The given amount can not result in a negative balance in the bank account.

    Return True if the user's account was updated.

    For example, if Brandon has 115.50 in his account:
    - Calling update(bank, log_in, "Brandon", 50) will return False, unless␣
↪"Brandon" is first logged in.  Then it
    will return True.  Brandon will then have 165.50 in his account.
    - Calling update(bank, log_in, "Brandon", -200) will return False because␣
↪Brandon does not have enough in his
    account.
    '''

    # your code here
    if username in log_in and log_in[username]:
        if username not in bank:
            bank[username] = 0

        new_balance = bank[username] + amount

        if new_balance >= 0:
            bank[username] = new_balance
            return True

    return False
```

[11]:
```
############################
### TEST YOUR SOLUTION ###
############################
bank = import_and_create_bank("bank.txt")
user_accounts, log_in = import_and_create_accounts("user.txt")

tools.assert_false(update(bank,log_in,"Jack",100), "When the user is not logged␣
 ↪in, return False." )
login(user_accounts, log_in, "Brandon", "brandon123ABC")
```

```
tools.assert_false(update(bank,log_in,"Brandon",-400), "When the user does not␣
 ↪have enough money in the account, return False.")
tools.assert_true(update(bank,log_in,"Brandon",100), "'Brandon' should be able␣
 ↪to increase the amount in the account by 100.")
tools.assert_almost_equal(bank.get("Brandon"),215.5, msg="After the update, the␣
 ↪total amount for Brandon is incorrect. It should be 215.5.")

signup(user_accounts, log_in, "BrandonK", "123aABCD")
tools.assert_is_none(bank.get("BrandonK"), "'BrandonK' should not be in the␣
 ↪bank dictionary yet.")
login(user_accounts,log_in,"BrandonK","123aABCD")
tools.assert_true(update(bank,log_in,"BrandonK", 100), "'BrandonK' should be␣
 ↪able to increase the amount in the account by 100.")
tools.assert_almost_equal(100, bank.get("BrandonK"), msg="After the update, the␣
 ↪total amount for 'BrandonK' is incorrect. It should be 100.")
print("Success!")
```

```
Success!
```

[12]:
```
def transfer(bank, log_in, userA, userB, amount):
    '''
    In this function, you will try to make a transfer between two user accounts.
    bank is a dictionary where the key is the username and the value is the␣
 ↪user's account balance.
    log_in is a dictionary where the key is the username and the value is the␣
 ↪user's log-in status.
    amount is the amount to be transferred between user accounts (userA and␣
 ↪userB).  amount is always positive.

    What you will do:
    - Deduct the given amount from userA and add it to userB, which makes a␣
 ↪transfer.
    - You should consider some following cases:
      - userA must be in the bank and his/her log-in status in log_in must be␣
 ↪True.
      - userB must be in log_in, regardless of log-in status.  userB can be␣
 ↪absent in the bank.
      - No user can have a negative amount in their account. He/she must have a␣
 ↪positive or zero balance.

    Return True if a transfer is made.

    For example:
    - Calling transfer(bank, log_in, "BrandonK", "Jack", 100) will return False
    - Calling transfer(bank, log_in, "Brandon", "JackC", 100) will return False
```

```
    - After logging "Brandon" in, calling transfer(bank, log_in, "Brandon",␣
↪"Jack", 10) will return True
    - Calling transfer(bank, log_in, "Brandon", "Jack", 200) will return False
    '''

    # your code here
    if userA in log_in and log_in[userA]:
        if userB in log_in:
            balanceA = bank.get(userA, 0)
            balanceB = bank.get(userB, 0)

            if balanceA >= amount >= 0:
                bank[userA] = balanceA - amount
                bank[userB] = balanceB + amount
                return True

    return False
```

[13]:
```
############################
### TEST YOUR SOLUTION ###
############################
bank = import_and_create_bank("bank.txt")
user_accounts, log_in = import_and_create_accounts("user.txt")

tools.assert_false(transfer(bank,log_in,"BrandonK","Jack",100), "'BrandonK' is␣
↪not in the bank or log_in dictionaries yet, so this should return False.")
tools.assert_false(transfer(bank,log_in,"Brandon","JackC",100), "'JackC is not␣
↪in log_in, so this should return false.")
tools.assert_false(transfer(bank,log_in,"Brandon","Jack",100), "'Brandon'␣
↪should be logged in to transfer money, so this should return False.")

login(user_accounts,log_in,"Brandon","brandon123ABC")
tools.assert_false(transfer(bank,log_in,"Brandon","Jack",200), "'Brandon' does␣
↪not have enough money to transfer to 'Jack', so this should return False.")
tools.assert_true(transfer(bank,log_in,"Brandon","Jack",10), "'Brandon' does␣
↪have enough money to transfer to 'Jack', so this should return True.")
tools.assert_almost_equal(105.5, bank.get("Brandon"), msg="After the transfer,␣
↪the total amount for 'Brandon' is incorrect. It should be 105.5.")
tools.assert_almost_equal(55, bank.get("Jack"), msg="After the transfer, the␣
↪total amount for 'Jack' is incorrect. It should be 55.")

signup(user_accounts,log_in,"BrandonK","123aABCD")
tools.assert_is_none(bank.get("BrandonK"), "'BrandonK' should not be in the␣
↪bank dictionary yet.")
login(user_accounts,log_in,"BrandonK","123aABCD")
```

```
tools.assert_true(transfer(bank,log_in,"Brandon","BrandonK",10), "'Brandon' does have enough money to transfer to 'BrandonK', so this should return True.")
tools.assert_almost_equal(bank.get("Brandon"),95.5, msg="After the transfer, the total amount for 'Brandon' is incorrect. It should be 95.5.")
tools.assert_almost_equal(bank.get("BrandonK"),10, msg="After the transfer, the total amount for 'BrandonK' is incorrect. It should be 10.")
print("Success!")
```

```
Success!
```

[14]:
```
def change_password(user_accounts, log_in, username, old_password, new_password):
    '''
    This function allows users to change their password.

    If all of the following requirements are met, changes the password and returns True. Otherwise, returns False.
    - The username exists in the user_accounts.
    - The user is logged in (the username is associated with the value True in the log_in dictionary)
    - The old_password is the user's current password.
    - The new_password should be different from the old one.
    - The new_password fulfills the requirement in signup.

    For example:
    - Calling change_password(user_accounts, log_in, "BrandonK", "123abcABC","123abcABCD") will return False
    - Calling change_password(user_accounts, log_in, "Brandon", "123abcABCD","123abcABCDE") will return False
    - Calling change_password(user_accounts, log_in, "Brandon","brandon123ABC", "brandon123ABC") will return False
    - Calling change_password(user_accounts, log_in, "Brandon","brandon123ABC", c"123abcABCD") will return True

    Hint: Think about defining and using a separate valid(password) function that checks the validity of a given password.
    This will also come in handy when writing the signup() function.
    '''

    # your code here
    def valid(password):
        if (
            len(password) >= 8 and
            any(char.islower() for char in password) and
            any(char.isupper() for char in password) and
```

```
            any(char.isdigit() for char in password)
        ):
            return True
        else:
            return False

    if (
        username in user_accounts and
        log_in.get(username, False) and
        user_accounts[username] == old_password and
        old_password != new_password and
        valid(new_password)
    ):
        user_accounts[username] = new_password
        return True
    else:
        return False
```

[15]:
```
##########################
### TEST YOUR SOLUTION ###
##########################
bank = import_and_create_bank("bank.txt")
user_accounts, log_in = import_and_create_accounts("user.txt")

tools.
 ↪assert_false(change_password(user_accounts,log_in,"BrandonK","123abcABC","123abcABCD"),␣
 ↪"BrandonK is not in user_accounts yet, so this should return False.")
tools.
 ↪assert_false(change_password(user_accounts,log_in,"Brandon","brandon123ABC","123abcABCD"),␣
 ↪"A user must be logged in to change the password.")

login(user_accounts,log_in,"Brandon","brandon123ABC")
tools.
 ↪assert_false(change_password(user_accounts,log_in,"Brandon","123abcABCD","123abcABCDE"),␣
 ↪"The old password entered should be the same as the current password.")

tools.
 ↪assert_false(change_password(user_accounts,log_in,"Brandon","brandon123ABC","brandon123ABC"
 ↪"The new password should be different from the old one.")

tools.
 ↪assert_false(change_password(user_accounts,log_in,"Brandon","brandon123ABC","123ABCD"),␣
 ↪"The new password should be valid.")

tools.
 ↪assert_true(change_password(user_accounts,log_in,"Brandon","brandon123ABC","123abcABCD"),␣
 ↪"This function should return True when there are no issues.")
```

```
tools.assert_equal("123abcABCD",user_accounts["Brandon"], "The password␣
↪associated with 'Brandon' is incorrect.")
print("Success!")
```

Success!

[16]:
```
def delete_account(user_accounts, log_in, bank, username, password):
    '''
    Completely deletes the user from the online banking system.
    If the user exists in the user_accounts dictionary and the password is␣
↪correct, and the user
    is logged in (the username is associated with the value True in the log_in␣
↪dictionary):
    - Deletes the user from the user_accounts dictionary, the log_in␣
↪dictionary, and the bank dictionary.
    - Returns True.
    Otherwise:
    - Returns False.

    For example:
    - Calling delete_account(user_accounts, log_in, bank, "BrandonK",␣
↪"123abcABC") will return False
    - Calling delete_account(user_accounts, log_in, bank, "Brandon",␣
↪"123abcABDC") will return False
    - If you first log "Brandon" in, calling delete_account(user_accounts,␣
↪log_in, bank, "Brandon", "brandon123ABC")
    will return True
    '''

    # your code here
    if (
        username in user_accounts and
        user_accounts[username] == password and
        log_in.get(username, False) is True
    ):
        del user_accounts[username]
        del log_in[username]
        if username in bank:
            del bank[username]
        return True
    else:
        return False
```

[17]:
```
############################
### TEST YOUR SOLUTION ###
############################
```

```
bank = import_and_create_bank("bank.txt")
user_accounts, log_in = import_and_create_accounts("user.txt")

tools.
 ↪assert_false(delete_account(user_accounts,log_in,bank,"BrandonK","123abcABC"),␣
 ↪"'BrandonK' is not in user_accounts yet, so this should return False.")
tools.
 ↪assert_false(delete_account(user_accounts,log_in,bank,"Brandon","brandon123ABC"),␣
 ↪"'Brandon' is not logged in yet, so this should return False.")

login(user_accounts,log_in,"Brandon","brandon123ABC")
tools.
 ↪assert_false(delete_account(user_accounts,log_in,bank,"Brandon","123abcABDC"),␣
 ↪"The password is incorrect, so this should return False." )

tools.
 ↪assert_true(delete_account(user_accounts,log_in,bank,"Brandon","brandon123ABC"),"'Brandon'␣
 ↪should be able to delete the account when there are no issues.")
tools.assert_is_none(user_accounts.get("Brandon"), "After deleting the account,␣
 ↪'Brandon' should not be in user_accounts.")
tools.assert_is_none(log_in.get("Brandon"), "After deleting the account,␣
 ↪'Brandon' should not be in log_in.")
tools.assert_is_none(bank.get("Brandon"), "After deleting the account,␣
 ↪'Brandon' should not be in bank.")
print("Success!")
```

```
Success!
```

[ ]:
```
def main():
    '''
    The main function is a skeleton for you to test if your overall programming␣
 ↪is working.
    Note we will not test your main function. It is only for you to run and␣
 ↪interact with your program.
    '''

    bank = import_and_create_bank("bank.txt")
    user_accounts, log_in = import_and_create_accounts("user.txt")

    while True:
        # for debugging
        print('bank:', bank)
        print('user_accounts:', user_accounts)
        print('log_in:', log_in)
        print('')
        #
```

```
        option = input("What do you want to do?  Please enter a numerical␣
↪option below.\n"
        "1. login\n"
        "2. signup\n"
        "3. change password\n"
        "4. delete account\n"
        "5. update amount\n"
        "6. make a transfer\n"
        "7. exit\n")
        if option == "1":
            username = input("Please input the username\n")
            password = input("Please input the password\n")

            # add code to login
            login(user_accounts, log_in, username, password);
        elif option == "2":
            username = input("Please input the username\n")
            password = input("Please input the password\n")

            # add code to signup
            signup(user_accounts, log_in, username, password)
        elif option == "3":
            username = input("Please input the username\n")
            old_password = input("Please input the old password\n")
            new_password = input("Please input the new password\n")

            # add code to change password
            change_password(user_accounts, log_in, username, old_password,␣
↪new_password)
        elif option == "4":
            username = input("Please input the username\n")
            password = input("Please input the password\n")

            # add code to delete account
            delete_account(user_accounts, log_in, bank, username, password)
        elif option == "5":
            username = input("Please input the username\n")
            amount = input("Please input the amount\n")
            try:
                amount = float(amount)

                # add code to update amount
                update(bank, log_in, username, amount)
            except:
                print("The amount is invalid. Please reenter the option\n")
```

```
        elif option == "6":
            userA = input("Please input the user who will be deducted\n")
            userB = input("Please input the user who will be added\n")
            amount = input("Please input the amount\n")
            try:
                amount = float(amount)

                # add code to transfer amount
                transfer(bank, log_in, userA, userB, amount)
            except:
                print("The amount is invalid. Please re-enter the option.\n")
        elif option == "7":
            break;
        else:
            print("The option is not valid. Please re-enter the option.\n")

#This will automatically run the main function in your program
#Don't change this
if __name__ == '__main__':
    main()
```

```
bank: {'Brandon': 115.5, 'Patrick': 18.9, 'Sarah': 827.43, 'Jack': 45.0,
'James': 128.87}
user_accounts: {'Brandon': 'brandon123ABC', 'Jack': 'jack123POU', 'James':
'100jamesABD', 'Sarah': 'sd896ssfJJH'}
log_in: {'Brandon': False, 'Jack': False, 'James': False, 'Sarah': False}
```