Домашнее задание

Дисциплина	Python для инженерии данных
Тема	Темы 1-5
Форма проверки	**Проверяет преподаватель**
Имя преподавателя	Дарья Погудина
Время выполнения	6 часов
Цель задания	Отработать навыки разработки программ на Python
Инструменты для выполнения ДЗ	Jupyter Notebook или Google Colab
Правила приёма работы	Прикрепите в LMS ссылку на выполненное задание в Google Colab или GitHub (если вы использовали Jupyter Notebook). **Важно:** убедитесь, что по ссылке есть доступ в Google Colab (иногда в Google Colab нет доступа для другого логина)
Критерии оценки	**Максимальное количество баллов за задание — 10 баллов** **Задание считается выполненным, если:** - прикреплена ссылка на файл с выполненным заданием; - доступ к файлу открыт; - код даёт правильный ответ к задаче. **Задание не выполнено, если:** - файл с заданием не прикреплён или отсутствует доступ по ссылке; - код выдаёт ошибку или даёт неправильный ответ
Дедлайн	**14 дней после даты проведения вебинара**

Задание 1.

Реализуйте базовый класс Account, который моделирует поведение банковского счёта. Этот класс должен не только выполнять базовые операции, но и вести детальный учёт всех действий, а также предоставлять аналитику по истории операций.

Этап 1. Реализация базового класса Account

Класс должен быть инициализирован с параметрами:

- account_holder (str) — имя владельца счёта;
- balance (float, по умолчанию 0) — начальный баланс счёта, не может быть отрицательным.

Атрибуты:

- _account_counter — приватный атрибут для хранения количества созданных счетов. Отсчет начинается с 1000;
- holder — хранит имя владельца;
- account_number — хранит номер счёта;
- _balance — приватный атрибут для хранения текущего баланса;
- operations_history — список или другая структура для хранения истории операций.

Важно: каждая операция должна храниться не просто как число, а как структурированная информация, например, словарь, кортеж или класс. Минимальный набор данных для операции: тип операции ('deposit' или 'withdraw'), сумма, дата и время операции, текущий баланс после операции, статус ('success' или 'fail').

Этап 2. Реализация методов

1. __init__(self, account_holder, balance=0) — конструктор. Обратите внимание, что в конструкторе должен автоматически формироваться номер счёта в формате ‘ACC-XXXX’, где XXXX — порядковый номер счёта;
2. deposit(self, amount) — метод для пополнения счёта:
   - принимает сумму (должна быть положительной), попытка положить отрицательную сумму, должна вызывать исключение;
   - в случае успеха обновляет баланс и добавляет запись в историю операций.
3. withdraw(self, amount) — метод для снятия средств:
   - принимает сумму (должна быть положительной);
   - проверяет, достаточно ли средств на счёте, если нет — операция не проходит, но ее попытка с статусом 'fail' все равно фиксируется в истории;
   - в случае успеха обновляет баланс и добавляет запись в историю.
4. get_balance(self) — метод, который возвращает текущий баланс.
5. get_history(self) — метод, который возвращает историю операций.

Важно: продумайте, в каком формате его вернуть. Для работы с датой и временем используйте модуль datetime. Получить текущее время можно с помощью datetime.now().

Этап 3. Визуализация истории операций. Дополнительное задание для претендующих на оценку 8 и выше баллов (выполняется по желанию).

1. Создайте метод plot_history(self), который использует библиотеку Pandas для создания датафрейма из истории операций.
2. Продумайте, с помощью какой библиотеки можно отобразить изменение баланса с течением времени. Постройте простой линейный график, где по оси X будет время операции, а по оси Y — баланс после каждой операции. График должен иметь заголовок, подписи осей.

Задание 2.

Этап 4. Реализация наследования

1. Реализуйте два класса CheckingAccount (расчётный счёт) и SavingsAccount (сберегательный счёт), которые отражают абстракцию базового поведения банковских аккаунтов:

- наследуются от базового класса Account;
- хранят атрибут класса account_type.

2. Класс SavingsAccount (сберегательный счёт) дополнительно должен реализовывать метод расчёта процентов на остаток apply_interest(self, rate) (например, 7% на остаток).

3. Класс SavingsAccount (сберегательный счёт) позволяет снимать деньги только до определенного порога баланса: нельзя снять больше 50% от баланса. Переопределите метод снятия со счёта.

4. Реализуйте валидацию на отрицательные суммы и корректность имени владельца:

- имя владельца счёта должно быть в формате «Имя Фамилия» с заглавных букв, кириллицей или латиницей, иначе — должно вызываться исключение;
- попытка положить отрицательную сумму должна вызывать исключение.

5. Реализуйте метод для анализа истории транзакций по размеру и дате:

- метод должен выводить последние n крупных операций.

Задание 3. Дополнительное задание для претендущих на оценку 9-10 баллов (выполняется по желанию).

Перед началом работы загрузите из личного кабинете файлы, на которых можно проверить код с «грязными» данными: transactions_dirty.csv и transactions_dirty.json.

Реализуйте для классов аккаунтов CheckingAccount (расчётный счёт) и SavingsAccount (сберегательный счёт) два метода:

- Метод загрузки истории в аккаунт из файла с транзакциями (файл транзакций общий для всех аккаунтов, необходимо учесть фильтрацию загружаемых значений).
- Метод clean_history(), который ищет ошибки в данных перед записью транзакций в историю (опечатки, отрицательные суммы, неверные даты). Обратите внимание, что для SavingsAccount (сберегательный счёт) доступно три типа операции (deposit, withdraw и interest), в то время как для CheckingAccount (расчётный счёт) доступны только два типа операции (deposit, withdraw). Все данные с ошибками считаем невалидными и не записываем в историю операций.
- После загрузки истории операций в аккаунт, баланс счёта должен обновиться.

Критерии оценивания

Критерий	Баллы	Описание
Корректность ООП	2	Класс корректно инициализируется, используются инкапсуляция (приватные атрибуты, геттеры), методы deposit и withdraw работают без ошибок, логически верно обрабатывают неверные входные данные (отрицательные суммы).
История операций и работа с датами	2	История хранится в структурированном виде (словарь/кортеж/объект), каждая операция содержит дату/время, тип, сумму, баланс, статус. get_history() возвращает читаемый формат.
Наследование (CheckingAccount и SavingsAccount)	2	Корректно реализовано наследование от Account. Правильно работают ограничения для SavingsAccount, метод apply_interest, атрибут account_type. Нет дублирования кода.
Валидация и обработка ошибок	1	Проверка формата имени владельца, запрет на отрицательные суммы, корректная обработка недостатка средств.
Визуализация истории операций	1	Создан график изменения баланса с помощью pandas и matplotlib. Есть подписи осей и заголовок.
Работа с «грязными» данными (файлы, очистка)	2	Реализована загрузка из CSV/JSON, фильтрация и очистка (clean_history()), корректное обновление баланса (задание 3).
Чистота кода	-0.5 за каждое нарушение	Код откомментирован, читаемый, соответствует PEP8