```
"""
Michael Pszonka
BIA-660
Homework 2
A module/script to simulate a basic ATM.
Author: John Doe III., johndoethird@megacorp.com
Notes: Script works only for one user, one pin, but modules can
be used for other cases.  This is been debugged ~Michael
"""

def authenticated_pin(user_pin, entered_pin, username="Valuable
Customer"):
    """Returns whether the enter pin is correct. We are ignoring the
username for now"""

    if user_pin == None:
        return False  # Every user must have pin number on file in the
system

    if len(str(user_pin)) != 4:
        return False  # User pins must be four digits


    if username == "_system" and user_pin == entered_pin:
        return False

    if user_pin == entered_pin:
        return True  # otherwise, default falls through to None
    else:
        return False # this is a placeholder to log incorrect attempts,
etc.
        # you can even use something like "return user_pin ==
entered_pin", but less flexibility to make changes

def valid_withdrawal_request(current_balance, requested_withdrawal,
grace=0):
    """
    Returns whether the requested amount is valid (less than or equal to
allowed amount)
    Grace is 0 and is a fraction of additional amount (a line of credit)
    So, if grace is 0.1, we can withdraw 110% of balance.
    If grace is negative, say -0.2, it means we can withdraw less than
the balance (only 80% in case grace = -0.2)
    """

    if requested_withdrawal < 0:
        return False  # we will not allow a user to try and make a
negative withdrawal

    if requested_withdrawal > 50000:
        return False # we will not allow a user to make a withdrawal of
more then 50,000 through an ATM

    if requested_withdrawal >= 2 * current_balance:
        return False # we will not allow users to withdraw more then
twice there balance, regardless of the value of grace

    if requested_withdrawal <= current_balance * ( 1 + grace):
        return True  # otherwise, default falls through to None
    else:
        return False #
        # Here, you can log the attempts, so we can later on send "zero-
interest credit cards to them"!

def welcome_greeting(username, user_pin):
    """Welcome greeting for the ATM, handles the initial input part"""
    print("Welcome to MegaCorp ATM")
    PIN_attempt = input("Please Enter your PIN: ") # We assume the
username is known based on the card
    PIN_attempt = int(PIN_attempt) # We should use a try-catch part here,
but that will be later
    if not authenticated_pin(user_pin, PIN_attempt, username):
        print("Invalid PIN.")
        return False
    else:
        return True

def process_withdrawal_request(username, current_balance, grace):
    """Processes withdrawal request for a given username"""
    print("Welcome {}".format(username))
    amount_to_withdraw = input("How much would you like to withdraw? ")
    amount_to_withdraw = int(amount_to_withdraw)
    if not valid_withdrawal_request(current_balance, amount_to_withdraw,
grace):
        print("The amount you requested ${}, is too much.
Sorry!".format(amount_to_withdraw))
        return False
    disburse_cash(username, current_balance, amount_to_withdraw)
    return True

def disburse_cash(username, current_balance, amount_to_withdraw):
    """Give requested amount of cash to user... This is called after all
checks are done"""
    user_balance = current_balance - amount_to_withdraw
    print("Disbursing ${}".format(amount_to_withdraw))
    print("Remaining Balance is ${}".format(user_balance))
    print("Ending transaction.")
    # This is where the updating of user account database will be useful

def Tests_for_ATM_simulator():
   """some tests that anyone can write and provide to the main coder"""


   # With default grace=0, user cannot withdraw more than balance
   assert not valid_withdrawal_request(current_balance=1000,
requested_withdrawal=1000.1, grace=0)
   assert valid_withdrawal_request(current_balance=1000,
requested_withdrawal=1000.1, grace=0.1)


   # Normally, you would have a lot of test cases (in many cases, we
should write test cases before we write code!)

   # Here is one test case that business can provide,
   # There is no way the system should let anyone get more than
   # twice their balance [so, grace code needs to later have some checks]
   assert not valid_withdrawal_request(current_balance=1000,
requested_withdrawal=10000, grace=1000)

   # The system user cannot login at any ATM even if the pin is correct
   assert not authenticated_pin(user_pin=5678, entered_pin = 5678,
username="_system")


   #    MY TESTS

   # A user will not be able to make a negative withdrawal
   assert not valid_withdrawal_request(current_balance = 5000,
requested_withdrawal = -500, grace = 0)

   # This particular user will not be able to withdraw any funds if they
have a grace value of negative 1 (or more)
   assert not valid_withdrawal_request(current_balance = 100,
requested_withdrawal = 25, grace = -1)

   # No user regardless or their balance, will be able to withdraw more
then 50,000 from an ATM
   assert not valid_withdrawal_request(current_balance = 1000000,
requested_withdrawal= 50001, grace = 1)
   assert valid_withdrawal_request(current_balance = 1000000,
requested_withdrawal= 50000, grace = 1)

   # Every user must have a valid pin number on file in the system
   assert not authenticated_pin(user_pin = None, entered_pin = 5678,
username = "Michael Pszonka")

   #All users must have a pin number that is exactly four digits.
   assert not authenticated_pin(user_pin = 123, entered_pin = 123)
   assert not authenticated_pin(user_pin = 12345, entered_pin = 12345)
   assert authenticated_pin(user_pin = 1234, entered_pin = 1234)

   print("Tests Passed!")

def main():
    """A main function we want to call when we run this as a script"""
    user_name = "Valuable Customer 5678"
    user_PIN = 5678
    user_balance = 5000 # Normally, these are obtained from a database
    grace = 0.1
    if not welcome_greeting(user_name, user_PIN):
        return False # Failed because of invalid PIN
    if not process_withdrawal_request(user_name, user_balance, grace):
        return False # Failed because of invalid amount of withdrawal
    print("Success!")

if __name__ == "__main__":
    # Python has an internal variable that indicates whether this has
been called as a script or not
    import sys # we want this to see what command line options were
passed
    # sys.argv is a list where first element is the program name and rest
are what we pass as arguments
    if len(sys.argv) == 2 and sys.argv[1] == "--test":  # valid if we run
as python ex1.py --test
        Tests_for_ATM_simulator()
    else:
        main() # We run the main process
```