ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«УФИМСКИЙ УНИВЕРСИТЕТ НАУКИ И ТЕХНОЛОГИЙ»

Институт информатики, математики и робототехники
Кафедра математического и компьютерного моделирования

Лабораторная работа №11: TOML-INI-CSV

ОБУЧАЮЩЕГОСЯ
4 курса группы ПИ-4ИВТ221Б

Санникова Михаила Александровича

Уровень высшего образования:	высшее образование – бакалавриат
Направление подготовки (специальность)	09.03.03 “Прикладная информатика”
Направленность (профиль) программы	Информационные и вычислительные технологии
Дата выполнения	25.11.2025

УФА – 2025

Постановка задачи:

Необходимо разработать приложение, работающее с документами TOML. Приложение должно уметь сохранять и загружать данные из форматов TOML, INI и CSV.

Этап 1: Создайте TOML файл, содержащий информацию об авиарейсах
Этап 2: Чтение и базовый анализ данных
Напишите программу на Python, которая:

- Читает TOML файл и извлекает данные из вложенных структур
- Рассчитывает средние показатели
- Выявляет рейс с наилучшими и наихудшими показателями
- Формирует сводный отчет по эффективности работы авиарейсов

Этап 3: Расширенная фильтрация и группировка
Этап 4: Генерация отчетных конфигураций
Реализуйте систему управления конфигурациями, которая:

- Формирует отдельные TOML файлы
- Генерирует конфигурации
- Создает отчеты
- Сохраняет историю изменений ключевых параметров

Этап 5: Обработка ошибок и валидация данных
Реализуйте систему обработки ошибок, которая:

- Проверяет целостность вложенных структур данных
- Валидирует форматы дат и числовых значений
- Обрабатывает отсутствующие или некорректные данные в массивах таблиц
- Предоставляет детализированные сообщения об ошибках для различных сценариев

Практическая часть:

Листинг кода с комментариями

```
import toml
import statistics
from datetime import datetime
import configparser
import csv
import os



# ============================================================================
# КЛАСС ДЛЯ ЧТЕНИЯ И БАЗОВОГО АНАЛИЗА ДАННЫХ (ЭТАП 2)
# ============================================================================

class FlightAnalyzer:
    def __init__(self, toml_file):
        self.data = self.load_toml_data(toml_file)
        self.flights = self.data['flights']

    def load_toml_data(self, file_path):
        """Загрузка данных из TOML файла"""
        try:
            with open(file_path, 'r', encoding='utf-8') as file:
                return toml.load(file)
        except Exception as e:
            print(f"Ошибка загрузки файла: {e}")
            return {}

    def calculate_average_metrics(self):
        """Расчет средних показателей"""
        load_factors = [flight['daily_metrics']['load_factor'] for flight in self.flights]
        fuel_consumptions = [flight['daily_metrics']['fuel_consumption'] for flight in self.flights]
        delays = [flight['daily_metrics']['delay_minutes'] for flight in self.flights]

        return {
            'avg_load_factor': statistics.mean(load_factors),
            'avg_fuel_consumption': statistics.mean(fuel_consumptions),
            'avg_delay': statistics.mean(delays),
            'total_flights': len(self.flights)
        }

    def find_best_worst_performers(self):
        """Поиск рейсов с лучшими и худшими показателями"""
        flights_with_metrics = []

        for flight in self.flights:
            score = (
                    flight['daily_metrics']['load_factor'] * 100 -
                    flight['daily_metrics']['delay_minutes'] * 0.5 -
                    flight['daily_metrics']['fuel_consumption'] * 0.001
            )
            flights_with_metrics.append({
                'flight': flight['number'],
                'route': flight['route'],
                'score': score,
                'load_factor': flight['daily_metrics']['load_factor'],
                'delay': flight['daily_metrics']['delay_minutes']
```

```
        })

    # Сортировка по эффективности
    sorted_flights = sorted(flights_with_metrics, key=lambda x: x['score'], reverse=True)

    return {
        'best': sorted_flights[0],
        'worst': sorted_flights[-1],
        'ranking': sorted_flights
    }

def generate_summary_report(self):
    """Формирование сводного отчета"""
    avg_metrics = self.calculate_average_metrics()
    performers = self.find_best_worst_performers()

    print("=== СВОДНЫЙ ОТЧЕТ ПО АВИАРЕЙСАМ ===")
    print(f"Всего рейсов: {avg_metrics['total_flights']}")
    print(f"Средняя загрузка: {avg_metrics['avg_load_factor']:.1%}")
    print(f"Средняя задержка: {avg_metrics['avg_delay']} минут")
    print(f"Средний расход топлива: {avg_metrics['avg_fuel_consumption']:.0f} кг")

    print(f"\n🚀 Лучший рейс: {performers['best']['flight']}")
    print(f"   Маршрут: {performers['best']['route']}")
    print(f"   Загрузка: {performers['best']['load_factor']:.1%}")

    print(f"\n⚠️  Худший рейс: {performers['worst']['flight']}")
    print(f"   Маршрут: {performers['worst']['route']}")
    print(f"   Загрузка: {performers['worst']['load_factor']:.1%}")
    print(f"   Задержка: {performers['worst']['delay']} минут")

    return {
        'average_metrics': avg_metrics,
        'performance': performers
    }


# ================================================================================
# КЛАСС ДЛЯ РАСШИРЕННОЙ ФИЛЬТРАЦИИ И ГРУППИРОВКИ (ЭТАП 3)
# ================================================================================

class FlightFilter:
    def __init__(self, flights_data):
        self.flights = flights_data

    def filter_by_status(self, status):
        """Фильтрация по статусу рейса"""
        return [flight for flight in self.flights if flight['status'] == status]

    def filter_by_complex_criteria(self, min_load=0.7, max_delay=30, max_fuel=5000):
        """Фильтрация по комплексным критериям"""
        filtered = []
        for flight in self.flights:
            metrics = flight['daily_metrics']
            if (metrics['load_factor'] >= min_load and
                metrics['delay_minutes'] <= max_delay and
                metrics['fuel_consumption'] <= max_fuel):
```

```
            filtered.append(flight)
        return filtered

    def group_by_aircraft_type(self):
        """Группировка по типу самолета"""
        groups = {}
        for flight in self.flights:
            aircraft_type = flight['aircraft_type']
            if aircraft_type not in groups:
                groups[aircraft_type] = []
            groups[aircraft_type].append(flight)
        return groups

    def detect_anomalies(self):
        """Выявление аномалий в данных"""
        anomalies = []
        if len(self.flights) < 2:
            return anomalies

        load_factors = [f['daily_metrics']['load_factor'] for f in self.flights]
        avg_load = statistics.mean(load_factors)
        std_load = statistics.stdev(load_factors) if len(load_factors) > 1 else 0

        for flight in self.flights:
            load = flight['daily_metrics']['load_factor']
            delay = flight['daily_metrics']['delay_minutes']

            # Аномалия загрузки
            if std_load > 0 and abs(load - avg_load) > 2 * std_load:
                anomalies.append({
                    'flight': flight['number'],
                    'type': 'аномальная загрузка',
                    'value': f"{load:.1%}",
                    'expected': f"{avg_load:.1%}"
                })

            # Аномалия задержки
            if delay > 60:  # Задержка более 1 часа
                anomalies.append({
                    'flight': flight['number'],
                    'type': 'значительная задержка',
                    'value': f"{delay} минут"
                })

        return anomalies

    def print_filter_results(self):
        """Вывод результатов фильтрации"""
        print("\n=== РЕЗУЛЬТАТЫ ФИЛЬТРАЦИИ ===")
        print("Рейсы по расписанию:", [f['number'] for f in self.filter_by_status('scheduled')])
        print("Задержанные рейсы:", [f['number'] for f in self.filter_by_status('delayed')])

        optimal_flights = self.filter_by_complex_criteria(min_load=0.8, max_delay=15)
        print("Оптимальные рейсы (загрузка >80%, задержка <15 мин):",
              [f['number'] for f in optimal_flights])

        anomalies = self.detect_anomalies()
```

```
    if anomalies:
        print("\n⚠ Обнаружены аномалии:")
        for anomaly in anomalies:
            print(f"  {anomaly['flight']}: {anomaly['type']} - {anomaly['value']}")
    else:
        print("\n✅ Аномалий не обнаружено")


# ============================================================================
# КЛАСС ДЛЯ ГЕНЕРАЦИИ ОТЧЕТНЫХ КОНФИГУРАЦИЙ (ЭТАП 4)
# ============================================================================

class ReportGenerator:
    def __init__(self, flights_data):
        self.flights = flights_data

    def generate_aircraft_type_reports(self):
        """Генерация отдельных отчетов по типам самолетов"""
        groups = FlightFilter(self.flights).group_by_aircraft_type()

        # Создание папки для отчетов
        os.makedirs('reports', exist_ok=True)

        for aircraft_type, flights in groups.items():
            report_data = {
                'aircraft_type': aircraft_type,
                'flights_count': len(flights),
                'flights': flights,
                'summary': self._calculate_type_summary(flights)
            }

            # Сохранение в отдельный TOML файл
            filename = f"reports/{aircraft_type.replace(' ', '_').lower()}_report.toml"
            with open(filename, 'w', encoding='utf-8') as f:
                toml.dump(report_data, f)
            print(f"Создан отчет: {filename}")

    def generate_maintenance_config(self):
        """Генерация конфигураций для техобслуживания"""
        maintenance_config = {
            'maintenance_schedule': {
                'next_check': datetime.now().strftime('%Y-%m-%d'),
                'flights_due': []
            }
        }

        for flight in self.flights:
            if flight['daily_metrics']['fuel_consumption'] > 4000:
                maintenance_config['maintenance_schedule']['flights_due'].append({
                    'flight_number': flight['number'],
                    'aircraft_type': flight['aircraft_type'],
                    'reason': 'высокий расход топлива',
                    'priority': 'high' if flight['daily_metrics']['fuel_consumption'] > 5000 else 'medium'
                })

        with open('reports/maintenance_schedule.toml', 'w', encoding='utf-8') as f:
            toml.dump(maintenance_config, f)
```

```
        print("Создан график техобслуживания: reports/maintenance_schedule.toml")

    def _calculate_type_summary(self, flights):
        """Расчет сводки по типу самолета"""
        return {
            'total_flights': len(flights),
            'avg_load_factor': statistics.mean(f['daily_metrics']['load_factor'] for f in flights),
            'avg_delay': statistics.mean(f['daily_metrics']['delay_minutes'] for f in flights)
        }

    def generate_all_reports(self):
        """Генерация всех отчетов"""
        print("\n=== ГЕНЕРАЦИЯ ОТЧЕТОВ ===")
        self.generate_aircraft_type_reports()
        self.generate_maintenance_config()


# ============================================================================
# КЛАСС ДЛЯ ОБРАБОТКИ ОШИБОК И ВАЛИДАЦИИ ДАННЫХ (ЭТАП 5)
# ============================================================================

class DataValidator:
    def __init__(self, flights_data):
        self.flights = flights_data
        self.errors = []

    def validate_data_integrity(self):
        """Проверка целостности данных"""
        for i, flight in enumerate(self.flights):
            self._validate_flight_structure(flight, i)
            self._validate_numeric_values(flight, i)
            self._validate_dates(flight, i)

        return self.errors

    def _validate_flight_structure(self, flight, index):
        """Проверка структуры данных рейса"""
        required_sections = ['schedule', 'airport_info', 'status_info', 'passenger_stats', 'daily_metrics']
        for section in required_sections:
            if section not in flight:
                self.errors.append(f"Рейс {index}: Отсутствует секция {section}")

    def _validate_numeric_values(self, flight, index):
        """Валидация числовых значений"""
        try:
            # Проверка загрузки
            load_factor = flight['daily_metrics']['load_factor']
            if not 0 <= load_factor <= 1:
                self.errors.append(f"Рейс {flight['number']}: Некорректная загрузка {load_factor}")

            # Проверка расхода топлива
            fuel = flight['daily_metrics']['fuel_consumption']
            if fuel <= 0:
                self.errors.append(f"Рейс {flight['number']}: Некорректный расход топлива {fuel}")

        except KeyError as e:
            self.errors.append(f"Рейс {flight['number']}: Отсутствует поле {e}")
```

```python
def _validate_dates(self, flight, index):
    """Валидация дат и времени"""
    try:
        departure = flight['schedule']['departure']
        arrival = flight['schedule']['arrival']

        # Проверка, что это объекты datetime
        if not isinstance(departure, datetime) or not isinstance(arrival, datetime):
            self.errors.append(f"Рейс {flight['number']}: Неверный тип данных для дат")

        # Проверка, что время прибытия позже вылета
        if arrival <= departure:
            self.errors.append(f"Рейс {flight['number']}: Время прибытия раньше вылета")

    except KeyError as e:
        self.errors.append(f"Рейс {flight['number']}: Отсутствует поле {e}")
    except Exception as e:
        self.errors.append(f"Рейс {flight['number']}: Ошибка при проверке дат - {e}")

def handle_missing_data(self):
    """Обработка отсутствующих данных"""
    for flight in self.flights:
        # Заполнение отсутствующих значений по умолчанию
        flight['daily_metrics'].setdefault('delay_minutes', 0)
        flight['passenger_stats'].setdefault('boarded', 0)

def get_validation_report(self):
    """Получение отчета о валидации"""
    errors = self.validate_data_integrity()
    self.handle_missing_data()

    report = {
        'validation_date': datetime.now().isoformat(),
        'total_flights_checked': len(self.flights),
        'errors_found': len(errors),
        'errors_details': errors,
        'status': 'PASS' if not errors else 'FAIL'
    }

    return report

def print_validation_report(self):
    """Вывод отчета о валидации"""
    validation_report = self.get_validation_report()

    print("\n=== ОТЧЕТ О ВАЛИДАЦИИ ДАННЫХ ===")
    print(f"Статус: {validation_report['status']}")
    print(f"Найдено ошибок: {validation_report['errors_found']}")
    if validation_report['errors_details']:
        print("Детали ошибок:")
        for error in validation_report['errors_details']:
            print(f"  - {error}")
    else:
        print("✅ Все данные прошли валидацию успешно!")
```

```python
# ============================================================================
# УТИЛИТЫ ДЛЯ КОНВЕРТАЦИИ В ДРУГИЕ ФОРМАТЫ
# ============================================================================

def convert_to_csv(flights_data, output_file):
    """Конвертация в CSV формат"""
    with open(output_file, 'w', newline='', encoding='utf-8') as csvfile:
        fieldnames = ['number', 'route', 'status', 'load_factor', 'delay_minutes', 'fuel_consumption']
        writer = csv.DictWriter(csvfile, fieldnames=fieldnames)

        writer.writeheader()
        for flight in flights_data:
            writer.writerow({
                'number': flight['number'],
                'route': flight['route'],
                'status': flight['status'],
                'load_factor': flight['daily_metrics']['load_factor'],
                'delay_minutes': flight['daily_metrics']['delay_minutes'],
                'fuel_consumption': flight['daily_metrics']['fuel_consumption']
            })
    print(f"Данные сохранены в CSV: {output_file}")


def convert_to_ini(flights_data, output_file):
    """Конвертация в INI формат"""
    config = configparser.ConfigParser()

    for flight in flights_data:
        section = f"Flight_{flight['number'].replace('-', '_')}"
        config[section] = {
            'route': flight['route'],
            'status': flight['status'],
            'load_factor': str(flight['daily_metrics']['load_factor']),
            'delay': str(flight['daily_metrics']['delay_minutes'])
        }

    with open(output_file, 'w', encoding='utf-8') as f:
        config.write(f)
    print(f"Данные сохранены в INI: {output_file}")


def convert_all_formats(flights_data):
    """Конвертация во все форматы"""
    print("\n=== КОНВЕРТАЦИЯ В ДРУГИЕ ФОРМАТЫ ===")
    convert_to_csv(flights_data, "flights_data.csv")
    convert_to_ini(flights_data, "flights_data.ini")


# ============================================================================
# ОСНОВНАЯ ФУНКЦИЯ ДЛЯ ЗАПУСКА ВСЕЙ СИСТЕМЫ
# ============================================================================

def main():
    """Основная функция для запуска всей системы анализа авиарейсов"""

    # Проверка существования файла
    toml_file = "data.toml"
```

```
    if not os.path.exists(toml_file):
        print(f"❌ Файл {toml_file} не найден!")
        print("Убедитесь, что файл aviation_data.toml находится в той же папке, что и этот скрипт.")
        return

    print(f"✅ Найден файл с данными: {toml_file}")

    # Инициализация анализатора
    analyzer = FlightAnalyzer(toml_file)

    # Проверка загрузки данных
    if not analyzer.flights:
        print("❌ Не удалось загрузить данные из файла")
        return

    print(f"✅ Успешно загружено рейсов: {len(analyzer.flights)}")

    # ЭТАП 2: Базовый анализ данных
    analyzer.generate_summary_report()

    # ЭТАП 3: Фильтрация и группировка
    filter_system = FlightFilter(analyzer.flights)
    filter_system.print_filter_results()

    # ЭТАП 4: Генерация отчетов
    report_generator = ReportGenerator(analyzer.flights)
    report_generator.generate_all_reports()

    # ЭТАП 5: Валидация данных
    validator = DataValidator(analyzer.flights)
    validator.print_validation_report()

    # Конвертация в другие форматы
    convert_all_formats(analyzer.flights)

    print("\n" + "=" * 50)
    print("✅ ВСЕ ЭТАПЫ ВЫПОЛНЕНЫ УСПЕШНО!")
    print("=" * 50)


# Запуск основной функции
if __name__ == "__main__":
    main()
```

Скриншоты выполнения программы: