|  |  |  |
| --- | --- | --- |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное автономное  образовательное учреждение высшего образования  «Пермский государственный национальный  исследовательский университет» | | |
|  | Институт компьютерных наук и технологий | |
| **ОТЧЁТ**  по индивидуальной работе №1  по дисциплине «Языки Программирования»  Вариант 12 | | |
|  | | Работу выполнил  студент группы ПМИ-9,10-2023 1 курса  Черанёва Н.А.  «\_\_\_» \_\_\_\_\_\_\_\_2023 г. |
| Работу проверил  \_\_\_\_\_\_\_\_\_\_\_\_\_\_.  «\_\_\_» \_\_\_\_\_\_\_\_\_ 2023 г. |
| Пермь 2024 | | |

СОДЕРЖАНИЕ

[Постановка задачи 3](file:///C:\Users\chera\Downloads\Telegram%20Desktop\Шаблон_отчёт%20(2).docx#_Toc153130027)

[Алгоритм решения 3](file:///C:\Users\chera\Downloads\Telegram%20Desktop\Шаблон_отчёт%20(2).docx#_Toc153130028)

[Тестирование 5](file:///C:\Users\chera\Downloads\Telegram%20Desktop\Шаблон_отчёт%20(2).docx#_Toc153130029)

[Код программы 9](file:///C:\Users\chera\Downloads\Telegram%20Desktop\Шаблон_отчёт%20(2).docx#_Toc153130030)

# Постановка задачи

Создать файл записей, в котором хранится информация о личном бюджете: дата транзакции, время транзакции, направление (приход / расход), категория (питание, транспорт, развлечения, зарплата, аванс, подарок и т.д.), сумма, контрагент (источник поступления средств или их получатель в зависимости от направления). Разработать и реализовать программу "Персональный бюджет", которая считывает исходную информацию и позволяет на основе неё создавать следующие отчёты:

1.      Список всех поступлений, отсортированный по дате по убыванию, за прошедшие N дней (N вводится пользователем). Для поступлений, пришедших в один день, использовать второй ключ сортировки — сумма (по убыванию).

2.      Список всех затрат заданной категории, отсортированный по следующему ключу: дата (по убыванию), контрагент (по возрастанию), сумма (по убыванию).

3.      Список всех затрат, которые были понесены в определённом временном промежутке за все даты (например, с 18.00 до 21.00 - вводится с клавиатуры пользователем), отсортированный по ключу: сумма (по убыванию), контрагент (по возрастанию).

Создать базу транзакций, включающую не менее 25 записей и на основе неё сформировать все указанные списки (можно использовать вложенные списки и словари) и на основе неё сформировать все указанные списки). Взаимодействие с базой данных обеспечивается отдельными функциями – добавление/удаление/изменение записи, вывода на экран всей базы и отдельно сформированных списков. Интерфейс пользователя удобнее организовать в виде меню с выбором соответствующего действия. В программе обязательно наличие **дружественного интерфейса** и**защиты от некорректного пользовательского ввода.**  База должна содержать такие записи, чтобы во всех списках явно прослеживался заданный вид сортировки по всем ключам. Для сортировки записей использовать пирамидальную сортировку

# Алгоритм решения

1. Чтение файла функцией read\_transaction()

Создание нового списка, перебор каждой строчки файла и отслеживание текущей позиции в файле. Разделение строки на части по запятой и пробелу. Добавление частей, строк в словарь транзакций. Возвращение разделенного на части список.

1. Функция записи транзакций в файл write\_transactions(transactions)

Открытие файла и запись новой информации

1. Функция добавления транзакции add\_transaction()

Чтение файла и проверка на правильность вводимых данных от пользователя. Добавление новой информации в файл.

1. Функция удаления транзакции delete\_transaction()

Чтение транзакций с файла, ввод индекса для просмотра нужной строки. Удаление строки по вводимой пользователем информации.

1. Функция обновления транзакции update\_transaction()

Проверка на правильность вводимых данных от пользователя. Ввод индекса для просмотра нужной строки. Обновление и замена транзакции новой информации. Открытие, запись и вывод транзакций.

1. Функция вывода всех транзакций display\_all\_transactions()

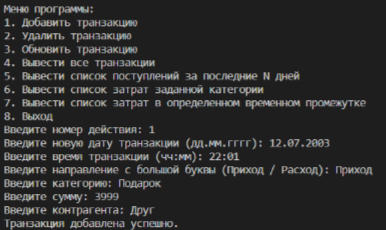
Проверка на пустоту базы транзакций, если она не пуста, то вывод.

1. Функция сортировки всех поступлений по дате (по убыванию), за прошедшие N дней. Для поступлений, пришедших в один день, второй ключ сортировки — сумма (по убыванию). sorted\_5
2. Функция сортировки всех затрат заданной пользователем категории по дате (по убыванию), по контрагенту (по возрастанию), по сумме (по убыванию). sorted\_6
3. Функция сортировки всех затрат, которые были понесены в определённом временном промежутке за все даты, вводимые пользователем, отсортированные по ключу: сумма (по убыванию), контрагент (по возрастанию). sorted\_7
4. Сортировки внутри функций производятся пирамидальным видом.

Пирамидальная сортировка – это алгоритм сортировки, который использует структуру данных под названием "куча". Куча - это полное бинарное дерево, которое удовлетворяет свойству кучи: значение каждого узла больше или равно значению его потомков.

# Тестирование

1 Тест. Добавление транзакции в список

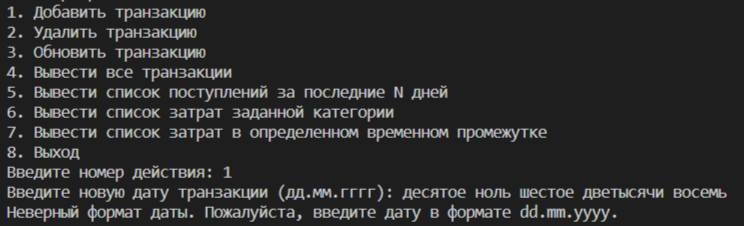


Результат:

В последнюю строчку в текстовом файле показывает добавленную транзакцию.



2 Тест. Защита от введения неверных данных.

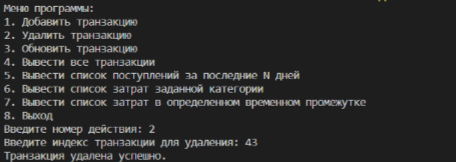


Результат:

Программа не дает вводить для даты буквы, ставить дату более 31 дня, месяц более 12 и год меньше 1000 или больше 2024. Для времени не дает ввод часов более 23 и более 59 минут. Для направления не дает вводить пустое значение. Для категории не дает выводить пустое значение. Для суммы не дает выводить буквы и пустое значение. Для контрагента не дает ввод пустого значения.

3 Тест. Удаление транзакции.

Удалим последнюю транзакцию под номером 43.

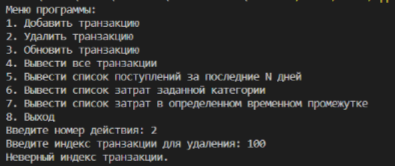


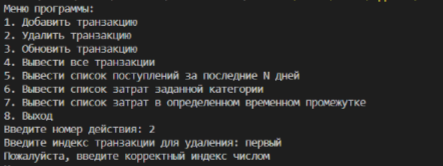
Результат:

Последняя транзакция успешно удалена



3 Тест. Защита от введения неверного индекса.

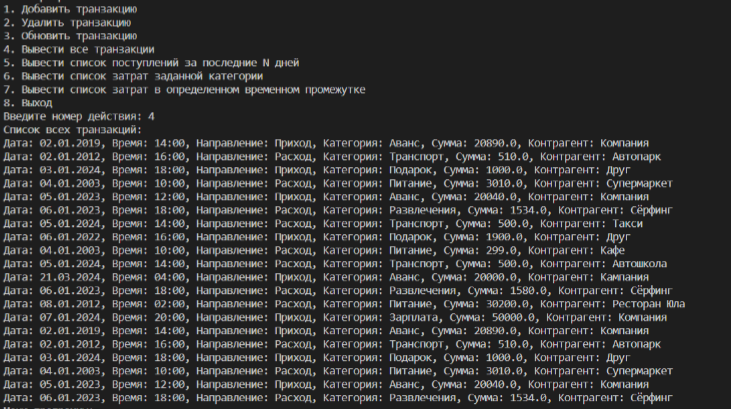




Результат:

Выведение ошибки, если такого номера транзакции нет. Если индекс введен буквами, просим пользователя ввести индекс числом.

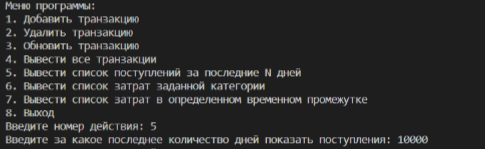
4 Тест. Вывод всех транзакций.



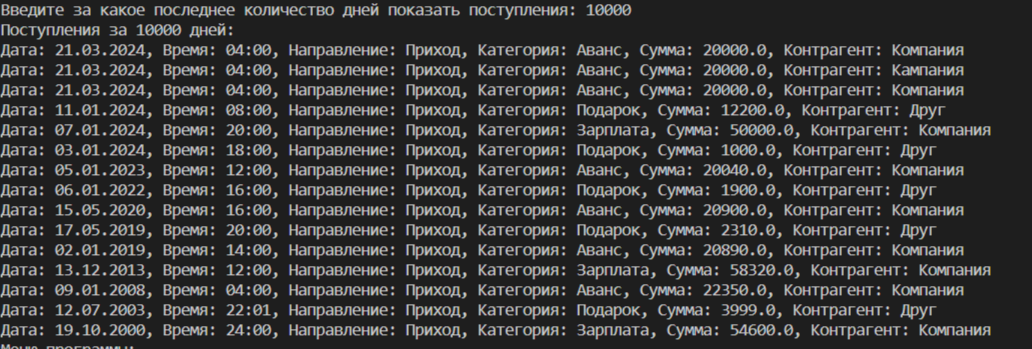
Результат:

Вывод всех транзакций, содержащихся в списке.

5 Тест. Вывод списка поступлений за последние N дней.

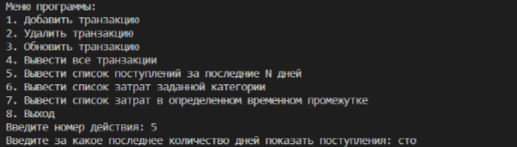


Результат:



Вывод сортировки всех поступлений по дате (по убыванию), за прошедшие N дней. Для поступлений, пришедших в один день, второй ключ сортировки — сумма (по убыванию).

6 Тест. Защита от неверно вводимых данных

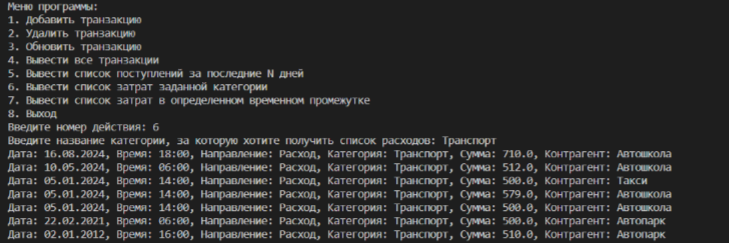


Результат:



Просим ввести корректное количество дней.

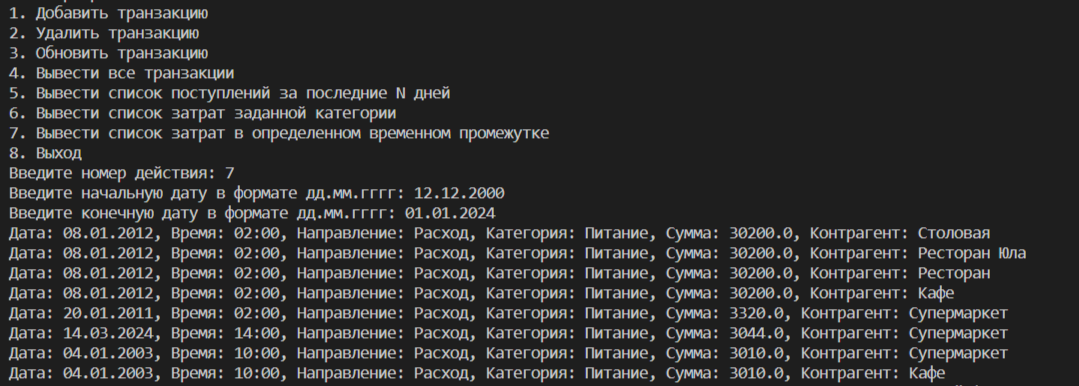
7 Тест. сортировка всех затрат заданной пользователем категории по дате (по убыванию), по контрагенту (по возрастанию), по сумме (по убыванию).



Результат:

Вывод отсортированных транзакций.

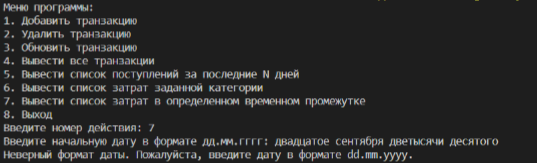
8 Тест. Сортировка всех затрат, которые были понесены в определённом временном промежутке за все даты, вводимые пользователем, отсортированные по ключу: сумма (по убыванию), контрагент (по возрастанию).



Результат:

Вывод отсортированных данных.

9 Тест. Защита от неверно вводимых данных.



Результат:

Просим пользователя ввести дату корректно.

# Код программы

from datetime import datetime

def read\_transactions(): #Функция чтения строк, разделения на части по необходимой информации

transactions = [] #Создание списка для хранения информации

with open('transactions.txt', 'r', encoding='utf-8') as file: #Открытие и интерпретация файла в кодировке UTF-8 для русского языка

lines = file.readlines() #Чтение каждой строчки файла

for i, line in enumerate(lines): # переменная i для отслеживания позиции текущей строки в списке lines.

#генерирация индексов и значений для каждой строки в lines списке

parts = line.strip().split(', ') # Разделение на части по запятой и пробелу

transaction = parts

transactions.append(transaction)#Добавление всех частей, строк в отдельный список

return transactions #Возвращает прочитанный, разделенный по частям список

def write\_transactions(transactions): #Функция чтения и записи транзакций в файл

with open('transactions.txt', 'w', encoding='utf-8') as file:

for transaction in transactions:

file.write(", ".join([transaction[0], transaction[1], transaction[2], transaction[3], str(transaction[4]), transaction[5]]) + '\n')

def add\_transaction():

transactions = read\_transactions()

while True:

date = input("Введите новую дату транзакции (дд.мм.гггг): ") #Проверка на дурака

if len(date) != 10 or date[2] != '.' or date[5] != '.' or not date.replace('.','').isdigit():

print("Неверный формат даты. Пожалуйста, введите дату в формате dd.mm.yyyy.")

break

day = int(date[:2]) #Срез и запоминание информации с помощью переменной

month = int(date[3:5])

year = int(date[6:])

if len(date) != 10 or date[2] != '.' or date[5] != '.': #Проверка на дурака

print("Неверный формат даты. Пожалуйста, введите дату в формате dd.mm.yyyy.")

break

elif day < 1 or day > 31 or month < 1 or month > 12 or (year <1000 or year>2024):

print("Неверный день, месяц или год. Пожалуйста, введите дату снова.")

break

time = input("Введите время транзакции (чч:мм): ")

if len(time) != 5 or time[2] != ':':

print("Неверный формат времени. Пожалуйста, введите время в формате hh:mm.")

break

else:

hour = int(time[:2])

minute = int(time[3:])

if hour > 23 or minute > 59:

print("Неверный час или минуты. Пожалуйста, введите время снова.")

break

direction = input("Введите направление с большой буквы (Приход / Расход): ")

if direction not in ['Приход', 'Расход']:

print("Неверное направление. Пожалуйста, введите Приход или Расход.")

break

category = input("Введите категорию: ")

if not category:

print("Категория не может быть пустой. Пожалуйста, введите категорию заново")

break

amount = input("Введите сумму: ")

if not amount.replace('.', '', 1).isdecimal():#Замена знаков чтоб осталась строка, которую мы могли бы проверить на то, что введены числа

print("Неверный формат суммы. Пожалуйста, введите сумму как число.")

break

counterparty = input("Введите контрагента: ")

if not counterparty:

print("Контрагент не может быть пустым. Пожалуйста, введите Контрагента заново")

break

transaction = [date, time, direction, category, float(amount), counterparty]

transactions.append(transaction)

write\_transactions(transactions)

print("Транзакция добавлена успешно.")

break

def delete\_transaction():

transactions = read\_transactions()

if not transactions:

print("База транзакций пуста.")

return

index = input("Введите индекс транзакции для удаления: ")

if not index.isdigit(): #Проверка, что строка содержит только число

print('Пожалуйста, введите корректный индекс числом')

return

index = int(index)

if index <= 0 or index >= len(transactions)+1: #Смотрим входит ли индекс в номер списка

print("Неверный индекс транзакции.")

return

else:

transactions.pop(index-1)#Удаление транзакции

print("Транзакция удалена успешно.")

write\_transactions(transactions)

def update\_transaction():

transactions = read\_transactions()

if not transactions:

print("База транзакций пуста.")

return

index = input("Введите индекс транзакции для обновления: ")

if not index.isdigit(): #Проверка на ввод чисел

print('Пожалуйста, введите корректный индекс')

return

index = int(index) #Преобразование введеннного пользователем значение в целое число и сохранение его в переменной index.

transaction = transactions[index-1]

while True:

date = input("Введите новую дату транзакции (дд.мм.гггг): ")#Проверка на дурака

if len(date) != 10 or date[2] != '.' or date[5] != '.' or not date.replace('.','').isdigit(): #Проверка на дурака

print("Неверный формат даты. Пожалуйста, введите дату в формате dd.mm.yyyy.")

break

day = int(date[:2]) #Срез и запоминание информации с помощью переменной

month = int(date[3:5])

year = int(date[6:])

if len(date) != 10 or date[2] != '.' or date[5] != '.': #Проверка на дурака

print("Неверный формат даты. Пожалуйста, введите дату в формате dd.mm.yyyy.")

break

elif day < 1 or day > 31 or month < 1 or month > 12 or (year <1000 or year>2024):

print("Неверный день, месяц или год. Пожалуйста, введите дату снова.")

break

time = input("Введите время транзакции (чч:мм): ")

if len(time) != 5 or time[2] != ':':

print("Неверный формат времени. Пожалуйста, введите время в формате hh:mm.")

break

else:

hour = int(time[:2])

minute = int(time[3:])

if hour > 23 or minute > 59:

print("Неверный час или минуты. Пожалуйста, введите время снова.")

break

direction = input("Введите направление с большой буквы (Приход / Расход): ")

if direction not in ['Приход', 'Расход']:

print("Неверное направление. Пожалуйста, введите Приход или Расход.")

break

category = input("Введите категорию: ")

if not category:

print("Категория не может быть пустой. Пожалуйста, введите категорию заново")

break

amount = input("Введите сумму: ")

if not amount.replace('.', '', 1).isdecimal(): #Проверка, что вводят числа

print("Неверный формат суммы. Пожалуйста, введите сумму как число.")

break

counterparty = input("Введите контрагента: ")

if not counterparty:

print("Контрагент не может быть пустым. Пожалуйста, введите Контрагента заново")

break

transaction[0] = date #Обновление и замена транкзации новой информацией

transaction[1] = time

transaction[2] = direction

transaction[3] = category

transaction[4] = amount

transaction[5] = counterparty

with open("transactions.txt", "w", encoding='utf-8') as file:

write\_transactions(transactions)

print("Транзакция обновлена успешно.")

break

def display\_all\_transactions():

transactions = read\_transactions()

if not transactions: #Проверка является ли список пустым

print("База транзакций пуста.")

return

print("Список всех транзакций:")

for transaction in transactions:

print(f"Дата: {transaction[0]}, Время: {transaction[1]}, Направление: {transaction[2]}, Категория: {transaction[3]}, Сумма: {transaction[4]}, Контрагент: {transaction[5]}")

def compare\_dates\_5(date1, date2, amount1, amount2):

date1\_obj = datetime.strptime(date1, "%d.%m.%Y") #Разделение даты на день, месяц и год

date2\_obj = datetime.strptime(date2, "%d.%m.%Y")

if date1\_obj > date2\_obj: #Сравнение дат для сортировки данных

return 1

elif date1\_obj < date2\_obj:

return -1

elif amount1 > amount2:

return 1

elif amount1 < amount2:

return -1

else:

return 0

def heapify\_5(arr, n, i):

largest = i

left = 2 \* i + 1

right = 2 \* i + 2

if left < n: #Проверка существования левого дочернего элемента

date\_comp = compare\_dates\_5(arr[i][0], arr[left][0], arr[i][4], arr[left][4])

if date\_comp == 1: #Если текущий узел меньше его левого потомка, обновляет largest индексом левого потомка.

largest = left

if right < n: #Проверка существования правого дочернего элемента

date\_comp = compare\_dates\_5(arr[largest][0], arr[right][0], arr[largest][4], arr[right][4])

if date\_comp == 1:#Если текущий узел меньше его правого потомка, обновляет largest индексом правого потомка.

largest = right

if largest != i:

arr[i], arr[largest] = arr[largest], arr[i]

heapify\_5(arr, n, largest)

def heap\_sort\_5(arr):

n = len(arr)

for i in range(n//2 - 1, -1, -1): #начинаем с элемента, который находится на позиции n//2 - 1 и заканчивая элементом на позиции 0

heapify\_5(arr, n, i)

for i in range(n-1, 0, -1): #Проходимся сортировкой в обратном порядке

arr[i], arr[0] = arr[0], arr[i]

heapify\_5(arr, i, 0)

def sorted\_5\_0():

transactions = read\_transactions()

heap\_sort\_5(transactions)

transaction\_0 = []

for transaction in transactions:

if transaction[2] == 'Приход':

transaction\_0.append(transaction)

return transaction\_0

def sorted\_5():

transactions\_sorted = sorted\_5\_0() #Сохранение отсортированной информации как переменной

#выводим список задач за последние n дней

n = input("Введите за какое последнее количество дней показать поступления: ")

if not n.isdigit():

print('Пожалуйста, введите корректное количество дней')

return

n = int(n)

print("Поступления за", n, "дней:")

current\_date = datetime.now().date() #использование datetime модуля для получения текущей даты и сохранения ее в переменной current\_date.

for transactions\_0 in transactions\_sorted:#запуск цикла, который повторяет каждый элемент в tasks\_sorted списке. Каждый элемент временно сохраняется в переменной transactions\_0.

transaction\_date = datetime.strptime(transactions\_0[0], '%d.%m.%Y') #преобразует часть даты транзакции (которая является строкой) в datetime.datetime объект и присваивает ее transaction\_date переменной.

current\_date = datetime.combine(current\_date, datetime.min.time()) #преобразование current\_date в datetime.datetime для последующего вычитания

#проверка, является ли разница в днях между текущей датой и датой транзакции меньше или равна n.

if (current\_date - transaction\_date).days <= n:

print(f'Дата: {transactions\_0[0]}, Время: {transactions\_0[1]}, Направление: {transactions\_0[2]}, Категория: {transactions\_0[3]}, Сумма: {transactions\_0[4]}, Контрагент: {transactions\_0[5]}')

def compare\_dates\_6(date1, date2, counterparty1, counterparty2, amount1, amount2):

date1\_obj = datetime.strptime(date1, "%d.%m.%Y")

date2\_obj = datetime.strptime(date2, "%d.%m.%Y")

if date1\_obj > date2\_obj:

return 1

elif date1\_obj < date2\_obj:

return -1

elif counterparty1 > counterparty2:

return 1

elif counterparty1 < counterparty2:

return -1

elif amount1 > amount2:

return 1

elif amount1 < amount2:

return -1

else:

return 0

def heapify\_6(arr, n, i):

largest = i

left = 2 \* i + 1

right = 2 \* i + 2

if left < n:

date\_comp = compare\_dates\_6(arr[i][0], arr[left][0], arr[i][5], arr[left][5], arr[i][4], arr[left][4])

if date\_comp == 1:#Если текущий узел меньше его левого потомка, обновляет largest индексом левого потомка.

largest = left

if right < n:

date\_comp = compare\_dates\_6(arr[largest][0], arr[right][0], arr[largest][5], arr[right][5], arr[largest][4], arr[right][4])

if date\_comp == 1:#Если текущий узел меньше его правого потомка, обновляет largest индексом правого потомка.

largest = right

if largest != i:

arr[i], arr[largest] = arr[largest], arr[i]

heapify\_6(arr, n, largest)

def heap\_sort\_6(arr):

n = len(arr)

for i in range(n//2 - 1, -1, -1):#начинаем с элемента, который находится на позиции n//2 - 1 и заканчивая элементом на позиции 0

heapify\_6(arr, n, i)

for i in range(n - 1, 0, -1):

arr[i], arr[0] = arr[0], arr[i]

heapify\_6(arr, i, 0)

def sorted\_6():

n = input('Введите название категории, за которую хотите получить список расходов: ').strip().lower()

transactions = read\_transactions()

heap\_sort\_6(transactions)

found = False

for transaction in transactions:

if transaction[2] == 'Расход' and n == transaction[3].strip().lower(): #удалить начальные и конечные пробелы и преобразовать все в нижний регистр, чтоб чтение проимходило с любым регистром

print (f'Дата: {transaction[0]}, Время: {transaction[1]}, Направление: {transaction[2]}, Категория: {transaction[3]}, Сумма: {transaction[4]}, Контрагент: {transaction[5]}')

found = True

if not found:

print('Транзация с направлением "Расход" или с выбранной вами категорией не существует. Пожалуйста, попробуйте ещё раз.')

def sorted\_7\_proverka():

transactions =read\_transactions()

while True:

#Преобразование строки начальной даты, введенную пользователем, в объект datetime с помощью функции

start\_time = input('Введите начальную дату в формате дд.мм.гггг: ')

if len(start\_time) != 10 or start\_time[2] != '.' or start\_time[5] != '.' or not start\_time.replace('.','').isdigit(): #Проверка на дурака

print("Неверный формат даты. Пожалуйста, введите дату в формате dd.mm.yyyy.")

return

day = int(start\_time[:2]) #Срез и запоминание информации с помощью переменной

month = int(start\_time[3:5])

year = int(start\_time[6:])

if day < 1 or day > 31 or month < 1 or month > 12 or (year <1000 or year>2024):

print("Неверный день, месяц или год. Пожалуйста, введите дату снова.")

break

end\_time = input('Введите конечную дату в формате дд.мм.гггг: ')

if len(end\_time) != 10 or end\_time[2] != '.' or end\_time[5] != '.' or not end\_time.replace('.','').isdigit(): #Проверка на дурака

print("Неверный формат даты. Пожалуйста, введите дату в формате dd.mm.yyyy.")

return

day = int(end\_time[:2]) #Срез и запоминание информации с помощью переменной

month = int(end\_time[3:5])

year = int(end\_time[6:])

if len(end\_time) != 10 or end\_time[2] != '.' or end\_time[5] != '.': #Проверка на дурака

print("Неверный формат даты. Пожалуйста, введите дату в формате дд.мм.гггг.")

break

elif day < 1 or day > 31 or month < 1 or month > 12 or (year < 1000 or year>2024):

print("Неверный день, месяц или год. Пожалуйста, введите дату снова.")

break

start\_time = datetime.strptime(start\_time, '%d.%m.%Y')

end\_time = datetime.strptime(end\_time, '%d.%m.%Y')

break

filtered\_transactions = []

for transaction in transactions:

if start\_time <= datetime.strptime(transaction[0], '%d.%m.%Y') <= end\_time and transaction[2] == 'Расход':

filtered\_transactions.append(transaction)#Если дата подходит, то записываем её

if not filtered\_transactions:#Если в списке нет ни одной транзакции - пишем не найдено

print("Транзакций с данным промежутком времени не найдено")

else:

sorted\_7()

def compare\_7(amount1, amount2, counterparty1, counterparty2):

amount1 = float(amount1) #Присваивание элементу тип данных с плавающей запятой

amount2 = float(amount2)

if amount1 > amount2:

return 1

elif amount1 < amount2:

return -1

elif counterparty1 > counterparty2:

return 1

elif counterparty1 < counterparty2:

return -1

else:

return 0

def heapify\_7(arr, n, i):

largest = i

left = 2 \* i + 1

right = 2 \* i + 2

if left < n:

date\_comp = compare\_7( arr[i][4], arr[left][4], arr[i][5], arr[left][5])

if date\_comp == 1:#Если текущий узел меньше его левого потомка, обновляет largest индексом левого потомка.

largest = left

if right < n:

date\_comp = compare\_7( arr[largest][4], arr[right][4], arr[largest][5], arr[right][5])

if date\_comp == 1:#Если текущий узел меньше его правого потомка, обновляет largest индексом правого потомка.

largest = right

if largest != i:

arr[i], arr[largest] = arr[largest], arr[i]

heapify\_7(arr, n, largest)

def heap\_sort\_7(arr):

n = len(arr)

for i in range(n//2 - 1, -1, -1): #начинаем с элемента, который находится на позиции n//2 - 1 и заканчивая элементом на позиции 0

heapify\_7(arr, n, i)

for i in range(n - 1, 0, -1):

arr[i], arr[0] = arr[0], arr[i]#Запуск цикла от последнего элемента массива до второго.

heapify\_7(arr, i, 0)

def sorted\_7():

transactions = read\_transactions()

transaction\_0 = []

heap\_sort\_7(transactions)

for transaction in transactions:

if transaction[2] == 'Расход':#Добавление транзакций направления "Расход"

transaction\_0.append(transaction)

for transaction in transaction\_0:

print (f'Дата: {transaction[0]}, Время: {transaction[1]}, Направление: {transaction[2]}, Категория: {transaction[3]}, Сумма: {transaction[4]}, Контрагент: {transaction[5]}')

def menu():

while True:

print("Меню программы:")

print("1. Добавить транзакцию")

print("2. Удалить транзакцию")

print("3. Обновить транзакцию")

print("4. Вывести все транзакции")

print("5. Вывести список поступлений за последние N дней")

print("6. Вывести список затрат заданной категории")

print("7. Вывести список затрат в определенном временном промежутке")

print("8. Выход")

choice = input("Введите номер действия: ")

if choice == "1":

add\_transaction() #После получения выбора запускаем функцию

elif choice == "2":

delete\_transaction()

elif choice == "3":

update\_transaction()

elif choice == "4":

display\_all\_transactions()

elif choice == "5":

sorted\_5()

elif choice == "6":

sorted\_6()

elif choice == "7":

sorted\_7\_proverka()

elif choice == "8":

break

else:

print("Неверный выбор. Попробуйте еще раз.")

menu()