<html><head></head><body><pre style="word-wrap: break-word; white-space: pre-wrap;">#
Practice: ATM Interface

Save your solution in a directory in `practice/` named `atm-interface`.

An account will be a class named `Account` in a module named `account`: it will have
private attributes for the balance and interest rate.
Remember to underscore `_` prefix any private attributes.
A newly-instantiated account will have zero balance and an interest rate of 0.1%.

Write class methods in the account class that:

* `get_funds()` Return account balance
* `deposit(amount)` Deposit to the account
* `check_withdrawal(amount)` Return `True` if large enough balance for a withdrawal
* `withdraw(amount)` Withdraw an allowed amount; raise a `ValueError` if insufficent
balance
* `calc_interest()` Calculate and return interest on the current account balance

I've already written out [some test code](/practice/atm-interface/account_test.py) that
will check that your `Account` class behaves as expected.
Save it as `account_test.py` in your solution directory.
These tests should all pass for your `Account` implementation.
Either run `py.test` from the `atm-interface` directory, or setup a PyCharm test Run
Configuration.

You should still write doctests for your functions that test internal implementation.
E.g. Check that internal variables are set correctly.
My classes can't do that since internal variables are not part of the defined behavior.

Advanced

Write a program that functions as a simple ATM for two accounts:

1. Checking account
1. Savings account

Implement a user interface in a module `main` that lets a user pick each of those actions
for a given account and updates the account.
After each action it will print the balance.

Super Advanced

Adds some advanced features to the account class.

*   Add a function called `get_standing()` have it return a bool with whether the account
has less than $1000 in it.

*   Predatorily charge a transaction fee every time a withdrawal or deposit happens if the
account is in bad standing.

*   Save the account balance to a file after each operation.
    Read that balance on startup so the balance persists across program starts.

*   Add to each account class an account ID number.

*   Allow the user to open more than one account.
    Let them perform all of the above operations by account number.

With each advanced feature, of course add doctests, but also expand the
`TestAccountBehavior` with new behavior test functions.
</pre></body></html>