Московский государственный технический
университет им. Н. Э. Баумана

Факультет «Информатика и системы управления»
Кафедра ИУ5 «Системы обработки информации и
управления»

Курс «Парадигмы и конструкции языков
программирования»
Реферат на тему:
«Изучение языка программирования Python. Разработка консольного приложения «Персональный менеджер задач»

Выполнил:
Студент группы ИУ5И-31Б
Хамет Виам

Проверил:
Гапанюк Ю. Е.

2025 г.

1. Описание задания

Цель: Изучить новый язык программирования (Python) через практическую реализацию проекта.

Задачи:

1. Установить интерпретатор Python и среду разработки (IDE).
2. Изучить базовый синтаксис, структуры данных и парадигмы языка Python.
3. Разработать консольное приложение «Персональный менеджер задач», реализующее базовые операции CRUD (Create, Read, Update, Delete).
4. Использовать встроенные возможности Python для работы с файлами (для хранения данных).
5. Детально прокомментировать код.

Выбор языка: Python — высокоуровневый язык общего назначения с ясным, читаемым синтаксисом. Поддерживает несколько парадигм: императивное, объектно-ориентированное и функциональное программирование. Является интерпретируемым языком. Не изучался мной ранее в рамках академического курса.

Установленное ПО:

1. Интерпретатор: Python 3.12 с официального сайта python.org.
2. Среда разработки (IDE): Visual Studio Code с расширением Python.

2. Проект: «Персональный менеджер задач» (To-Do List Manager)

2.1. Описание проекта

Проект представляет собой простое консольное приложение для управления списком личных задач. Задачи хранятся в текстовом файле tasks.txt в формате JSON, что позволяет сохранять данные между запусками программы.

Основные функции (возможности):

1. Просмотр всех задач (Read).
2. Добавление новой задачи (Create).
3. Отметка задачи как выполненной (Update).

4. Удаление задачи (Delete).
5. Автоматическое сохранение и загрузка списка из файла.

2.2. Текст программы с детальными комментариями

```
"""
Персональный менеджер задач (To-Do List Manager)
Автор: [Ваше ФИО]
Язык: Python 3
"""
import json  # Модуль для работы с JSON форматом (сохранение/загрузка данных)
import os    # Модуль для взаимодействия с операционной системой (проверка файлов)

# Имя файла для хранения задач
FILE_NAME = "tasks.txt"

def load_tasks():
    """
    Функция для загрузки списка задач из файла.
    Использует парадигму структурного программирования.

    Возвращает:
        list: Список словарей, каждый из которых представляет одну задачу.
              Если файл не существует или пуст, возвращает пустой список.
    """
    # Проверяем, существует ли файл
    if not os.path.exists(FILE_NAME):
        return []  # Возвращаем пустой список, если файла нет

    try:
        # Открываем файл для чтения ('r' - read)
        with open(FILE_NAME, 'r', encoding='utf-8') as file:
            # Загружаем данные из JSON формата в объект Python
            tasks = json.load(file)
            # Убеждаемся, что это список
            if isinstance(tasks, list):
                return tasks
            else:
                return []
    except (json.JSONDecodeError, FileNotFoundError):
        # Если файл поврежден или пуст, возвращаем пустой список
        return []

def save_tasks(tasks):
    """
    Функция для сохранения списка задач в файл.

    Аргументы:
        tasks (list): Список задач для сохранения.
    """
    # Открываем файл для записи ('w' - write). Файл будет создан, если его нет.
    with open(FILE_NAME, 'w', encoding='utf-8') as file:
        # Преобразуем список задач в JSON строку с отступами (indent=4) для читаемости
        json.dump(tasks, file, ensure_ascii=False, indent=4)

def display_tasks(tasks):
    """
    Функция для отображения всех задач в консоли.
```

```
    Аргументы:
        tasks (list): Список задач для отображения.
    """
    if not tasks:  # Если список пуст
        print("\n--- Список задач пуст. Добавьте первую задачу! ---")
        Return

    print("\n" + "="*40)
    print("МОЙ СПИСОК ЗАДАЧ".center(40))
    print("="*40)

    # Используем цикл for для итерации по списку задач.
    # Функция enumerate() возвращает индекс и элемент списка.
    for index, task in enumerate(tasks, start=1):
        # Тернарный оператор для выбора статуса задачи
        status = "✓" if task['completed'] else "✗"
        print(f"{index}. [{status}] {task['title']}")
    print("="*40)

def add_task(tasks):
    """
    Функция для добавления новой задачи.
    Демонстрирует работу с вводом пользователя и модификацией списка.

    Аргументы:
        tasks (list): Список задач, который будет изменен.
    """
    print("\n--- Добавление новой задачи ---")
    # Получаем описание задачи от пользователя
    title = input("Введите описание задачи: ").strip()

    if title:  # Проверяем, что строка не пустая
        # Создаем словарь, представляющий задачу
        new_task = {
            'title': title,
            'completed': False  # Новая задача по умолчанию не выполнена
        }
        # Добавляем задачу в конец списка (метод append)
        tasks.append(new_task)
        save_tasks(tasks)  # Сохраняем изменения в файл
        print(f"Задача '{title}' успешно добавлена!")
    else:
        print("Ошибка: описание задачи не может быть пустым.")

def complete_task(tasks):
    """
    Функция для отметки задачи как выполненной.
    Демонстрирует доступ к элементам списка по индексу и изменение словаря.

    Аргументы:
        tasks (list): Список задач.
    """
    if not tasks:
        print("Список задач пуст. Нечего отмечать как выполненное.")
        Return

    display_tasks(tasks)
    try:
        # Получаем номер задачи от пользователя (преобразуем строку в число)
        task_num = int(input("\nВведите номер задачи для отметки как выполненной: "))
```

```
        # Проверяем, что номер находится в допустимом диапазоне
        if 1 <= task_num <= len(tasks):
            # Индексы в Python начинаются с 0, поэтому вычитаем 1
            tasks[task_num - 1]['completed'] = True
            save_tasks(tasks)  # Сохраняем изменения
            print(f"Задача №{task_num} отмечена как выполненная!")
        else:
            print(f"Ошибка: введите номер от 1 до {len(tasks)}.")
    except ValueError:  # Обработка исключения, если введено не число
        print("Ошибка: пожалуйста, введите корректный номер.")

def delete_task(tasks):
    """
    Функция для удаления задачи из списка.
    Демонстрирует удаление элемента списка по индексу.

    Аргументы:
        tasks (list): Список задач, который будет изменен.
    """
    if not tasks:
        print("Список задач пуст. Нечего удалять.")
        Return

    display_tasks(tasks)
    try:
        task_num = int(input("\nВведите номер задачи для удаления: "))

        if 1 <= task_num <= len(tasks):
            # Метод pop() удаляет элемент по индексу и возвращает его
            removed_task = tasks.pop(task_num - 1)
            save_tasks(tasks)  # Сохраняем изменения
            print(f"Задача '{removed_task['title']}' удалена.")
        else:
            print(f"Ошибка: введите номер от 1 до {len(tasks)}.")
    except ValueError:
        print("Ошибка: пожалуйста, введите корректный номер.")

def main():
    """
    Главная функция программы. Реализует основной цикл меню.
    Демонстрирует использование условных операторов и циклов в Python.
    """
    # Загружаем задачи при запуске программы
    tasks = load_tasks()

    # Бесконечный цикл для отображения меню
    while True:
        print("\n" + "-"*40)
        print("ПЕРСОНАЛЬНЫЙ МЕНЕДЖЕР ЗАДАЧ".center(40))
        print("-"*40)
        print("1. Показать все задачи")
        print("2. Добавить новую задачу")
        print("3. Отметить задачу как выполненную")
        print("4. Удалить задачу")
        print("5. Выйти")
        print("-"*40)

        # Получаем выбор пользователя
        choice = input("Выберите действие (1-5): ").strip()

        # Условный оператор для выбора функции
```

```
        if choice == '1':
            display_tasks(tasks)
        elif choice == '2':
            add_task(tasks)
        elif choice == '3':
            complete_task(tasks)
        elif choice == '4':
            delete_task(tasks)
        elif choice == '5':
            print("\nСпасибо за использование менеджера задач. До свидания!")
            break  # Выход из цикла и завершение программы
        else:
            print("Ошибка: пожалуйста, выберите число от 1 до 5.")

# Стандартная конструкция для точки входа в программу на Python
if __name__ == "__main__":
    main()
```

2.3. Экранные формы (Пример работы программы)

3. Особенности языка Python, использованные в проекте

1. Интерпретируемость и простота синтаксиса: Не требуется объявления переменных, точка с запятой не нужна. Код очень читаем.
2. Динамическая типизация: Тип переменной (tasks, title) определяется во время выполнения.
3. Структуры данных высокого уровня: Активно использованы list (список) для хранения задач и dict (словарь) для представления отдельной задачи. Это мощные встроенные типы.

4. Управляющие конструкции: Применены цикл while, цикл for с enumerate(), условный оператор if-elif-else.
5. Работа с файлами: Использован контекстный менеджер with open(...) as file, который автоматически закрывает файл. Это идиоматический подход в Python.
6. Обработка исключений: Использованы блоки try-except для обработки ошибок (некорректный ввод числа, поврежденный файл).
7. Функциональная парадигма: Программа структурирована как набор функций, что делает код модульным и переиспользуемым.
8. Стандартная библиотека: Использованы модули json и os, которые являются частью богатой стандартной библиотеки Python.
9. Строковые методы: strip(), center().
10. Тернарный оператор: Использован для компактного отображения статуса задачи status = "✓" if task['completed'] else "✗".

4. Инструкция по запуску

1. Установите Python 3.x с официального сайта.
2. Скопируйте приведенный выше код в файл с расширением .py, например, todo_manager.py.
3. Откройте терминал (командную строку) в папке с файлом.
4. Выполните команду: python todo_manager.py
5. Следуйте инструкциям в меню.

5. Заключение

В ходе выполнения домашнего задания был успешно изучен язык программирования Python через практическую разработку. Созданное консольное приложение «Персональный менеджер задач» демонстрирует понимание основных конструкций языка: работу с переменными, списками, словарями, функциями, циклами, условными операторами, файловым вводом-выводом и обработкой исключений. Проект является полностью функциональным и готов к использованию. Python подтвердил свою репутацию как языка с низким порогом входа и высокой выразительностью кода.