Title: Banking app Algorithm

Author: Shaun Clarke

Goal: This program mimics some of the basic functions of a bank.

Steps:

1. Define a class Account:
    - a. This class creates a bank account.
    - b. The constructor takes no parameters and initializes the following.
        - i. Owner first name
        - ii. Owner last name
        - iii. Ssn
        - iv. Balance
        - v. Account number
        - vi. Pin
    - c. Define a dunder method def __eq__(self, other_account_number: str) -> bool:
        - i. This method will allow us to compare an object.name attribute to a name string.
        - ii. If the account we are comparing is an instance of Account:
            1. if the account name is equal to the string name.
                - a. return true
            2. otherwise return false.
    - d. Define a method get_owner_first_name(self) -> str:
        - i. This method returns the owner's first name.
    - e. Define a method set_owner_first_name(self, first_name: str) -> bool:
        - i. This method updates the account owner’s first name and returns a Boolean value.
        - ii. Set owner’s first_name to first_name.
        - iii. If it was not updated:
            1. Return False
        - iv. If it was updated:
            1. Return True
    - f. Define a method get_owner_last_name(self) -> str:
        - i. This method returns the owner's last name.
    - g. Define a method set_owner_last_name(self, last_name: str) -> bool:
        - i. This method updates the account owner’s last name and returns a Boolean value.
        - ii. Set owner’s last_name to last_name.

        iii. If it was not updated:
            1. Return False
        iv. If it was updated:
            1. Return True

h. Define a method def get_ssn(self) -> str:
    i. This method returns the account holder's SSN

i. Define a method set_ssn(self, ssn: str) -> bool:
    i. This method updates the account holder's SSN and returns a boolean value.
j. Define a method get_balance(self) -> int:
    i. This method returns the account holder’s balance as a float.
k. Define a method set_balance(self, amount: float) -> bool:
    i. This method updates the account holder’s balance and returns a Boolean value.
    ii. Set the private attribute balance to equal to balance.
    iii. If it was not updated:
        1. Return False
    iv. If it was updated:
        1. Return True
l. Define a method get_pin(self) -> int:
    i. This method returns the users pin as an int.
m. Define a method set_pin(self, pin: int) -> bool:
    i. This method update’s the account holder’s pin number and returns a Boolean value.
    ii. Update the private pin attribute to equal to pin
    iii. If it was not updated:
        1. Return False
    iv. If it was updated:
        1. Return True
n. Define a method get_account_number(self) -> int:
    i. This method returns the user's account number as an int.
o. Define a method set_account_number(self, account_number: int) -> bool:
    i. This method update’s the account holder’s account number and returns a Boolean value.
    ii. Update the account number attribute to equal to account_number
    iii. If it was not updated:
        1. Return False
    iv. If it was updated:
        1. Return True
p. Define a method deposit(self, amount: int) -> int:

  - i. This method adds the entered amount to the account holder’s balance and returns the updated balance:
  - ii. Add present balance to a variable called previous_balance.
  - iii. Update present balance by adding the deposit amount to it.
  - iv. if the new balance minus the added amount equals to the previous_balance variable.
    1. Return the present balance.
  - v. Else:
    1. Return false

- q. Define a method withdraw(self, amount: int) -> int:
  - i. This method subtracts the entered amount the the account holders present balance.
  - ii. If the present balance is less than the entered amount.
    1. Return a message telling the user they have insufficient funds.
  - iii. Otherwise.
  - iv. Add present balance to a variable called previous_balance.
  - v. Update present balance by subtracting the withdrawal amount from it.
  - vi. if the new balance plus the withdrawal amount equals to the previous_balance variable.
    1. Return the present balance.
  - vii. Else:
    1. Return false
- r. Define a method is_pin_valid(self, pin: str) -> bool:
  - i. This method checks if a pin is valid and returns a Boolean value.
  - ii. If the account holders pin matches the entered pin.
    1. Return True
  - iii. Else:
    1. Return false.
- s. Define a method __tostring(self) -> str:
  - i. This meth returns the account holders information asa formatted string.
  - ii. Convert the account holder’s present balance from cents to dollars
  - iii. Return:
    1. Account number
    2. First name
    3. Last name
    4. Ssn
    5. Pin
    6. Balance
- t. Define a dunder method __repr__(self) -> str:
  - i. This method calls the __tostring method and displays the account information when the object is printed.

        ii. Return the output from __tostring()
2. Define a class Bank:
    a. This class interacts with the account through some basic banking functions to create an accountholder.
    b. The constructor takes no parameters and initilzes the following:
        i. An accounts list
        ii. And a variable to hold the total number of accounts
    c. Define a private method __does_account_exist(self, account: Account) -> bool:
        i. This method prevents duplicate accounts by checking if an account already exists.
        ii. For loop:
            1. If the account exists:
                a. Return True
            2. Else:
                a. Return false
    d. Define a method add_account_to_bank(self, account: Account) -> Union[bool,str]:
        i. his method adds an account object to the accounts list. It also make sure they are no duplicates and only allows 100 accounts.
        ii. If we already have 100 accounts:
            1. Tell the user no more accounts available and return false.
        iii. If the account exist:
            1. Return account already exist
        iv. Otherwise
        v. Add the accountholder object to the bank list
        vi. increment total accounts by 1
        vii. return True
    e. Define a method remove_account_from_bank(self, account: Account) -> bool:
        i. This method removes and account from the bank.
        ii. For loop:
            1. If an account exists:
                a. If said account is the one we want to remove.
                    i. Replace the account in the bank list with None
                    ii. Subtract 1 from total accounts.
                    iii. Return true
                b. Else:
                    i. Return false
        iii. If the loop ends and no condition was met.
            1. Return false
    f. Define method find_account(self, account_number: int) -> Account:
        i. This method checks the bank account list and returns the specified account if it exists.

```
            ii. For loop:
                1. If the account exists
                    a. If the account number we are looking for matches
                    b. Return the object for that account
            iii. If the for loop ends and no conditions were satisfied.
                1. Return false
        g. Define a method add_monthly_interest(self, interest_rate: float) -> bool:
            i. This method calculates the monthly interest and adds it to all accounts.
            ii. For loop:
                1. If account exists:
                    a. Get the account balance
                    b. Calculate the interest
                    c. Call the deposit method to add the interest ot the account.
                    d. Convert interest to dollars
                    e. Get balance and convert it to dollars.
                    f. Display the updated balance, interest and account number
    3. Define a class BankUtility:
        a. Create a private class variable set called used numbers
        b. This method is like  swiss army knife. It does everything from prompt user for input to converting dollars to cents.
        c. Define a method get_string_input(self, input_message: str) -> str:
            i. This method gets the user string input and makes sure its not empty.
        d. Define a method prompt_user_for_positive_umber(self, input_message: str) -> Union[float, str]:
            i. This method asks the user for a positive number.
        e. Define a method number_generator(self,minimum: int, maximum: int) -> int:
            i. This method uses the min and max input to generate a random series of numbers.
            ii. While loop:
                1. Use random to generate a number
                2. If the number is not in used numbers and the number doesn't start with 0.
                    a. Add it to the used numbers set
                    b. Return the number
        f. Define a method convert_dollars_and_cents(self, amount: int) -> int:
            i. This method converts dollars to cents.
            ii. Convert dollars to cents
            iii. Return cents
        g. Define a static method is_numeric(numberToCheck) -> bool:
            i. This method check if an input is a digit or string.
            ii.   If the number to check is a digit:
                  1.  Return true
            iii.  Else:
                  1.  Return false.
4.  Define a class CoinCollector:
      a.  This class counts coins and deposit the total in cents to the user’s account.
      b.  The constructor raises a type error because this class should not be isntatiated.
      c.  Define a static method parseChange(coins: str) -> Tuple[int, List]:
            i.    Create a dictionary that maps each letter to its number value.
            ii.   Create an invalid coins list
            iii.  Set a coin counter variable to 0
            iv.   For loop:
                  1.  If  the coin doesn’t match any in the dictionary:
                        a.  Add that coin to the invalid coins list
                  2.  Otherwise:
                        a.  Find the number value for the coin in the dict and add
                            the vlue to the coin counter variable.
            v.    Convert the total coins to dollars
            vi.   Return the invalid coins list and the coin counter total.
5.  Define a class BankManager:
      a.  This class ties the program flow together.
      b.  The constructor takes all the other classes as parameters and initializes the
          following:
            i.    Bank object
            ii.   BankUtility Object
            iii.  Account class
            iv.   Coin collector class
            v.    Banking menu list
      c.  Define a method prompt_for_account_and_pin(self, bank_object: Bank) ->
          object:
            i.    This method prompts the user for their account and pin number.
            ii.   While loop:
                  1.  Ask user for account number
                  2.  If the account number is not 8 digits:
                        a.  Ask the user to reenter it.
                  3.  Otherwise
                  4.  Call find_account
                  5.  If the account doesn’t exist:
                        a.  Let the user know
                  6.  If the account was found break the while loop.
            iii.  While loop
                  1.  Ask user for pin number
```

            2. If the pin number is not 4 digits:
                a. Ask the user to reenter it.
            3. Otherwise
            4. Call get_pin()
            5. If the pin entered matches:
                a. Return the account object
            6. If the pin did not match
                a. Let the user know and ask them to try again.

d. Define a method get_menu_number_input(self,input_message: str, menu_options: List) -> int:
    i. This method gets the menu item input the user selects.
    ii. While loop:
        1. Ask user to to select a menu item number
        2. If the input was empty ask the user to try again.
        3. If the user selects a number that is out of menu range.
            a. Ask them to try again
        4. Otherwise
        5. Return the user input.
e. Define a method format_balance_output(self, balance: int) -> str:
    i. This method formats the account balance to display it in dollars with a dollar sign.
    ii. Divide the cents balance by 100 to convert it to dollars.
    iii. Return the f string formatted balance.
f. Define a method calculate_bills(self, amount: int) -> dict:
    i. This method calculates the number of bills in an ATM transaction.
    ii. Create an empty dict to hold bills 5,10,20
    iii. Use floor division to add the amount of times 20 goes into amount tto the dict.
    iv. Amount modulo 20 and save the remainder in the amount variable.
    v. Use floor division to add the amount of times 10 goes into amount_variable to the dict.
    vi. Amount modulo 10 and save the remainder in the amount variable.
    vii. Use floor division to add the amount of times 5 goes into amount_variable to the dict.
    viii. Amount modulo 5 and save the remainder in the amount variable.
    ix. Return the dict
g. Define a method display_menu(self) -> str:
    i. This method displays the menu and returns the user selection.
    ii. Define a border variable
    iii. Print the border
    iv. Print the menu items
    v. Print the border again at the bottom

```
        vi. Call menu_input to get the user input
       vii. Return the menu input
  h. Define a method main(self):
         i. While loop:
              1. Call menu_selection to display the menu
              2. If the user selected 1:
                   a. Get the user first and last name
                   b. While loop:
                        i. Get the users ssn
                       ii. Make sure its 9 digits
                             1. Ask the user to reenter if it is not.
                   c. Create an account object
                   d. Use the input collected to create the user account.
                   e. Generate pin and account number to add them to the
                      user account.
                   f. Add the ccomplete account to the bank list
                   g. If the account was added print confirmation.
              3. If the user selected 2:
                   a. Prompt the user for account and pin.
                   b. Display the account info
              4. If the user selected 3:
                   a. While loop:
                        i. Prompt user for account number and pin.
                       ii. Ask user to enter new pin and make sure it
                           meets criteria.
                      iii. If it meets criteria break the loop
                   b. While loop
                        i. Prompt user to enter new pin again and makes
                           sure it meets criteria.
                       ii. If the new pin matched the second pin.
                      iii. If it does break loop.
                       iv. Update pin
                        v. Display confirmation
              5. If the user selected 4:
              6. Prompt user for account and pin.
                   a. While loop:
                        i. Ask the user to enter the deposit amount
                       ii. If amount is greater than zero:
                             1. Convert the mount to cents
                             2. Deposit the cents amount in the
                                account
                             3. If deposit was completed
```

```
                a. format balance to dollars
                b. Print confirmation
            4. If not ask user to try again
        iii. If balance is 0 or less
            1. Tell user amount cannot be negative.
7. If the user selected 5:
    a. Prompt user for account number and pin to return from account
    b. Prompt user for account number and pin to return to account
    c. While loop:
        i. Ask the user for the amount they want to transfer
        ii. If amount is more than 0:
            1. Convert amount to cents
            2. Withdraw the amount
            3. If insufficient funds:
                a. Let the user know and return to main menu.
            4. Otherwise.
            5. Deposit the amount that was withdrawn to the to account.
            6. If deposit was successful:
                a. Format the balance for the to and from account.
                b. Display confirmation for both accounts.
                c. Break loop.
            7. If not let user know deposit failed
        iii. Let user know amount can tbe negative and try again.
8. If user selected 6:
    a. Prompt user for account and pin
    b. While loop:
        i. Ask the user to enter the withdrawal  amount
        ii. If amount is greater than zero:
            1. Convert the mount to cents
            2. withdraw the cents amount from the account
            3. if insufficient funds let the user know.
                a. Break loop
            4. If withdrawal was completed
```

```
                  a.  format balance to dollars
                  b.  Print confirmation
                  c.  Break loop
              5.  If not ask user to try again
          iii.  If balance is 0 or less
              1.  Tell user amount cannot be negative.
9.  If the user selects 7:
      a.  Prompt user for account number and pin
      b.  If amount is more than 0, >= 5, <= 1000 an ddivisible by
          5:
           i.  Convert dollars to cents
          ii.  Withdraw amount in cents
         iii.  If insufficient funds let the user know
              1.  Break loop
          iv.  If the withdrawal was successful
              1.  Calculate how many bills make up the
                  withdrawal amount.
              2.  Print the number of 5,10,20 make up
                  the withdrawal amount.
              3.  Display new balance.
              4.  Break loop
           v.  Let user know the balance was not updated.
      c.  Let the user know the amount was invalid.
10. If user selects 8:
      a.  Prompt user for account and pin
      b.  While loop:
           i.  Ask the user to enter the coins to be deposited.
          ii.  If the length of the coin string is greater than
               zero:
              1.  Call parse change
              2.  If they are invalid coins:
                  a.  Print them
              3.  Convert the calculated coints to cents
              4.  Deposit the coins
              5.  If deposit was completed
                  a.  format balance to dollars
                  b.  Print confirmation
                  c.  Break loop
              6.  If not ask user to try again
         iii.  If balance is 0 or less
              1.  Tell user amount cannot be negative.
11. If user selects 9:
```

```
            a. Prompt user for account and pin.
            b. While loop:
                i. Get account number.
                ii. Call the remove account from bank method
                iii. If account was removed
                    1. Print confirmation
                    2. Break loop
        12. If user selects 10:
            a. While loop:
                i. Prompt user for the amount of interest they
                   want to deposit.
                ii. If amount is more than 0:
                    1. Call the add monthly interest method
                       with the amount
                    2. Break loop
                iii. If amount is o or less
                    1. Let the user know amount cannot be
                       negative.
        13. If user enters 11:
        14. Exit program
6. Instantiate class BankManager(Bank, BankUtility, Account, CoinCollector)
7. Run the program.
```