ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ Р.МОЛДОВЫ

ФАКУЛЬТЕТ МАТЕМАТИКИ И ИНФОРМАТИКИ

ДЕПАРТАМЕНТ ИНФОРМАТИКИ

**Мамалига Артур**

**Отчет**

по дисциплине „ПРОГРАММИРОВАНИЕ В PYTHON”

Руководитель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Плешка Наталья, лектор

(подпись)

Автор: Мамалига

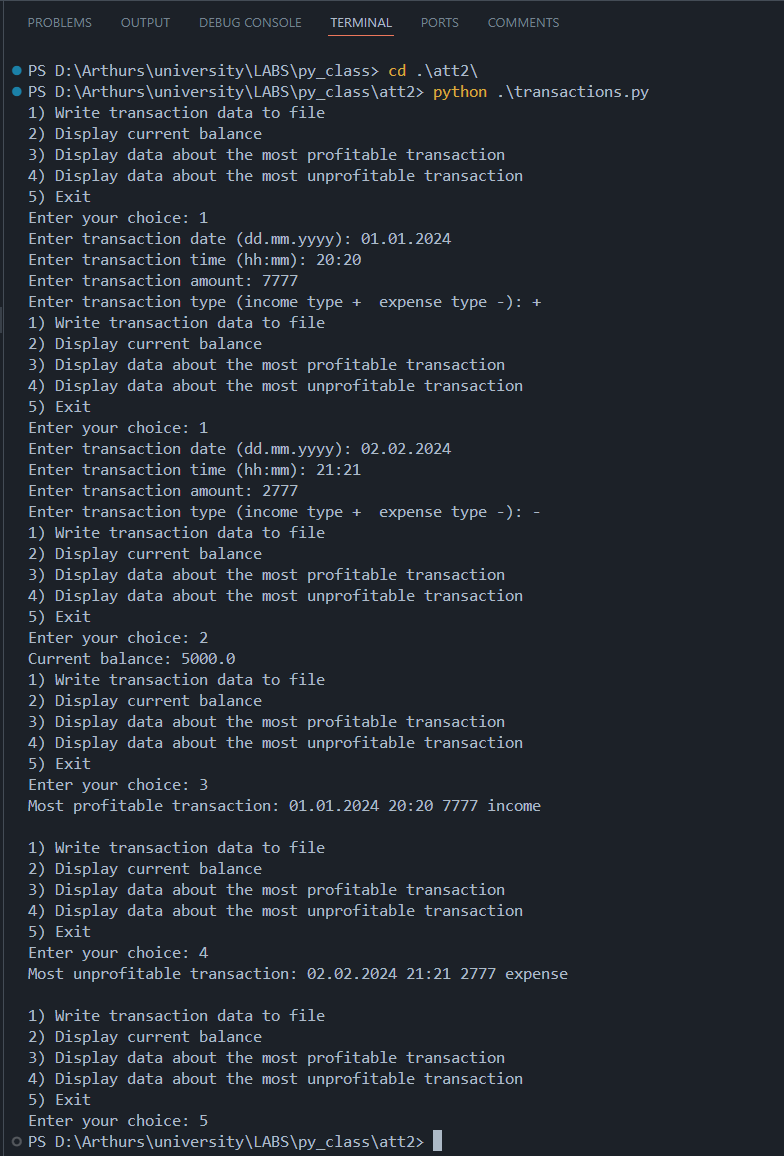
Кишинев, 2024

**Постановка задачи**Программа позволяет пользователю записывать данные о транзакциях в файл, вычислять текущий баланс организации, выводить информацию о самой прибыльной и убыточной транзакциях. Для записи данных в файл и подсчета баланса реализованы проверки на валидность даты, времени и типа операции.

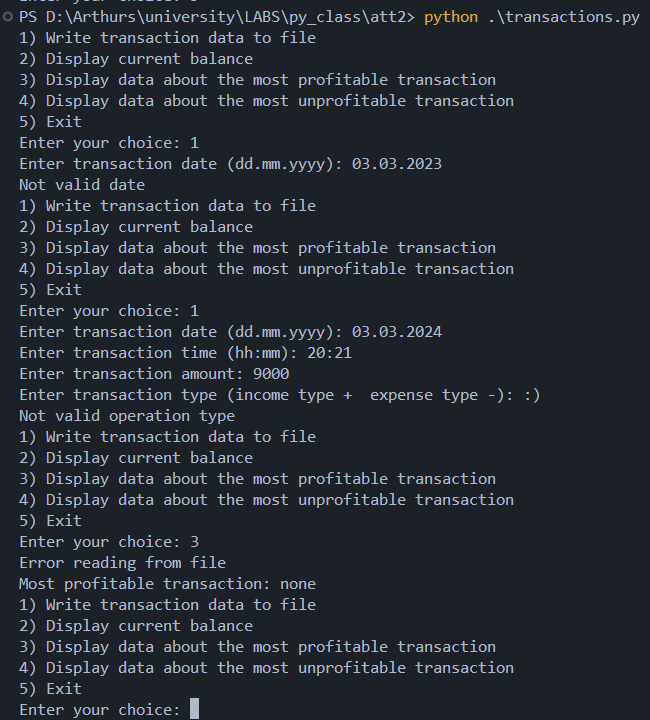
**Логика реализованных алгоритмов**

Программа содержит 2 файла, основной файл с меню и подключенный модуль который реализует логику обработки полученных данных из консоли для обработки данных о транзакциях. Данный модуль содержит следующие функции с нижеописанной логикой работы

1. **is\_valid\_date(date\_str):**
   1. Функция проверяет, является ли строка date\_str валидной датой в формате "%d.%m.%Y".
   2. Проверяется, что дата больше или равна 01.01.2024 и не позже текущей даты.
   3. Если дата валидна, возвращает True, иначе False.
2. **is\_valid\_time(time\_str):**
   1. Проверяет, является ли строка time\_str валидным временем в формате "%H:%M".
   2. Если время валидно, возвращает True, иначе False.
3. **write\_transaction(file\_path, date, time, amount, operation\_type):**
   1. Записывает информацию о транзакции в файл по указанному пути file\_path.
   2. Преобразует операцию "+" в "income" и "-" в "expense".
   3. Добавляет строку с информацией о транзакции в файл.
4. **calculate\_balance(file\_path):**
   1. Считывает все транзакции из файла по указанному пути file\_path и вычисляет текущий баланс организации.
   2. Для каждой транзакции проверяет тип операции и соответственно увеличивает или уменьшает баланс.
5. **find\_most\_profitable\_transaction(file\_path):**
   1. Находит самую прибыльную транзакцию в файле по указанному пути file\_path.
   2. Проходит по всем транзакциям в файле, находит те, которые являются доходом и сравнивает их с текущей максимальной прибылью.
6. **find\_most\_unprofitable\_transaction(file\_path):**
   1. Находит самую убыточную транзакцию в файле по указанному пути file\_path.
   2. Проходит по всем транзакциям в файле, находит те, которые являются расходом и сравнивает их с текущей максимальной убыточностью.

**Пример работы программы**

В случае неверного ввода программа словит ошибку и уведомит пользователя о совершенном



**Структуры данных, способ хранения информации**

В данной работе были использованы следующие структуры данных:

1. **Строки (string):** Строки используются для хранения и обработки текстовых данных, таких как дата, время, сумма транзакции и тип операции. Встроенные методы строк позволяют производить различные операции с данными.
2. **Списки (list):** Списки используются для хранения данных, разделенных пробелами в каждой строке файла. Например, список line\_splitted содержит элементы строки файла, разделенные пробелами.
3. **Исключения (Exception):** Использование исключений позволяет обрабатывать ошибки, возникающие при работе с файлами или преобразовании данных. В коде используются блоки try-except для обработки исключений и вывода сообщений об ошибках.
4. **Встроенные типы данных Python:** Такие как int, float, bool используются для хранения числовых и логических данных в коде. Например, переменная max\_profit используется для хранения максимальной прибыли, а balance - для хранения текущего баланса организации.

Была использованна библиотека datetime которая обеспечивает удобные средства для работы с датами и временем, что делает ее полезной при разработке программ, требующих обработки временных данных.

Для хранения данных я использовал файл .txt что обеспечивает простой и быстрый доступ к информации, что также позволяет без особых трудностей обрабатывать данные этого файла в других программах на разных языках программирования, как вариант данный способ хранения данных дает возможность впоследствие без усложнения всей системы преобразовывать данные для хранения в других форматах без труда.

**Описание функционала**

Для записи данных файл производятся проверки введенных пользователем данных:

date = input("Enter transaction date (dd.mm.yyyy): ")

        if not is\_valid\_date(date):

          print("Not valid date")

          continue

        time = input("Enter transaction time (hh:mm): ")

        if not is\_valid\_time(time):

          print("Not valid time")

          continue

        amount = input("Enter transaction amount: ")

        operation\_type = input("Enter transaction type (income type +  expense type -): ")

        if not operation\_type in ["+","-"]:

          print("Not valid operation type")

          continue

        write\_transaction(file\_path, date, time, amount, operation\_type)

def is\_valid\_date(date\_str):

  try:

    date = datetime.datetime.strptime(date\_str, "%d.%m.%Y")

    if date >= datetime.datetime(2024,1,1) and date <= datetime.datetime.now():

      return True

  except:

    return False

def is\_valid\_time(time\_str):

  try:

    datetime.datetime.strptime(time\_str, "%H:%M")

    return True

  except:

    return False

данный код при помощи библиотеки datetime и метода strftime проверят корректность введенной даты и времени, а при помощи простого условия if проверяет на правильность веденного типа операции и если все данные верны будет произведена запись в файл

def write\_transaction(file\_path, date, time, amount, operation\_type):

  try:

    operation\_type = "income" if operation\_type == "+" else "expense" if operation\_type == "-" else None

    with open(file\_path, "a") as file:

      file.write("{} {} {} {}\n".format(date, time, amount, operation\_type))

  except:

    print('Error writing to file')

  return

перед записью в файл + и – преобразовываются в слова, а запись производится с использование with open для упрощения кода. Также присутствует проверка на ошибку записи.

Для подсчета баланса написан следующий код

def calculate\_balance(file\_path):

  balance = 0

  try:

    with open(file\_path, "r") as file:

      for line in file:

        line\_splitted = line.split()

        amount = float(line\_splitted[2])

        operation\_type = line\_splitted[3]

        if operation\_type == "income":

          balance += amount

        elif operation\_type == "expense":

          balance -= amount

  except:

    print("Error reading from file")

  return balance

данный код в зависимости от типа операции либо прибавляет к балансу сумму либо отнимает, также присутствует проверка на ошибку чтения файла.

Следующие две функции находят транзакции с самой большой прибылью или убытком

def find\_most\_profitable\_transaction(file\_path):

  max\_profit = float("-inf")

  res = "none"

  try:

    with open(file\_path, "r") as file:

      for line in file:

        line\_splitted = line.split()

        if line\_splitted[3] != "income":

          continue

        amount = float(line\_splitted[2])

        if amount > max\_profit:

          max\_profit = amount

          res = line

  except:

    print("Error reading from file")

  return res

def find\_most\_unprofitable\_transaction(file\_path):

  max\_expense = float("-inf")

  res = "none"

  try:

    with open(file\_path, "r") as file:

      for line in file:

        line\_splitted = line.split()

        if line\_splitted[3] != "expense":

          continue

        amount = float(line\_splitted[2])

        if amount > max\_expense:

          max\_expense = amount

          res = line

  except:

    print("Error reading from file")

  return res

данные две функции обладают идентичным функционалом за исключением цели поиска. Я не стал их соеденять в одну функцию с возможностью выбора параметра для определения цели поиска, так как логика использования данных функций разная и для удобства последующего использования функции логичнее обладать отдельной функцией для поиска наиприбыльной и наиубыточной транзакции.

**Вывод**

В ходе выполнения данной работы была разработана программа для учета финансовых транзакций. Программа позволяет пользователю записывать данные о транзакциях в файл, вычислять текущий баланс организации, а также находить самую прибыльную и убыточную транзакции. Для обеспечения корректной работы с датами и временем была использована библиотека datetime. Основной функционал программы реализован через несколько функций, каждая из которых отвечает за определенную операцию. В ходе работы были использованы структуры данных, такие как строки, списки и числа для обработки информации о транзакциях. В целом, выполнение работы позволило ознакомиться с принципами работы с файлами и обработки данных о финансовых операциях в Python.