Symulator bankomatu — lista zadań dla ucznia

Cel projektu

Twoim zadaniem jest stworzenie prostego symulatora bankomatu w Pythonie. Program będzie rozwijany etapami. Najpierw wykonasz wersję podstawową dla jednego użytkownika, a potem rozbudujesz ją o logowanie, wielu klientów i historię operacji.

Nie pisz całego programu od razu. Wykonuj zadania po kolei i testuj program po każdym większym kroku.

Część I — wersja podstawowa

Zadanie 1. Przygotuj zmienne główne programu

Utwórz dwie zmienne:

- `wybor` — będzie przechowywać numer opcji wybranej przez użytkownika,
- `saldo` — będzie przechowywać aktualny stan konta.

Na początku ustaw:

```
wybor = 0
saldo = 0
```

Sprawdź, czy program uruchamia się bez błędów.

Zadanie 2. Stwórz funkcję wyświetlającą menu

Napisz funkcję `glowne_menu()`, która wyświetli użytkownikowi dostępne opcje:

1. Wpłata
2. Wypłata
3. Sprawdzenie stanu konta
4. Zakończ

Funkcja ma tylko wyświetlać tekst. Nie powinna niczego zwracać.

Test: wywołaj funkcję raz pod zmiennymi i sprawdź, czy menu wyświetla się poprawnie.

Zadanie 3. Stwórz funkcję pobierającą wybór użytkownika

Napisz funkcję `pobierz_wybor_klienta()`, która:

- pyta użytkownika o wybór opcji,
- zamienia wpisaną wartość na liczbę całkowitą,
- zwraca tę liczbę.

Przykład działania:

```
twój wybór to: 1
```

Test: zapisz wynik funkcji do zmiennej i wyświetl go za pomocą `print()`.

Zadanie 4. Stwórz główną pętlę programu

Pod zmiennymi głównymi napisz pętlę `while`, która działa tak długo, dopóki użytkownik nie wybierze opcji `4`.

W środku pętli:

1. wyświetl menu,
2. pobierz wybór użytkownika,
3. tymczasowo wypisz wybrany numer na ekranie.

Test: sprawdź, czy menu powtarza się po każdym wyborze i czy program kończy się po wpisaniu `4`.

Zadanie 5. Dodaj instrukcję warunkową obsługującą wybór

W głównej pętli dodaj instrukcję `if / elif / else`.

Program powinien rozpoznawać wybory:

- `1` — wpłata,
- `2` — wypłata,
- `3` — sprawdzenie salda,
- `4` — zakończenie programu,
- inna liczba — komunikat o błędnym wyborze.

Na razie w miejscach, gdzie nie ma jeszcze gotowej funkcji, użyj `pass`.

Test: wpisz kilka różnych liczb i sprawdź, czy program trafia do odpowiednich przypadków.

Zadanie 6. Stwórz funkcję pobierającą kwotę

Napisz funkcję `pobierz_kwote(tekst)`, która:

- przyjmuje parametr `tekst`,
- wyświetla ten tekst jako pytanie do użytkownika,
- pobiera liczbę,
- zamienia ją na `float`,
- zwraca pobraną kwotę.

Funkcja ma działać zarówno dla wpłaty, jak i wypłaty.

Test:

```
kwota = pobierz_kwote("Podaj kwotę: ")
print(kwota)
```

Zadanie 7. Stwórz funkcję pokazującą stan konta

Napisz funkcję `pokaz_stan_konta(saldo)`, która wyświetla aktualny stan konta.

Przykład komunikatu:

```
Stan konta wynosi 150.0 złotych
```

Test: wywołaj funkcję z kilkoma różnymi wartościami salda.

Zadanie 8. Stwórz funkcję wpłaty

Napisz funkcję `wplata(saldo)`, która:

1. pyta użytkownika, ile chce wpłacić,
2. dodaje tę kwotę do salda,
3. pokazuje nowy stan konta,
4. zwraca nowe saldo.

Następnie podłącz tę funkcję do opcji `1` w menu:

```
saldo = wplata(saldo)
```

Test:

- uruchom program,
- wybierz wpłatę,

- wpisz kwotę,
- sprawdź, czy saldo się zwiększyło.

Zadanie 9. Stwórz funkcję wypłaty

Napisz funkcję `wyplata(saldo)`, która:

1. pyta użytkownika, ile chce wypłacić,
2. sprawdza, czy na koncie jest wystarczająco dużo pieniędzy,
3. jeśli środków jest za mało — wyświetla komunikat o błędzie i zwraca stare saldo,
4. jeśli środków wystarczy — odejmuje kwotę od salda,
5. wyświetla komunikat o wypłacie,
6. zwraca nowe saldo.

Podłącz funkcję do opcji `2` w menu.

Test:

- wpłać 100 zł,
- wypłać 40 zł,
- sprawdź saldo,
- spróbuj wypłacić więcej, niż masz na koncie.

Zadanie 10. Podłącz sprawdzanie stanu konta

W opcji `3` wywołaj funkcję `pokaz_stan_konta(saldo)`.

Test:

- wpłać pieniądze,
- wybierz sprawdzenie salda,
- wypłać część pieniędzy,
- ponownie sprawdź saldo.

Zadanie 11. Obsłuż zakończenie programu

W opcji `4` wyświetl komunikat:

```
Wyłączanie bankomatu
```

Program powinien zakończyć działanie po wybraniu tej opcji.

Zadanie 12. Przetestuj wersję podstawową

Sprawdź następujące scenariusze:

1. Użytkownik wpłaca 100 zł.
2. Użytkownik sprawdza saldo.
3. Użytkownik wypłaca 30 zł.
4. Użytkownik próbuje wypłacić 1000 zł.
5. Użytkownik wpisuje niepoprawną opcję, np. `9`.
6. Użytkownik kończy program.

Jeżeli wszystko działa, masz gotową wersję podstawową.

Część II — rozszerzenie: karta i PIN

Zadanie 13. Dodaj dane logowania

Dodaj do programu dwie stałe:

```
KARTA = "0001"
PIN = "1234"
```

Będą one oznaczały poprawny numer karty i poprawny PIN.

Zadanie 14. Stwórz funkcję pobierającą dane logowania

Napisz funkcję `pobierz_dane(dana)`, która:

- przyjmuje parametr `dana`,
- pyta użytkownika o podanie numeru karty albo PIN-u,
- zwraca wpisany tekst.

Przykład:

```
podana_karta = pobierz_dane("karty")
podany_pin = pobierz_dane("PIN")
```

Zadanie 15. Stwórz funkcję sprawdzającą zgodność danych

Napisz funkcję `sprawdz_zgodnosc_danych(baza, pobrane)`, która:

- porównuje wartość zapisaną w programie z wartością wpisaną przez użytkownika,
- zwraca `True`, jeśli dane są takie same,

- zwraca `False`, jeśli dane są różne.

Przykład:

```
sprawdz_zgodnosc_danych(KARTA, podana_karta)
```

Zadanie 16. Zablokuj dostęp bez poprawnego logowania

Przed główną pętlą programu pobierz numer karty i PIN.

Następnie dodaj warunek:

- jeśli karta i PIN są poprawne — uruchom główną pętlę bankomatu,
- jeśli karta lub PIN są błędne — wyświetl komunikat `Błąd logowania`.

Pamiętaj, że główna pętla programu musi zostać przesunięta o jedno wcięcie, żeby znalazła się wewnątrz `if`.

Test:

1. Wpisz poprawną kartę i PIN.
2. Wpisz poprawną kartę i błędny PIN.
3. Wpisz błędną kartę i poprawny PIN.
4. Wpisz błędną kartę i błędny PIN.

Część III — rozszerzenie: wielu klientów

Zadanie 17. Zamień pojedynczą kartę i PIN na bazę klientów

Zakomentuj albo usuń zmienne:

```
KARTA = "0001"
PIN = "1234"
```

Utwórz listę klientów:

```
KLIENCI = [
    ["0001", "1234", 0],
    ["0002", "1111", 120],
    ["0003", "3232", 1223.33]
]
```

Każdy klient ma trzy dane:

1. numer karty,
2. PIN,
3. saldo.

Zadanie 18. Przerób funkcję sprawdzającą logowanie

Zmień funkcję `sprawdz_zgodnosc_danych`.

Nowa funkcja powinna przyjmować:

```
sprawdz_zgodnosc_danych(baza, karta, pin)
```

Jej zadanie:

1. przejść przez listę klientów,
2. znaleźć klienta o podanym numerze karty,
3. sprawdzić, czy PIN tego klienta jest poprawny,
4. zwrócić `True` albo `False`.

Test:

- sprawdź logowanie dla każdego klienta z listy,
- sprawdź błędny PIN,
- sprawdź kartę, której nie ma w bazie.

Zadanie 19. Stwórz funkcję pobierającą saldo klienta

Napisz funkcję `pobierz_stan_konta(baza, karta)`, która:

1. przechodzi przez listę klientów,
2. znajduje klienta o podanym numerze karty,
3. zwraca jego saldo.

Jeśli nie znajdzie klienta, może zwrócić `0`.

Po udanym logowaniu ustaw saldo użytkownika:

```
saldo = pobierz_stan_konta(KLIENCI, podana_karta)
```

Test:

- zaloguj się jako klient `0001`,
- potem jako klient `0002`,

- potem jako klient `0003`,
- sprawdź, czy każdy zaczyna z innym saldem.

Część IV — rozszerzenie: historia operacji

Zadanie 20. Dodaj historię do każdego klienta

Zmień listę klientów tak, aby każdy klient miał dodatkową pustą listę na historię operacji.

Przykład:

```
KLIENCI = [
    ["0001", "1234", 0, []],
    ["0002", "1111", 120, []],
    ["0003", "3232", 1223.33, []]
]
```

Od tej chwili dane klienta mają kolejność:

1. numer karty,
2. PIN,
3. saldo,
4. historia operacji.

Zadanie 21. Dodaj nową opcję w menu

W funkcji `glowne_menu()` dodaj opcję:

```
5. Wyświetlenie historii operacji
```

Uważaj: program dalej powinien kończyć się po wybraniu opcji `4`.

Zadanie 22. Stwórz funkcję aktualizującą historię klienta

Napisz funkcję `aktualizuj_historie_klienta(baza, karta, operacja)`, która:

1. przechodzi przez bazę klientów,
2. znajduje klienta o podanym numerze karty,
3. dopisuje nazwę operacji do jego historii.

Przykładowe operacje:

- `"Wpłata"`,

- `"Wypłata"`,
- `"Wyświetlenie salda"`.

Zadanie 23. Dopisuj operacje do historii

W głównej pętli programu dopisz aktualizację historii:

- po wpłacie dopisz `"Wpłata"`,
- po wypłacie dopisz `"Wypłata"`,
- po sprawdzeniu salda dopisz `"Wyświetlenie salda"`.

Test:

1. Zaloguj się poprawnie.
2. Wykonaj wpłatę.
3. Wykonaj wypłatę.
4. Sprawdź saldo.
5. Na razie historia jeszcze się nie wyświetla, ale operacje powinny być dopisywane do listy.

Zadanie 24. Stwórz funkcję pokazującą historię klienta

Napisz funkcję `pokaz_historie_klienta(baza, karta)`, która:

1. znajduje klienta o podanym numerze karty,
2. przechodzi przez jego listę operacji,
3. wyświetla operacje w kolejności wykonania,
4. numeruje operacje od `1`.

Przykład wyniku:

```
1. Wpłata
2. Wypłata
3. Wyświetlenie salda
```

Zadanie 25. Podłącz historię do menu

W głównej pętli programu dodaj obsługę opcji `5`.

Po wybraniu tej opcji program powinien wyświetlić historię operacji aktualnie zalogowanego klienta.

Test:

1. Zaloguj się.
2. Wykonaj kilka operacji.
3. Wybierz opcję `5`.

4. Sprawdź, czy historia pokazuje właściwe operacje we właściwej kolejności.

Część V — zadania dodatkowe

Zadanie dodatkowe 1. Zabezpiecz wpłatę przed kwotą ujemną

Zmień funkcję `wplata(saldo)` tak, aby użytkownik nie mógł wpłacić kwoty ujemnej ani zera.

Jeśli poda niepoprawną kwotę, program powinien wyświetlić komunikat i zwrócić stare saldo.

Zadanie dodatkowe 2. Zabezpiecz wypłatę przed kwotą ujemną

Zmień funkcję `wyplata(saldo)` tak, aby użytkownik nie mógł wypłacić kwoty ujemnej ani zera.

Zadanie dodatkowe 3. Popraw komunikaty programu

Zadbaj o czytelne komunikaty, np.:

- dodaj spacje po pytaniach,
- dodaj informację o kwocie wpłaty,
- po każdej operacji pokazuj aktualne saldo.

Zadanie dodatkowe 4. Dodaj kwoty do historii operacji

Zamiast zapisywać tylko:

```
Wpłata
```

zapisuj dokładniejszą informację:

```
Wpłata: 100 zł
Wypłata: 40 zł
Wyświetlenie salda: 60 zł
```

To zadanie może wymagać zmiany funkcji `wplata()` i `wyplata()`, żeby łatwiej było zapamiętać wykonaną kwotę.

Zadanie dodatkowe 5. Aktualizuj saldo klienta w bazie

Po wpłacie albo wypłacie zmienia się zmienna `saldo`, ale warto też zaktualizować saldo zapisane w liście `KLIENCI`.

Stwórz funkcję, która:

1. znajdzie klienta po numerze karty,
2. wpisze nowe saldo do jego danych.

Przykładowa nazwa funkcji:

```
aktualizuj_saldo_klienta(baza, karta, nowe_saldo)
```

Końcowy test programu

Program uznajemy za gotowy, jeśli działa taki scenariusz:

1. Użytkownik wpisuje numer karty.
2. Użytkownik wpisuje PIN.
3. Jeśli dane są błędne, program pokazuje błąd logowania.
4. Jeśli dane są poprawne, program pokazuje menu bankomatu.
5. Użytkownik może wpłacić pieniądze.
6. Użytkownik może wypłacić pieniądze, ale nie więcej niż ma na koncie.
7. Użytkownik może sprawdzić saldo.
8. Użytkownik może wyświetlić historię operacji.
9. Użytkownik może zakończyć program.
10. Program poprawnie reaguje na niepoprawny wybór z menu.