5 ЛЕКЦИЯ: РАБОТА С ФАЙЛАМИ И ВВОД/ВЫВОД В PYTHON

Работа с файлами и вводом/выводом (I/O) является одной из основных задач программирования, особенно когда речь идет о сохранении и загрузке данных. В Python это реализовано через встроенные функции и модули, что делает работу с файлами достаточно простой и интуитивной. В этой лекции мы рассмотрим основные аспекты работы с файлами, включая чтение и запись данных, а также взаимодействие с операционной системой.

1. Основы работы с файлами

1.1. Открытие файла

В Python для работы с файлами используется встроенная функция `open()`. Эта функция принимает два основных параметра: имя файла и режим открытия.

```
file = open('example.txt', 'r')  # 'r' - режим чтения
```

1.2. Режимы открытия файлов

- `'r'` — Чтение (по умолчанию). Файл должен существовать.
- `'w'` — Запись. Если файл существует, он будет переписан. Если нет — создан.
- `'a'` — Добавление. Данные будут добавляться в конец файла.
- `'b'` — Бинарный режим. Используется для работы с бинарными файлами.
- `'x'` — Исключительная запись. Создает новый файл и вызывает ошибку, если файл с таким именем уже существует.

1.3. Закрытие файла

После завершения работы с файлом, его необходимо закрыть с помощью метода `close()`. Это помогает освободить ресурсы.

```
file.close()
```

1.4. Использование контекстного менеджера

Рекомендуется использовать контекстный менеджер `with`, который автоматически закрывает файл после выхода из блока.

```
with open('example.txt', 'r') as file:
    content = file.read()
    print(content)
# Файл автоматически закрывается здесь
```

2. Чтение данных из файла

2.1. Чтение всего содержимого

Метод `read()` позволяет считать все содержимое файла в одну строку.

```
with open('example.txt', 'r') as file:
    content = file.read()
    print(content)
```

2.2. Чтение построчно

Методы `readline()` и `readlines()` позволяют считывать данные построчно.

- `readline()` считывает одну строку.
- `readlines()` считывает все строки и возвращает их в виде списка.

```
with open('example.txt', 'r') as file:
```

```
    for line in file:
        print(line.strip())  # strip() удаляет символы новой строки
```

3. Запись данных в файл

3.1. Запись строк в файл

Для записи данных в файл используется метод `write()`.

```
with open('example.txt', 'w') as file:
    file.write('Hello, World!\n')
```

3.2. Запись списка строк

Метод `writelines()` позволяет записать список строк в файл.

```
lines = ['Line 1
', 'Line 2
', 'Line 3
']
with open('example.txt', 'w') as file:
    file.writelines(lines)
```

4. Работа с бинарными файлами

Для работы с бинарными файлами используются те же методы, но с режимом `'b'`.

Пример чтения и записи бинарного файла:

```
# Запись в бинарный файл
with open('example.bin', 'wb') as file:
    file.write(b'\x00\x01\x02')
# Чтение из бинарного файла
with open('example.bin', 'rb') as file:
    content = file.read()
    print(content)
```

5. Взаимодействие с операционной системой

Python предоставляет модуль `os`, который позволяет взаимодействовать с файловой системой.

5.1. Проверка существования файла

```
import os
if os.path.exists('example.txt'):
    print('Файл существует')
else:
    print('Файл не найден')
```

5.2. Удаление файла

```
os.remove('example.txt')
```

5.3. Создание директорий

```
os.mkdir('new_directory')
```

5.4. Перемещение и копирование файлов

Для копирования файлов используется модуль `shutil`.

```
import shutil
```

```
shutil.copy('example.txt', 'backup/example.txt')
```

Заключение

Работа с файлами и вводом/выводом в Python — это важная часть программирования, которая позволяет сохранять и загружать данные. Мы рассмотрели основные методы работы с текстовыми и бинарными файлами, а также взаимодействие с операционной системой. Умение работать с файлами открывает множество возможностей для создания эффективных и функциональных приложений.