ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ Р.МОЛДОВЫ

ФАКУЛЬТЕТ МАТЕМАТИКИ И ИНФОРМАТИКИ

ДЕПАРТАМЕНТ ИНФОРМАТИКИ

**Михайлов Пётр**

**Отчет**

по дисциплине „ПРОГРАММИРОВАНИЕ В PYTHON”

Вариант 3

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

(подпись)

Автор: \_\_\_\_\_Михайлов\_\_\_\_\_\_\_\_\_\_

(подпись)

Кишинев, 2024

Краткая постановка задачи:

Мне нужно разработать программу для учёта информации о сотрудниках компании «Х-фонды». Программа должна включать меню с несколькими опциями:

1. Запись данных о зарплатах сотрудников в файл. Я должен убедиться, что вводимые данные корректны: фамилии и имена должны состоять только из букв, даже если это сложные имена с тире. Название департамента должно состоять из букв; если оно из нескольких слов, между словами должен быть только один пробел. Зарплата должна быть вещественным числом в диапазоне от 1000.00 до 77000.00.

2. Вывод средней зарплаты сотрудников, где сумма всех зарплат делится на количество записей.

3. Вывод данных сотрудника с самой высокой зарплатой.

4. Вывод данных сотрудника с наименьшей зарплатой.

5. Вывод средней зарплаты по каждому департаменту, где сумма всех зарплат департамента делится на количество записей по этому департаменту.

6. Выход из программы.

Каждая запись в файле будет содержать фамилию, имя, департамент и зарплату сотрудника. Мне предстоит реализовать функции ввода и проверки данных, обработку и вывод информации согласно выбранному пункту меню.

Почему я выбрал формат файла txt?

Я выбрал формат txt для записи данных о сотрудниках, потому что он предоставляет простоту и универсальность. Текстовые файлы не требуют специальных программ для чтения или редактирования, что делает их доступными на любой операционной системе. Это особенно важно в контексте обработки и хранения данных, где необходима высокая совместимость и легкость взаимодействия с различными инструментами и платформами. Также, формат txt позволяет легко контролировать и отслеживать изменения в данных, поскольку я могу просто открыть файл в любом текстовом редакторе и быстро просмотреть или изменить информацию. Эта простота и открытость делают txt идеальным выбором для выполнения задач, связанных с записью и чтением данных.

Логика реализованных алгоритмов:

В рамках моего проекта по обработке данных сотрудников, я реализовал несколько ключевых алгоритмов, каждый из которых служит определенной цели. Во-первых, функция validate\_employee\_data используется для проверки корректности введенных данных. Это важно, чтобы убедиться, что все записи соответствуют установленному формату и допустимым диапазонам значений, что обеспечивает надежность данных. Я использую регулярные выражения для проверки формата фамилии, имени и департамента, а также диапазона зарплаты.

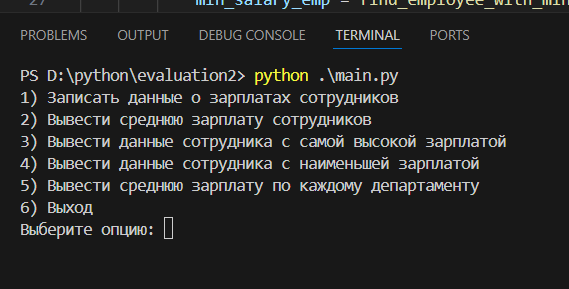
Функция write\_employee\_data предназначена для ввода данных пользователем и их записи в текстовый файл, если данные прошли валидацию. Это обеспечивает постоянство хранения информации и удобство последующего доступа к ней. Ошибка ввода или несоответствие формату данных приводит к предупреждению, что помогает избежать записи некорректной информации.

Следующая важная функция — read\_employee\_data, которая читает данные из файла и преобразует их в список сотрудников для дальнейшей обработки. Эта функция является основой для других алгоритмов анализа данных, таких как расчет средней зарплаты или поиск сотрудника с максимальной и минимальной зарплатой.

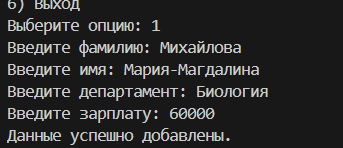
Функции calculate\_average\_salary, find\_employee\_with\_max\_salary, и find\_employee\_with\_min\_salary предоставляют статистический анализ данных. Они помогают определить средние зарплаты, а также идентифицировать сотрудников с наибольшими и наименьшими зарплатами в компании. Функции поиска максимальной и минимальной зарплаты могут возвращать не одного, а нескольких сотрудников в случае, если у них одинаковая зарплата.

Наконец, функция calculate\_average\_salary\_by\_department позволяет проводить анализ зарплат по департаментам, что может быть полезно для управленческих решений и планирования бюджета отделов. Этот алгоритм собирает данные по каждому департаменту и рассчитывает среднюю зарплату, что дает представление о структуре заработных плат в компании.

Скрины интерфейсов для взаимодействия с пользователем и их назначение:



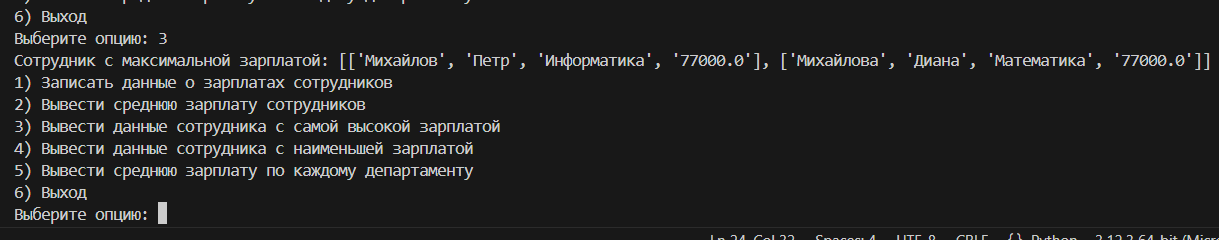
На данном скрине мы видим меню, с которым придется работать. В нем мы можешь записать данные о зарплатах сотрудников, вывести среднюю зарплату, вывести данные сотрудника с самой высокой и наименьшей зарплатой, вывести среднюю зарплату по каждому департаменту и сам выход из программы.



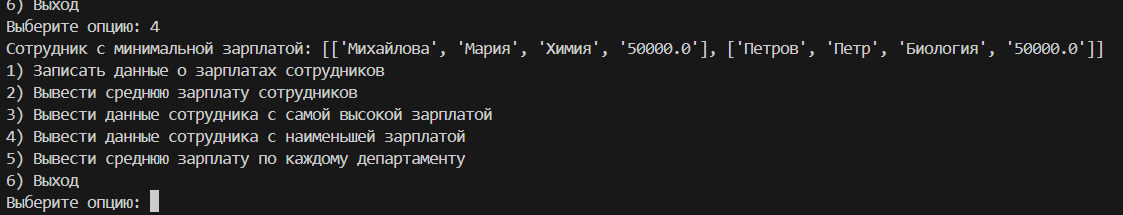
Запись данных выглядит так. Прошу заметить, имя двойное, было использовано тире и запись произошла успешно.



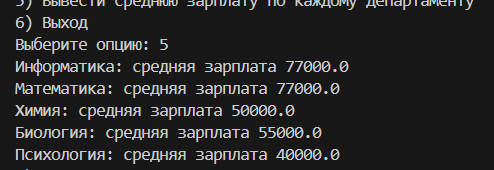
Вывод средней зарплаты выглядит так.



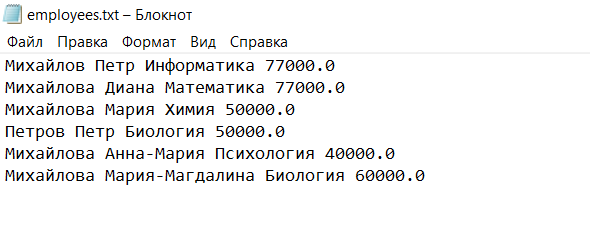
Вывод данных сотрудника с самой высокой зарплатой, здесь прошу заметить, что у двоих сотрудников одинаковая зарплата, поэтому и выводятся именно два сотрудника



То же самое происходит и с выводом самой наименьшей зарплатой, у двоих сотрудников наименьшая зарплата 50000, тем самым они выводятся оба.



Вывод средней зарплаты по каждому департаменту выглядит так, все департаменты разделены и была выведена правильная сумма средних зарплат.



Вот так выглядит запись в текстовый документ.

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

В моем приложении для управления данными сотрудников я использовал две основные структуры данных: списки и словари.

Списки используются для хранения данных о каждом сотруднике. Каждый сотрудник представлен как список, содержащий его фамилию, имя, департамент и зарплату. Эти списки собираются в один большой список, который содержит информацию о всех сотрудниках, позволяя легко перебирать и обрабатывать данные.

Словари применяются для агрегации и анализа данных, например, при расчете средней зарплаты по департаментам. Словарь организует зарплаты по ключам департаментов, где каждый ключ ссылается на список зарплат сотрудников данного департамента. Это облегчает вычисление статистических показателей.

Эти структуры данных позволяют мне эффективно управлять, анализировать и обновлять информацию в приложении.

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

Моё мини-приложение по управлению данными сотрудников включает в себя несколько основных функций, каждая из которых отвечает за определённые операции с данными. Давайте подробно рассмотрим каждую функцию с примерами кода и пояснениями.

1.Валидация данных сотрудника (validate\_employee\_data):

def validate\_employee\_data(last\_name, first\_name, department, salary):

    if not re.match(r'^[a-zA-Zа-яА-Я-]+$', last\_name):

        return False

    if not re.match(r'^[a-zA-Zа-яА-Я-]+$', first\_name):

        return False

    if not re.match(r'^[a-zA-Zа-яА-Я]+(?: [a-zA-Zа-яА-Я]+)?$', department):

        return False

    if not (1000.00 <= salary <= 77000.00):

        return False

    return True

Эта функция проверяет корректность данных сотрудника, таких как фамилия, имя, департамент и зарплата. Она использует регулярные выражения для проверки формата строк и проверяет, что зарплата находится в допустимом диапазоне.

2. Запись данных сотрудника в файл (write\_employee\_data):

def write\_employee\_data(filename):

    last\_name = input("Введите фамилию: ")

    first\_name = input("Введите имя: ")

    department = input("Введите департамент: ")

    salary\_input = input("Введите зарплату: ")

    try:

        salary = float(salary\_input)

    except ValueError:

        print("Некорректный ввод зарплаты. Зарплата должна быть числом.")

        return

    if validate\_employee\_data(last\_name, first\_name, department, salary):

        with open(filename, 'a') as file:

            file.write(f"{last\_name} {first\_name} {department} {salary}\n")

        print("Данные успешно добавлены.")

    else:

        print("Некорректные данные.")

Функция запрашивает у пользователя данные сотрудника, проверяет их с помощью validate\_employee\_data и записывает в файл, если они верны. На случай если что-то не введено правильно, выведется ошибка.

3.Чтение данных сотрудников из файла (read\_employee\_data):

def read\_employee\_data(filename):

    employees = []

    try:

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

            for line in file:

                parts = line.strip().split()

                if len(parts) == 4:

                    employees.append(parts)

    except FileNotFoundError:

        print("Файл не найден.")

    return employees

Читает данные из файла и возвращает список сотрудников. Каждый сотрудник представлен списком атрибутов. На случай если файл не найден, то будет выведена ошибка. Эта функция прочитывает файл построчно, разделяет строки на части и формирует список сотрудников.

4.Расчет средней зарплаты (calculate\_average\_salary):

def calculate\_average\_salary(employees):

    if not employees:

        return 0

    total\_salary = sum(float(emp[3]) for emp in employees)

    return total\_salary / len(employees)

Вычисляет среднюю зарплату для списка сотрудников. Эта функция берёт список сотрудников, извлекает зарплаты и рассчитывает их среднее значение.

5.Поиск сотрудника с максимальной и минимальной зарплатой (find\_employee\_with\_max\_salary, find\_employee\_with\_min\_salary):

def find\_employee\_with\_max\_salary(employees):

    if not employees:

        return []

    max\_salary = max(float(emp[3]) for emp in employees)

    return [emp for emp in employees if float(emp[3]) == max\_salary]

def find\_employee\_with\_min\_salary(employees):

    if not employees:

        return []

    min\_salary = min(float(emp[3]) for emp in employees)

    return [emp for emp in employees if float(emp[3]) == min\_salary]

Функции find\_employee\_with\_max\_salary и find\_employee\_with\_min\_salary идентифицируют сотрудников с максимальной и минимальной зарплатой соответственно. Эти функции определяют максимальную и минимальную зарплату в списке, а затем возвращают всех сотрудников с этими зарплатами.

6.Расчет средней зарплаты по департаментам (calculate\_average\_salary\_by\_department):

def calculate\_average\_salary\_by\_department(employees):

    departments = {}

    for last\_name, first\_name, department, salary in employees:

        if department not in departments:

            departments[department] = []

        departments[department].append(float(salary))

    averages = {dep: sum(sals) / len(sals) for dep, sals in departments.items()}

    return averages

Вычисляет среднюю зарплату для каждого департамента. Эта функция группирует зарплаты по департаментам и рассчитывает средние значения для каждой группы.

Каждая из этих функций взаимодействует с пользователем или данными для выполнения своих задач, обеспечивая удобный инструментарий для управления информацией о сотрудниках.

from functions import write\_employee\_data, read\_employee\_data, calculate\_average\_salary, \

    find\_employee\_with\_max\_salary, find\_employee\_with\_min\_salary, calculate\_average\_salary\_by\_department

Эта строка подключает необходимые функции из модуля functions. Эти функции отвечают за запись данных в файл, чтение данных, расчет средней зарплаты, поиск сотрудников с максимальной и минимальной зарплатой, а также расчет средней зарплаты по департаментам.

def main():

    filename = "employees.txt"

Здесь объявляется основная функция main, которая управляет всем приложением. Переменная filename указывает имя файла, в котором хранятся данные о сотрудниках.

while True:

        print("1) Записать данные о зарплатах сотрудников")

        print("2) Вывести среднюю зарплату сотрудников")

        print("3) Вывести данные сотрудника с самой высокой зарплатой")

        print("4) Вывести данные сотрудника с наименьшей зарплатой")

        print("5) Вывести среднюю зарплату по каждому департаменту")

        print("6) Выход")

        choice = input("Выберите опцию: ")

Этот цикл предоставляет пользователю меню с опциями. Пользователь вводит свой выбор, и в зависимости от этого выбора выполняется соответствующая операция.

  if choice == '1':

            write\_employee\_data(filename)

        elif choice == '2':

            employees = read\_employee\_data(filename)

            average = calculate\_average\_salary(employees)

            print("Средняя зарплата:", average)

        elif choice == '3':

            employees = read\_employee\_data(filename)

            max\_salary\_emp = find\_employee\_with\_max\_salary(employees)

            if max\_salary\_emp:

                print("Сотрудник с максимальной зарплатой:", max\_salary\_emp)

        elif choice == '4':

            employees = read\_employee\_data(filename)

            min\_salary\_emp = find\_employee\_with\_min\_salary(employees)

            if min\_salary\_emp:

                print("Сотрудник с минимальной зарплатой:", min\_salary\_emp)

        elif choice == '5':

            employees = read\_employee\_data(filename)

            averages = calculate\_average\_salary\_by\_department(employees)

            for department, average in averages.items():

                print(department + ":", "средняя зарплата", average)

        elif choice == '6':

            break

        else:

            print("Некорректный выбор.")

Каждый блок if и elif обрабатывает выбор пользователя, выполняя соответствующие функции для чтения данных, их анализа или вывода на экран. Выбрав 6, происходит выход программы. Также если не будет введена правильная цифра выйдет ошибка.

Вывод:

В этом проекте Python используется как основной инструмент для создания мини-приложения по управлению данными сотрудников, что позволяет мне полностью контролировать процесс с помощью простого и мощного синтаксиса языка. С его помощью я реализовал функции для чтения и записи файлов, валидации данных, а также для расчёта статистических данных, таких как средняя зарплата и определение сотрудников с максимальной и минимальной зарплатой. Python оказался идеальным выбором для этих задач благодаря своей читаемости и обширной поддержке библиотек, которые облегчают работу с файлами и обработку данных. Использование этого языка в проекте значительно упростило разработку и поддержку программы, позволяя мне сосредоточиться на логике обработки данных, а не на сложностях программирования.

Список используемой литературы:

1. Презентации лекций на Moodle.
2. YouTube
3. Official Python Documentation.