LEARN PYTHON

Bank Account

In this project, we'll create a Python class that can be used to create and manipulate a personal bank account.

The bank account class you'll create should have methods for each of the following:

- Accepting deposits
- Allowing withdrawals
- Showing the balance
- Showing the details of the account

Note: As with professional software development, you should be saving your code very often. As you code, make sure you click the "Save" button below to save your code/changes. Otherwise, you run the risk of losing all your code!

Let's begin!

Bank Account

1. On line 1, create a `BankAccount` class.

2. Next, add a member variable called `balance` and set it equal to `0`. This will represent the starting balance of any new `BankAccount` object.

3. Add the `__init__()` method that takes the default `self` parameter and an additional `name` parameter. Later, we'll use the `name` parameter to specify who the account belongs to.

4. Inside the `__init__()` method, assign the `self.name` property to the `name` parameter that the method accepts.

5. Well done. This method will make sure that whatever name the user types (when creating an object of this class) will be attributed to that object.

   Next, add a `__repr__()` method that takes the default `self` parameter.

6. The `__repr__()` method defines what represents the object when a user tries to print that object using `print`. Let's add to this method and make it descriptive.

   In the `__repr__()` method, return a message stating who the account belongs to. The message should also include the balance, limited to two decimal places. Use string formatting to complete the message.

7. Cool! The method you just added will return the bank account's information if a user tries to `print` a `BankAccount` object.

   Since printing an object isn't always useful, let's add a method called `show_balance()` that will print just the balance. It should accept the default `self` parameter.

8. On the next line, use string formatting to print the user's balance to two decimal places.

9. Great! We can use that method to print the user's balance.

   Next, let's add a method that allows deposits to the bank account. Add a method called `deposit()` that takes the default parameter, as well as an `amount` parameter.

10. Inside of the `deposit()` method, let's do some error checking. We shouldn't allow a user to deposit less than or equal to zero dollars (that doesn't make sense).

    Add an `if` statement that checks if `amount` is less than or equal to zero. Inside of the statement, print an appropriate error message. Then, on the next line, return.

11. Otherwise, we should print out the amount of the deposit and then increment the user's balance.

    Add a corresponding `else` block. Inside of the `else` block, print a message that displays the amount that the user is depositing. Use string formatting and print only to two decimal places.

12. On the next line, we should increment the user's balance.

    Increment the user's balance using the `+=` operator.

13. On the next line, display the new balance to the user by calling the `show_balance()` method.

    Methods can be called on objects - try to figure out which object the method is called on.

14. If a user can deposit, we should also allow them to withdraw.

    Add a new method called `withdraw()`. It should take the default parameter, as well as an `amount` parameter.

15. Let's do some error checking again. The user should not be allowed to withdraw more than what is currently in their bank account.

    On the next line, add an `if` statement that checks if `amount` is greater than the balance. Inside the statement, print an appropriate error message. Then on the next line, return.

16. Otherwise, we should allow the user to withdraw the funds.

    First, add a corresponding `else` block. Then, add a line that prints the amount that the user is withdrawing. Use string formatting and print the amount to two decimal places.

17. Next, we should update the user's balance.

    On the next line, decrement the balance by the `amount` using the `-=` operator.

18. On the next line, display the user's balance by calling the `show_balance()` method.

19. Perfect. We have minimal functionality and error checking that will allow a user to create an account, deposit to it, and withdraw from it. Let's test it out.

    Outside of any function, at the bottom of your file, create a `BankAccount` object called `my_account` and specify a name (as a string) for the argument.

20. What happens if we try to print the `my_account` object? The `__repr__()` method should handle that case.

    On the next line, print the `my_account` object.

21. Let's look at the difference between `__repr__()` and `show_balance()`.

    On the next line, call the `show_balance()` method on `my_account`.

22. Next, deposit 2000 dollars to `my_account` using the `deposit()` method.

23. Let's make sure the `withdraw()` function works. On the next line, withdraw 1000 dollars from `my_account` using the `withdraw()` method.

24. Finally, let's see if the `__repr__()` method accurately reflects the changes that have happened to the `my_account` object.

    As the last line of code, print the `my_account` object once again.

25. Now it's time to see the results of our class. Make sure that you have saved your code. Then, in the terminal, type the following and hit "Enter" on your keyboard:

```
python bankaccount.py
```

The console should show, in order:

- The bank account's initial information
- The balance
- The deposit (along with the balance)
- The withdrawal (along with the balance)
- The bank account's most recent information

Feel free to add or expand the functionality of the `BankAccount` class. Happy coding!