

ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ МОЛДОВЫ  
ФАКУЛЬТЕТ МАТЕМАТИКИ И ИНФОРМАТИКИ  
ДЕПАРТАМЕНТ ИНФОРМАТИКИ

**Отчет**  
**по дисциплине „ПРОГРАММИРОВАНИЕ В PYTHON”**

Руководитель: Плешка Наталья, лектор

Выполнил: студент группы  
I2302 Богданов Юрий

Кишинев, 2024

## Вариант 1. «Учет финансовых транзакций организации «Х-фонды»»

### Краткая постановка задачи:

1. Запись данных о транзакциях в файл: Пользователь может ввести данные о транзакциях (дата, время, сумма и тип операции), которые проверяются на соответствие определенным требованиям (валидность даты и времени, правильность типа операции). Данные записываются, только если они полностью валидны.
2. Вывод текущего баланса фондов организации: Программа считает сумму всех транзакций, где доходы увеличивают баланс, а расходы уменьшают его.
3. Вывод данных о самой прибыльной транзакции: Отображение информации о транзакции с максимальной суммой дохода.
4. Вывод данных о самой убыточной транзакции: Отображение информации о транзакции с максимальной суммой расхода.
5. Выход из программы: Завершение работы программы.

Необходимо предусмотреть обработку ошибок, таких как некорректный ввод данных или чтение несуществующих файлов, и обеспечить корректное отображение информации после каждой операции. Также выбрать для сохранения данных тип файла, мотивируя свой выбор.

### Выбор типа файла для сохранения данных:

Прежде чем начать разбор проекта необходимо отметить, что для сохранения данных был выбран тип текстового формата файла **.txt** так как он имеет ряд преимуществ:

1. Простота и универсальность: Текстовые файлы могут быть открыты и прочитаны практически в любой операционной системе без необходимости использования специализированного программного обеспечения.
2. Гибкость формата: Позволяет записывать данные в любом предпочитаемом формате (например, разделенные пробелами, табуляциями или специальными символами), делая его удобным для различных видов обработки данных.
3. Отсутствие сложных зависимостей: Работа с текстовыми файлами не требует подключения библиотек для работы с форматами, как в

случае с CSV или Excel, что упрощает код и снижает потребность во внешних зависимостях.

## Логика реализованных алгоритмов:

Давайте рассмотрим логику каждого файла нашего проекта по отдельности. Первый файл `main.py` является самым главным и содержит меню для «Учета финансовых транзакций организации «Х-фонды»» из 5 опций выбора.

Давайте рассмотрим логику каждого пункта из нашего меню:

Первой опцией нашего меню является опция “Записать данные о транзакции”. При выборе данной опции пользователю предлагается ввести дату, время, сумму и тип транзакции (доход или расход) по соответствующему шаблону дд.мм.гггг, а дд из интервала 01-31 (можно и доп. проверки по месяцам), мм – из интервала 01-12, а год – 2024. Проверяется чтобы введенное время, чч:мм – были валидными – чч из интервала 00-23, мм – из интервала 00-59. Тип операции – одно из слов – «доход» или «расход».

Программа проверяет корректность введенных данных с помощью функций из модуля `validation.py`. Если данные неверны, пользователю выводится соответствующее сообщение об ошибке “Ошибка в данных транзакции.”.

Если все данные корректны, они записываются в файл. В случае успешной записи выводится сообщение о том, что транзакция успешно добавлена “Транзакция записана.”. В противном случае выводится сообщение о ошибке записи.

Вторая опция нашего меню “Вывести текущий баланс”

При выборе этой опции программа считывает все записи из файла и рассчитывает текущий баланс с помощью функции `calculateBalance` из модуля `transactions.py`, учитывая доходы и расходы.

Результат – текущий баланс организации – выводится пользователю. Если файл с данными не найден, выводится сообщение об ошибке.

Третья опция меню позволяет “Вывести самую прибыльную транзакцию”. Эта опция использует функцию `findExtremes`, которая анализирует все транзакции и определяет самую прибыльную из них.

Если в файле есть данные, выводится информация о транзакции с максимальной суммой дохода. Если прибыльные транзакции отсутствуют или файл пуст, пользователю сообщается, что прибыльные транзакции отсутствуют "Нет данных или нет прибыльных транзакций."

Четвертая опция меню позволяет "Вывести самую убыточную транзакцию". По аналогии с предыдущим пунктом, функция `findExtremes` также определяет самую убыточную транзакцию.

Информация о транзакции с наибольшим расходом выводится пользователю. Если убыточные транзакции отсутствуют или файл пуст, выводится соответствующее уведомление "Нет данных или нет убыточных транзакций".

Наконец пятая последняя опция нашего меню позволяет пользователю выйти из программы.

При выборе этой опции программа завершает свою работу, выводя сообщение о выходе "Выход из программы.". Выполняется команда `break`, которая прерывает цикл `while True` и завершает выполнение программы.

Второй файл нашего проекта модуль `validation.py`, который предназначен для обеспечения корректности данных, вводимых пользователем при добавлении новой транзакции в систему. Он помогает избежать ошибок и несоответствий в данных, что важно для точности финансовых отчетов и последующих аналитических операций.

Давайте подробно рассмотрим логику каждой функции:

Функция `validateDate(dateStr)`

Эта функция проверяет, соответствует ли введенная пользователем дата формату "дд.мм.гггг" и попадает ли она в рамки заданного временного периода (год должен быть 2024, и дата не должна быть позже текущего дня). Это предотвращает возможность ввода дат из прошлого, не соответствующих текущему финансовому году, или дат из будущего, которые могли бы вызвать ошибки в финансовом планировании. Если дата не проходит эти проверки, функция выводит соответствующее сообщение об ошибке и возвращает `False`

"Дата должна быть в 2024 году и не позже текущего дня."

"Неверный формат даты. Используйте формат дд.мм.гггг.". В случае успешной проверки возвращает `True`.

### Функция `validateTime(timeStr)`

Функция предназначена для проверки времени, указанного в формате "чч:мм". Она использует `datetime.strptime()` для конвертации строки в объект времени, что позволяет убедиться в том, что введенное время действительно существует и корректно отформатировано. Это важно для предотвращения ошибок при регистрации времени совершения транзакции, что может быть критично для точности финансовых записей. В случае неверного формата или некорректного времени функция сообщит об ошибке и вернет `False` "Неверный формат времени. Используйте формат чч:мм.". При успешной проверке возвращает `True`.

### Функция `validateTransactionType(typeStr)`

Эта функция проверяет, является ли тип транзакции одним из двух допустимых значений: "доход" или "расход". Проверка важна для корректной классификации каждой транзакции, что напрямую влияет на финансовый учет и анализ доходов и расходов организации. Если тип транзакции не соответствует одному из этих значений, функция выводит сообщение о необходимости ввода допустимого типа операции и возвращает `False` "Тип операции должен быть 'доход' или 'расход'.". При правильно указанном типе транзакции функция возвращает `True`.

Третий файл нашего проекта модуль `transactions.py`, в котором представлены функции (`writeTransaction`, `calculateBalance`, `findExtremes`), которые обеспечивают логику обработки финансовых транзакций для организации. Каждая функция выполняет определённые задачи в контексте управления транзакциями.

Давайте подробно рассмотрим логику каждой функции:

### Функция `writeTransaction`

Эта функция предназначена для записи финансовых транзакций в файл. Она принимает параметры: дату (*date*), время (*time*), сумму (*amount*), тип операции (*transType*), и путь к файлу (*filePath*). В начале функция проверяет валидность даты, времени и типа транзакции с помощью вызова функций валидации. Если все параметры валидны, функция пытается преобразовать строку суммы в число. В случае успеха записывает полную информацию о транзакции в файл. Если в процессе возникает ошибка (например, сумма не является числом), функция сообщает об этом и возвращает `False` "Сумма транзакции должна быть числом.". При успешной записи данных в файл функция возвращает `True`.

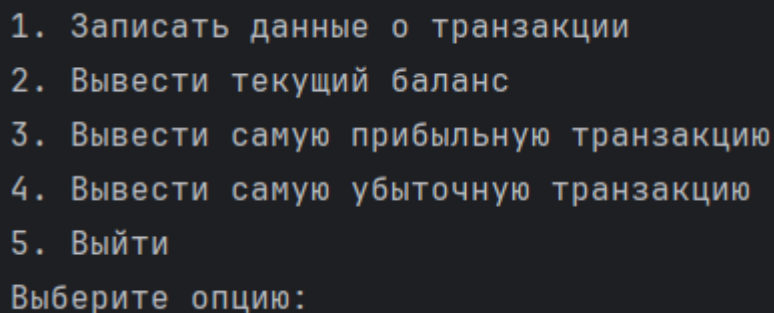
### Функция `calculateBalance`

Эта функция служит для расчета текущего баланса организации на основе сохраненных данных о транзакциях. Она читает файл, указанный в параметре `filePath`, и анализирует каждую строку, разделяя ее на составляющие. Для каждой транзакции сумма добавляется к общему балансу или вычитается из него в зависимости от ее типа (доход или расход). Если файл не найден, функция выводит соответствующее уведомление "Файл данных не найден." и возвращает текущий баланс.

### Функция `findExtremes`

Эта функция идентифицирует транзакции с максимальным доходом и максимальным расходом. Она проходит через каждую запись в файле, разбирает данные и сравнивает суммы транзакций. Для доходных транзакций функция ищет максимальную сумму, а для расходных — также максимальную, которая будет указывать на самую большую убыточную операцию. Функция возвращает данные о двух транзакциях: самой прибыльной и самой убыточной.

### Скрины интерфейсов для взаимодействия с пользователем и их назначение:



```
1. Записать данные о транзакции
2. Вывести текущий баланс
3. Вывести самую прибыльную транзакцию
4. Вывести самую убыточную транзакцию
5. Выйти
Выберите опцию:
```

На данном скриншоте показано наше меню, которое видит пользователь и с которым он может взаимодействовать.

Начнём:

```
Выберите опцию: 1
Введите дату (дд.мм.гггг): 25.04.2024
Введите время (чч:мм): 18:00
Введите сумму транзакции: 2455
Введите тип операции (доход/расход): доход
Транзакция записана.
```

На данном скриншоте показан выбор опции 1 для добавления новой транзакции дохода.

```
Выберите опцию: 1
Введите дату (дд.мм.гггг): 25.04.2024
Введите время (чч:мм): 18:01
Введите сумму транзакции: 133.45
Введите тип операции (доход/расход): расход
Транзакция записана.
```

На данном скриншоте показан выбор опции 1 для добавления новой транзакции расхода.

Далее:

```
Выберите опцию: 2
Текущий баланс: 33757.55
```

На данном скриншоте показан выбор опции 2 для вывода текущего баланса.

Далее:

```
Выберите опцию: 3
Самая прибыльная транзакция: 23.02.2024 12:33 23434.0 доход
```

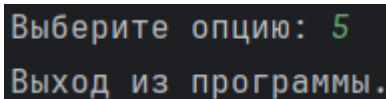
На данном скриншоте показан выбор опции 3 для вывода самой прибыльной транзакции.

Далее:

```
Выберите опцию: 4
Самая убыточная транзакция: 25.04.2024 11:11 2555.666 расход
```

На данном скриншоте показан выбор опции 4 для вывода самой убыточной транзакции.

Далее:



```
Выберите опцию: 5
Выход из программы.
```

На данном скриншоте показан выбор опции 5 для выхода из программы.

### **Структуры данных, с которыми работали в приложении (использовались для хранения данных):**

1. Списки: структура данных, используемая для хранения строковых представлений транзакций в файле. Каждая строка содержит информацию о дате, времени, сумме и типе транзакции.
2. Кортежи: используются для представления отдельных транзакций в виде четырех элементов: (дата, время, сумма, тип).
3. Словарь: используется внутри функции `findExtremes` для хранения информации о самых прибыльных и убыточных транзакциях. Каждая транзакция может быть представлена в виде пары ключ-значение, где ключом является тип транзакции ('доход' или 'расход'), а значением - кортеж с данными о самой транзакции (дата, время, сумма).
4. Файл: используется для хранения данных о транзакциях. Каждая транзакция записывается в файл в виде строки, содержащей дату, время, сумму и тип транзакции.

Эти структуры данных используются для хранения и обработки информации о финансовых транзакциях в нашем приложении.

### **Описание функционала мини-приложения, строками кода (кусками) и пояснениями:**

Для начала рассмотрим содержимое файла `main.py`.

```
from transactions import writeTransaction, calculateBalance, findExtremes
```

Эта строка импортирует три функции (`writeTransaction`, `calculateBalance`, `findExtremes`) из модуля `transactions.py`, которые будут использоваться в главной программе для работы с данными о транзакциях.



```
def mainMenu():  
    while True:  
        print("\n1. Записать данные о транзакции")  
        print("2. Вывести текущий баланс")  
        print("3. Вывести самую прибыльную транзакцию")  
        print("4. Вывести самую убыточную транзакцию")  
        print("5. Выйти")  
        choice = input("Выберите опцию: ")
```

Здесь определена функция `mainMenu()`, которая содержит бесконечный цикл. В каждой итерации цикла выводится меню опций для пользователя, и программа запрашивает выбор пользователя.

```
if choice == '1':  
    date = input("Введите дату (дд.мм.гггг): ")  
    time = input("Введите время (чч:мм): ")  
    amount = input("Введите сумму транзакции: ")  
    transType = input("Введите тип операции (доход/расход): ")  
    if writeTransaction(date, time, amount, transType, 'transactions.txt'):  
        print("Транзакция записана.")  
    else:  
        print("Ошибка в данных транзакции.")
```

Если пользователь выбирает опцию 1, программа запрашивает у пользователя данные о транзакции (дату, время, сумму и тип) и вызывает функцию `writeTransaction()` для записи этих данных в файл. Если транзакция успешно записана, выводится соответствующее сообщение, в противном случае выводится сообщение об ошибке.

```
elif choice == '2':  
    balance = calculateBalance('transactions.txt')  
    print(f"Текущий баланс: {balance}")
```

Когда пользователь выбирает опцию 2 из меню, программа вызывает функцию `calculateBalance('transactions.txt')`. Эта функция принимает путь к файлу с транзакциями и возвращает текущий баланс, основываясь на транзакциях в этом файле. Полученный баланс сохраняется в переменной `balance`, а затем выводится на экран с помощью функции `print()`.

```
elif choice == '3':
    transactions = findExtremes('transactions.txt')
    maxTransaction = transactions[0]
    if maxTransaction:
        print("Самая прибыльная транзакция:", " ".join(map(str,
maxTransaction)))
    else:
        print("Нет данных или нет прибыльных транзакций.")
```

Когда пользователь выбирает опцию 3, программа вызывает функцию `findExtremes('transactions.txt')`. Эта функция анализирует транзакции из файла и возвращает самую прибыльную транзакцию. Полученная транзакция сохраняется в переменной `maxTransaction`. Если такая транзакция найдена, она выводится на экран, иначе выводится сообщение о том, что либо данных нет, либо все транзакции убыточные.

```
elif choice == '4':
    transactions = findExtremes('transactions.txt')
    minTransaction = transactions[1]
    if minTransaction:
        print("Самая убыточная транзакция:", " ".join(map(str,
minTransaction)))
    else:
        print("Нет данных или нет убыточных транзакций.")
```

Когда пользователь выбирает опцию 4, программа также вызывает функцию `findExtremes('transactions.txt')`. Эта функция анализирует транзакции и возвращает самую убыточную транзакцию. Полученная транзакция сохраняется в переменной `minTransaction`. Если такая транзакция найдена, она выводится на экран, иначе выводится сообщение о том, что либо данных нет, либо все транзакции прибыльные.

```
elif choice == '5':  
    print("Выход из программы.")  
    break
```

Когда пользователь выбирает опцию 5, программа выводит сообщение о выходе из программы и завершает работу с помощью оператора `break`. Это позволяет выйти из бесконечного цикла `while True` и завершить выполнение программы.

```
else:  
    print("Неверная опция. Пожалуйста, выберите одну из  
предложенных опций.")
```

Если пользователь вводит значение, которое не соответствует ни одной из предложенных опций, программа выводит сообщение о неверной опции и возвращает его к выбору опций. Это обеспечивает корректное взаимодействие с пользователем, предотвращая неправильный ввод.

Мы рассмотрели содержимое файла `main.py` теперь перейдем к рассмотрению первого модуля `validation.py`.

```
import datetime
```

Эта строка добавляет в программу весь модуль **`datetime`**, что позволяет использовать его классы и функции.

```
try:  
    date = datetime.datetime.strptime(dateStr, '%d.%m.%Y')  
    if date.year == 2024 and date <= datetime.datetime.now():  
        return True  
    else:  
        print("Дата должна быть в 2024 году и не позже текущего дня.")  
        return False  
except ValueError:  
    print("Неверный формат даты. Используйте формат дд.мм.гггг.")  
    return False
```

Этот код в `validation.py` содержит функцию `validateDate(dateStr)`, которая проверяет правильность ввода даты. Она пытается разобрать строку

dateStr в объект даты с помощью метода `strptime()` из модуля `datetime`. Если это удастся, то проверяется, что дата находится в 2024 году и не позже текущего дня. Если дата не соответствует условиям, выводится соответствующее сообщение об ошибке и возвращается `False`. Если ввод некорректен, выводится сообщение о неверном формате даты и также возвращается `False`.

```
def validateTime(timeStr):
    try:
        time = datetime.datetime.strptime(timeStr, '%H:%M')
        return True
    except ValueError:
        print("Неверный формат времени. Используйте формат чч:мм.")
        return False
```

Эта функция `validateTime(timeStr)` в файле `validation.py` проверяет правильность ввода времени. Она пытается разобрать строку `timeStr` в объект времени с помощью метода `strptime()` из модуля `datetime`. Если это удастся, возвращается `True`. В случае ошибки выводится сообщение о неверном формате времени и возвращается `False`.

```
def validateTransactionType(typeStr):
    if typeStr.lower() in ['доход', 'расход']:
        return True
    else:
        print("Тип операции должен быть 'доход' или 'расход'.")
        return False
```

Эта функция `validateTransactionType(typeStr)` в файле `validation.py` проверяет правильность ввода типа операции. Если строка `typeStr` равна "доход" или "расход" (независимо от регистра), то возвращается `True`. В противном случае выводится сообщение о неправильном типе операции и возвращается `False`.

Мы рассмотрели все функции в файле `validation.py`, теперь перейдем к рассмотрению второго модуля `transactions.py`.

```
from validation import validateDate, validateTime, validateTransactionType
```

В этой строке происходит импорт функций `validateDate`, `validateTime`, `validateTransactionType` из модуля `validation.py`. Эти функции будут использоваться в файле `transactions.py` для проверки корректности введенных данных перед записью транзакции.

```
def writeTransaction(date, time, amount, transType, filePath):
```

```
    if validateDate(date) and validateTime(time) and  
    validateTransactionType(transType):
```

```
        try:
```

```
            amount = float(amount)
```

```
        except ValueError:
```

```
            print("Сумма транзакции должна быть числом.")
```

```
            return False
```

```
        with open(filePath, 'a') as file:
```

```
            file.write(f"{date} {time} {amount} {transType}\n")
```

```
        return True
```

```
    return False
```

Эта функция `writeTransaction()` записывает новую транзакцию в файл. Для начала она проверяет корректность даты, времени и типа транзакции с помощью вызова функций из модуля `validation`. Если все данные корректны, программа пытается преобразовать сумму транзакции в число. Если это удалось, транзакция записывается в файл в формате "дата время сумма тип", разделенные пробелами. Если во время выполнения возникает исключение или данные не проходят проверку, функция возвращает `False`.

```
def calculateBalance(filePath):  
  
    balance = 0  
  
    try:  
  
        with open(filePath, 'r') as file:  
  
            for line in file:  
  
                parts = line.strip().split()  
  
                amount = float(parts[2])  
  
                transType = parts[3]  
  
                if transType == 'доход':  
  
                    balance += amount  
  
                elif transType == 'расход':  
  
                    balance -= amount  
  
    except FileNotFoundError:  
  
        print("Файл данных не найден.")  
  
    return balance
```

Эта функция `calculateBalance()` вычисляет текущий баланс на основе транзакций из файла. Она открывает файл, читает каждую строку, разбирает ее на части и обновляет баланс в соответствии с суммой транзакции и ее типом (доход или расход). Если файл не найден, выводится сообщение об ошибке, и функция возвращает текущий баланс.

```

def findExtremes(filePath):

    maxTransaction = None

    minTransaction = None

    try:

        with open(filePath, 'r') as file:

            for line in file:

                parts = line.strip().split()

                date, time, amount, transType = parts[0], parts[1], float(parts[2]),
parts[3]

                if transType == 'доход' and (maxTransaction is None or amount >
maxTransaction[2]):

                    maxTransaction = (date, time, amount, transType)

                if transType == 'расход' and (minTransaction is None or amount >
minTransaction[2]):

                    minTransaction = (date, time, amount, transType)

    except FileNotFoundError:

        print("Файл данных не найден.")

    return maxTransaction, minTransaction

```

Эта функция `findExtremes()` находит самую прибыльную и самую убыточную транзакции в файле. Она открывает файл, читает каждую строку, разбирает ее на части и сравнивает сумму транзакции с текущими максимальными и минимальными значениями. Если файл не найден, выводится сообщение об ошибке, и функция возвращает `None` для обеих транзакций.

Таким образом, мы рассмотрели все функции из файла `transactions.py`.

## **Выводы:**

Использование Python для создания приложения учета финансовых транзакций оказалось эффективным и удобным. Python предлагает чистый и понятный синтаксис, что делает код более читаемым.

Модульность языка позволяет разделить функционал на отдельные модули, упрощая поддержку и сопровождение кода. Встроенные библиотеки для работы с датами и временем (например, datetime) облегчают обработку дат. Механизм обработки исключений помогает элегантно обрабатывать возможные ошибки. Легкость работы с файлами делает сохранение и загрузку данных удобной. В целом, Python оказался отличным выбором для данной задачи благодаря своей простоте, читаемости и богатой экосистеме библиотек.

## **Список используемой литературы:**

- <https://moodle.usm.md/course/view.php?id=6768>
- <https://pythonworld.ru/>
- <https://www.python.org/>