Основы работы с файлами в Python. Создание, чтение и запись.

Работа с файлами в Python — важный навык для каждого разработчика. Правильное открытие и закрытие файлов обеспечивает корректную работу программы и эффективное использование системных ресурсов.

Режимы открытия файлов

Выбор правильного режима открытия файла критически важен для корректной работы программы:

Режим	Описание
'r'	Открытие файла для чтения (режим по умолчанию)
'w'	Запись в файл (создает новый файл или перезаписывает существующий)
'a'	Добавление данных в конец файла
'x'	Создание нового файла (завершается ошибкой, если файл существует)
'rb'	Чтение файла в бинарном режиме
'wb'	Запись в файл в бинарном режиме
'ab'	Добавление в файл в бинарном режиме
'r+'	Чтение и запись в существующий файл
'w+'	Чтение и запись (создает новый файл или перезаписывает)
'a+'	Чтение и добавление в конец файла

Текстовые файлы в Python

Текстовые файлы содержат данные в виде символов, которые можно прочитать и интерпретировать человеком. К таким файлам относятся документы с расширениями .txt, .csv, .json, .html и другие файлы, содержащие текстовую информацию.

Режимы доступа к текстовым файлам

Python предоставляет следующие режимы для работы с текстовыми файлами:

– 'r' - чтение файла (файл должен существовать)

– 'w' - запись в файл (создает новый файл или перезаписывает существующий)

– 'a' - добавление данных в конец файла (создает новый файл или добавляет данные в существующий)

– 'r+' - чтение и запись (файл должен существовать)

– 'w+' - запись и чтение (создает новый файл или перезаписывает существующий)

– 'a+' - добавление и чтение (создает новый файл или добавляет данные в существующий)

Бинарные файлы в Python

Бинарные файлы содержат данные в виде последовательности байтов, которые не предназначены для чтения человеком. К таким файлам относятся изображения (.jpg, .png), аудиофайлы (.mp3, .wav), видеофайлы (.mp4, .avi), архивы (.zip, .rar) и исполняемые файлы.

Режимы доступа к бинарным файлам

Для работы с бинарными файлами используются следующие режимы:

– 'rb' - чтение бинарного файла (файл должен существовать)

– 'wb' - запись в бинарный файл (создает новый файл или перезаписывает существующий)

– 'ab' - добавление данных в конец бинарного файла

– 'rb+' - чтение и запись бинарного файла (файл должен существовать)

– 'wb+' - запись и чтение бинарного файла (создает новый файл или перезаписывает существующий)

– 'ab+' - добавление и чтение бинарного файла

Основы работы с файлами

Для создания новых файлов в Python используется встроенная функция open() с различными режимами открытия. Основная структура работы с файлами выглядит следующим образом:

```
# Создание нового текстового файла
with open('новый_файл.txt', 'режим') as f:
    pass  # В этом примере файл будет пустым, так как мы ничего
в него не записываем
```

Использование конструкции with гарантирует автоматическое закрытие файла после завершения работы, даже если произойдет ошибка.

Управление файлами с помощью модуля shutil

Копирование файлов

```
import shutil

# Копирование файла
shutil.copy('исходный_файл.txt', 'новый_файл.txt')

# Копирование файла с сохранением метаданных
shutil.copy2('исходный_файл.txt', 'новый_файл.txt')
```

```
# Копирование всего дерева каталогов
shutil.copytree('исходная_папка', 'целевая_папка')
```

Перемещение и переименование файлов

```
import os
import shutil

# Переименование файла
os.rename('старое_имя.txt', 'новое_имя.txt')

# Перемещение файла
shutil.move('исходный_файл.txt', 'целевая_папка/новое_имя.txt')
```

Удаление файлов и проверка существования

Удаление файлов

```
import os

# Удаление файла
try:
    os.remove('файл_для_удаления.txt')
    print('Файл успешно удален')
except FileNotFoundError:
    print('Файл не найден')
except PermissionError:
    print('Недостаточно прав для удаления файла')
```

Проверка существования файла

```
import os.path

# Проверка существования файла
if os.path.exists('файл.txt'):
    print('Файл существует')
else:
    print('Файл не существует')

# Проверка, является ли путь файлом
if os.path.isfile('файл.txt'):
    print('Это файл')

# Проверка, является ли путь директорией
if os.path.isdir('папка'):
    print('Это директория')
```

Поиск файлов

Поиск файлов по расширению или имени

```python
import glob
import os

# Поиск файлов с расширением ".txt"
txt_files = glob.glob('*.txt')
print('Найденные txt файлы:', txt_files)

# Поиск файлов в подпапках
all_txt_files = glob.glob('**/*.txt', recursive=True)

# Поиск файлов с помощью os.listdir()
files_in_directory = [f for f in os.listdir('.') if
f.endswith('.txt')]
```

Получение информации о файле

```python
import os.path
import time

# Получение размера файла в байтах
size = os.path.getsize('файл.txt')
print(f'Размер файла: {size} байт')

# Получение времени последнего доступа к файлу
atime = os.path.getatime('файл.txt')
print(f'Последний доступ: {time.ctime(atime)}')

# Получение времени последнего изменения файла
mtime = os.path.getmtime('файл.txt')
print(f'Последнее изменение: {time.ctime(mtime)}')

# Получение абсолютного пути к файлу
abs_path = os.path.abspath('файл.txt')
print(f'Абсолютный путь: {abs_path}')
```

Обработка ошибок при работе с файлами

При работе с файлами важно предусматривать обработку возможных ошибок:

```python
import os

def safe_file_operation(filename):
    try:
        with open(filename, 'r', encoding='utf-8') as f:
            content = f.read()
            return content
    except FileNotFoundError:
        print(f'Файл {filename} не найден')
    except PermissionError:
```

```
        print(f'Недостаточно прав для чтения файла {filename}')
    except UnicodeDecodeError:
        print(f'Ошибка кодировки при чтении файла {filename}')
    except Exception as e:
        print(f'Неожиданная ошибка: {e}')
    return None
```

Лучшие практики работы с файлами

1. Всегда используйте контекстный менеджер with для автоматического закрытия файлов
2. Указывайте кодировку при работе с текстовыми файлами (обычно encoding='utf-8')
3. Обрабатывайте исключения для предотвращения аварийного завершения программы
4. Проверяйте существование файлов перед операциями чтения
5. Используйте соответствующие модули (shutil для копирования, os для базовых операций)

Работа с файлами в Python предоставляет мощные возможности для управления данными. Правильное использование режимов открытия файлов и обработка исключений помогут создать надежные и эффективные программы.

Чтение и запись файлов

Функция open() для работы с файлами

Функция open() является основным инструментом для работы с файлами в Python. Она принимает два обязательных параметра: путь к файлу и режим доступа.

```
file = open("example.txt", "r")
```

Контекстный менеджер with — лучшая практика

Использование контекстного менеджера with является лучшей практикой при работе с файлами в Python. Он обеспечивает автоматическое закрытие файла после завершения операций, даже если в процессе работы возникнет исключение.

```
with open("example.txt", "r") as file:
    data = file.read()
    print(data)
# Файл автоматически закрывается после выхода из блока
```

Полное чтение файла методом read()

Для чтения всего содержимого файла используется метод read(), который возвращает данные в виде строки. Этот метод оптимален для небольших файлов, которые помещаются в оперативную память.

```
with open("example.txt", "r", encoding="utf-8") as file:
    data = file.read()
    print(data)
```

Важно: Всегда указывайте кодировку encoding="utf-8" для корректной работы с русскими символами.

Построчное чтение файла

Метод readline()

Читает файл по одной строке за вызов, что позволяет экономить память при работе с большими файлами:

```
with open("example.txt", "r", encoding="utf-8") as file:
    line = file.readline()
    while line:
        print(line.strip())  # strip() убирает символы перевода строки
        line = file.readline()
```

Метод readlines()

Читает все строки файла и возвращает их в виде списка:

```
with open("example.txt", "r", encoding="utf-8") as file:
    lines = file.readlines()
    for line in lines:
        print(line.strip())
```

Итерация по файлу (рекомендуемый способ)

Самый эффективный способ построчного чтения:

```
with open("example.txt", "r", encoding="utf-8") as file:
    for line in file:
        print(line.strip())
```

Запись данных в файлы Python

Полная перезапись файла

Режим 'w' полностью перезаписывает содержимое файла:

```
with open("example.txt", "w", encoding="utf-8") as file:
    file.write("Привет, мир!\n")
    file.write("Это тестовый файл.")
```

Добавление данных в конец файла

Режим 'a' (append) добавляет новые данные в конец файла, сохраняя существующее содержимое:

```
with open("example.txt", "a", encoding="utf-8") as file:
    file.write("\nЭта строка будет добавлена в конец файла.")
```

Запись списка строк

Для записи нескольких строк используйте метод writelines():

```
lines = ["Первая строка\n", "Вторая строка\n", "Третья строка\n"]
with open("example.txt", "w", encoding="utf-8") as file:
    file.writelines(lines)
```

Работа с бинарными файлами

Чтение бинарных файлов

```
with open("image.jpg", "rb") as file:
    binary_data = file.read()
    print(f"Размер файла: {len(binary_data)} байт")
```

Запись бинарных данных

```
with open("image.jpg", "wb") as file:
    file.write(binary_data)
```

Копирование бинарных файлов

```
with open("source.jpg", "rb") as source:
    with open("copy.jpg", "wb") as destination:
        destination.write(source.read())
```

Обработка исключений при работе с файлами

Всегда используйте обработку исключений для надежной работы с файлами:

```
try:
    with open("example.txt", "r", encoding="utf-8") as file:
        content = file.read()
        print(content)
except FileNotFoundError:
    print("Файл не найден")
except PermissionError:
    print("Нет прав доступа к файлу")
except Exception as e:
    print(f"Произошла ошибка: {e}")
```

Проверка существования файла

Перед работой с файлом проверьте его существование:

```
import os

if os.path.exists("example.txt"):
    with open("example.txt", "r", encoding="utf-8") as file:
        print(file.read())
else:
    print("Файл не существует")
```

Работа с путями к файлам

Альтернативный способ с использованием pathlib. Используйте модуль pathlib для удобной работы с путями:

```
from pathlib import Path

file_path = Path("data") / "example.txt"
if file_path.exists():
```

```
    with open(file_path, "r", encoding="utf-8") as file:
        content = file.read()
```

Кодировка файлов

При работе с текстовыми файлами важно указывать правильную кодировку:

```
# Явное указание кодировки UTF-8
with open("example.txt", "r", encoding="utf-8") as file:
    content = file.read()

# Для Windows может потребоваться cp1251
with open("example.txt", "r", encoding="cp1251") as file:
    content = file.read()
```

Практические примеры

Подсчет строк в файле

```
def count_lines(filename):
    try:
        with open(filename, "r", encoding="utf-8") as file:
            return sum(1 for line in file)
    except FileNotFoundError:
        return 0

print(f"Количество строк: {count_lines('example.txt')}")
```

Поиск текста в файле

```
def search_in_file(filename, search_text):
    try:
        with open(filename, "r", encoding="utf-8") as file:
            for line_num, line in enumerate(file, 1):
                if search_text in line:
                    print(f"Строка {line_num}: {line.strip()}")
    except FileNotFoundError:
        print("Файл не найден")
```

Правильное закрытие файлов

Закрытие файла методом close() освобождает системные ресурсы и гарантирует сохранение всех изменений:

```
file = open("example.txt", "r")
# работа с файлом
file.close()
```

Работа с большими файлами

При работе с большими файлами важно экономить память, читая файл частями:

```python
def read_large_file(filename, chunk_size=1024):
    """Чтение большого файла по частям"""
    with open(filename, "r", encoding="utf-8") as file:
        while True:
            chunk = file.read(chunk_size)
            if not chunk:
                break
            yield chunk
```

Рекомендации по работе с файлами

1. Всегда используйте контекстный менеджер with для автоматического закрытия файлов
2. Указывайте кодировку encoding="utf-8" при работе с текстовыми файлами
3. Обрабатывайте исключения для повышения надежности кода
4. Для больших файлов используйте построчное чтение для экономии памяти
5. Проверяйте существование файлов перед попыткой их открытия

Эти методы и подходы обеспечат эффективную и безопасную работу с файлами в Python, независимо от их размера и содержимого.