IT CAREER HUB

Урок 27.
Основы работы с файлами

Введение в работу с файлами	**1**
Типы файлов: текстовые и бинарные	**3**
Функция open	**4**
Режимы работы с файлами	**5**
Параметр encoding	**6**
Функция close	**8**
Задания для закрепления 1	**9**
Контекстный менеджер with	**10**
Чтение из файла	**11**
Запись в файл	**15**
Другие популярные режимы	**17**
Контекстный менеджер с несколькими файлами	**19**
Задания для закрепления 2	**20**
Ответы на задания	**21**
Практическая работа	**22**

IT CAREER HUB

Введение в работу с файлами

Работа с файлами — важная часть программирования, поскольку данные часто хранятся во внешних файлах. Python позволяет создавать, читать, изменять и удалять файлы.

Почему важно уметь работать с файлами?

1. **Хранение данных**
   - Файлы позволяют сохранять информацию между запусками программы.
   - Например, конфигурационные файлы, базы данных, отчёты.
2. **Обмен данными**
   - Файлы используются для передачи данных между программами (CSV, JSON, XML).
   - Например, работа с логами или загрузка файлов в веб-приложениях.
3. **Обработка больших объёмов информации**
   - Данные могут быть слишком объёмными для хранения в оперативной памяти.
   - Файлы позволяют считывать информацию по частям.
4. **Автоматизация работы с документами**
   - Скрипты могут автоматически создавать и изменять файлы, избавляя от ручной работы.
   - Например, генерация отчётов, обработка логов, создание резервных копий.

IT CAREER HUB

Типы файлов: текстовые и бинарные

Все файлы в операционной системе делятся на два типа: **текстовые** и **бинарные**.

1. Текстовые файлы

- Хранят информацию в виде обычного текста (читаемого человеком).
- Используют кодировки (UTF-8, CP1251 и др.).
- Примеры: .txt, .csv, .json, .html, .py.

Пример текстового файла data.txt

Имя, Возраст
Анна, 25
Иван, 30

2. Бинарные файлы

- Хранят данные в двоичном формате (не читаемом человеком).
- Используются для хранения изображений, видео, исполняемых файлов, баз данных.
- Примеры: .jpg, .png, .mp4, .exe, .zip.

Пример бинарного файла image.jpg

\xFF\xD8\xFF\xE0\x00\x10JFIF\x00\x01\x01\x00\x00\x01\x00\x01\x00\x00

IT CAREER HUB

Функция open

Функция `open()` используется для работы с файлами в Python. Она позволяет открывать файлы для чтения и записи данных, включая текстовую и бинарную информацию. Функция возвращает объект файла.

Синтаксис

```python
open(file, mode="mode", encoding="encoding")
```

- `file` – путь к файлу (относительный или абсолютный).
- `mode` *(необязательный)* – режим работы с файлом (`"r"`, `"w"`, `"a"`, `"b"` и др.).
- `encoding` *(необязательный)* – кодировка текста (`"utf-8"`, `"cp1251"` и др.). Используется только для текстовых файлов.

IT CAREER HUB

Режимы работы с файлами

При открытии файла с помощью `open()` необходимо указать режим работы.
По умолчанию файл открывается в режиме `"rt"` (чтение текста).

Режим	Описание
`"r"`	Чтение (по умолчанию). Ошибка, если файла нет.
`"w"`	Запись. Создаёт новый файл или перезаписывает существующий.
`"a"`	Добавление в конец файла. Создаёт новый, если его нет.
`"x"`	Создание нового файла. Ошибка, если файл уже существует.
`"b"`	Бинарный режим (например, `"rb"`, `"wb"` для изображений).
`"t"`	Текстовый режим (по умолчанию).
`"+"`	Открытие файла как для чтения, так и для записи (`"r+"`, `"w+"`, `"a+"`, `"x+"`).

Важно: Режимы можно комбинировать, например:

- `"rb"` – чтение в бинарном режиме.
- `"wt"` – запись в текстовом режиме.
- `"a+"` – добавление и чтение.

IT CAREER HUB

Параметр encoding

Параметр `encoding` в функции `open()` используется для указания кодировки текста при чтении или записи файлов. Он особенно важен при работе с файлами, содержащими символы, отличные от стандартных ASCII.

Почему `encoding` важен?

1. **Обеспечивает правильное чтение и запись файлов**
   - Без указания кодировки могут возникнуть ошибки при работе с текстами, содержащими кириллицу, умлауты, азиатские символы и т. д.
2. **Универсальность кода**
   - Разные системы могут использовать разные кодировки по умолчанию. Например, Windows часто использует cp1251 для русского текста, а Linux/macOS – utf-8.
3. **Предотвращает ошибки `UnicodeDecodeError`**
   - Если файл записан в одной кодировке, но читается в другой, может возникнуть ошибка.

Популярные кодировки

Кодировка	**Описание**
utf-8 (по умолчанию)	Универсальная кодировка, поддерживает все языки.
cp1251	Кодировка Windows для русского языка.
ascii	Ограничена английскими символами (0-127), устаревшая.

utf-16	Двухбайтовая кодировка, встречается в Windows-файлах.
utf-32	Четырёхбайтовая кодировка, занимает больше места.
iso-8859-1 (Latin-1)	Кодировка для западноевропейских языков.

Особенности encoding

- Если `encoding` не указан, Python использует **кодировку по умолчанию**, которая зависит от операционной системы.
- Используется **только для текстовых файлов** (при `"rb"` и `"wb"` `encoding` не нужен).
- Неверная кодировка может вызвать `UnicodeDecodeError`.

IT CAREER HUB

Функция close

Функция `close()` используется для закрытия файла после его открытия с помощью `open()`. Закрытие файла необходимо для освобождения ресурсов и предотвращения возможных ошибок при повторном доступе к нему.

Пример

```python
# Можно указать просто "r", а не mode="r"
file = open("example.txt", "r", encoding="utf-8")  # Открываем файл для чтения
content = file.read()  # Читаем содержимое файла
print(content)
file.close()  # Закрываем файл вручную
```

Зачем закрывать файл?

1. **Освобождение системных ресурсов**
   - Открытые файлы занимают память и могут блокировать доступ к файлу для других программ.
2. **Гарантированное сохранение данных**
   - В некоторых случаях изменения могут не записаться сразу, а только при закрытии файла.
3. **Предотвращение ошибок доступа**
   - В Windows файл может оставаться заблокированным, если его не закрыть явно.

IT CAREER HUB

Задания для закрепления 1

1. **Какой режим работы с файлом приведёт к ошибке, если файл уже существует?**
   a. `"w"`
   b. `"r"`
   c. `"x"`
   d. `"a"`

2. **Какой параметр обязателен для open()? Перечислите все варианты.**
   a. `mode`
   b. `encoding`
   c. `file`
   d. `buffering`

Посмотреть ответы

IT CAREER HUB

Контекстный менеджер with

Для работы с файлами используется встроенная функция `open()`, после чего файл необходимо вручную закрыть с помощью метода `close()`. Однако, если забыть закрыть файл, это может привести к утечке ресурсов.

Контекстный менеджер `with` используется для безопасной работы с файлами, автоматически закрывая файл после выхода из блока. Это предпочтительный способ работы с файлами в Python.

Зачем использовать with open()?

1. **Автоматическое закрытие файла**
    - Файл закрывается даже в случае ошибки в блоке `with`, предотвращая утечки ресурсов.
2. **Код становится чище**
    - Не нужно вручную вызывать `close()`, что уменьшает вероятность ошибок.

Синтаксис

```python
with open(file, mode="mode", encoding="encoding") as variable:
    # Операции с файлом
```

- `with` — ключевое слово для работы с контекстными менеджерами.
- `open` — функция для открытия файла, возвращающая объект для работы с ним.
- `as` — ключевое слово для присваивания объекта файла переменной.
- `variable` — переменная, ссылающаяся на объект файла, через которую можно работать с ним в блоке `with`.

Пример

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

Особенности

- Файл открывается только внутри блока `with`.
- После выхода из блока файл **автоматически закрывается**.

Чтение из файла

Python предоставляет несколько способов чтения данных из файла. Когда файл открывается, **указатель** устанавливается в **начало файла**, и по мере чтения он смещается вперёд.

Указатель файла — это позиция, с которой выполняется следующее чтение или запись.

1. Чтение всего содержимого файла

Метод `read()` позволяет считать весь текст из файла в одну строку.

Пример

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

Особенности:

- Читает весь файл целиком.
- Если файл большой, может занять много памяти.
- Указатель после чтения перемещается в конец файла.
- Переносы строк указываются как `\n`.

2. Чтение первых n символов

 **Метод `read(n)` позволяет считать указанное количество символов.**

 **Пример**

```python
with open("example.txt", "r", encoding="utf-8") as file:
    content = file.read(10)  # Читаем первые 10 символов
    print(content)
```

Особенности:

- Позволяет контролировать объём загружаемых данных.
- Полезно при чтении больших файлов по частям.

3. Чтение построчно

 **Метод `readline()` читает одну строку за вызов.**

 **Пример**

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

Особенности:

- Читает одну строку из файла.
- Каждый вызов `readline()` перемещает указатель дальше.
- Если в файле несколько строк, можно вызывать `readline()` повторно.

4. Чтение всех строк

 **Метод `readlines()` загружает все строки файла в список.**

 **Пример**

```python
with open("example.txt", "r", encoding="utf-8") as file:
    lines = file.readlines()
    print(type(lines))
    print(lines)
```

Особенности:

- Возвращает список, где каждая строка – отдельный элемент.
- Учитывает символ `\n` в конце строк.

5. Чтение файла в цикле

Файл является **итерируемым объектом**, поэтому можно перебирать его содержимое **построчно в цикле** без необходимости загружать весь файл в память.

 **Пример**

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

Особенности:

- Читает файл **по одной строке**, не загружая всё в память.
- Подходит для обработки больших файлов.

Запись в файл

Для записи в файл также представлено несколько методов.

1. Запись строки в файл: write()

Метод write() записывает строку в файл.

Пример

```python
with open("users.txt", "w", encoding="utf-8") as file:
    file.write("ID: 201 | Name: John | Age: 25 | Status: Active\n")
```

Особенности:

- Создаёт файл, если его нет.
- Удаляет старые данные в файле.
- `write()` не добавляет символ `\n` автоматически.

2. Запись нескольких строк: `writelines()`

Метод `writelines()` записывает список строк.

Пример

```python
data = [
    "ID: 202 | Name: Alice | Age: 30 | Status: Inactive\n",
    "ID: 203 | Name: Bob | Age: 27 | Status: Active\n"
]

with open("users.txt", "w", encoding="utf-8") as file:
    file.writelines(data)
```

Особенности:

- Принимает список строк и записывает их подряд.
- \n нужно добавлять вручную, иначе строки сольются.

Другие популярные режимы

Режим "a" – добавление в файл

Этот режим позволяет **сохранять существующее содержимое файла** и дописывать данные **в конец**.

Пример

```python
with open("users.txt", "a", encoding="utf-8") as file:
    file.write("Дополнительная строка\n")
```

Особенности:

- Создаёт файл, если его нет.
- Записывает данные **в конец файла**, а не перезаписывает его.

Режим "r+" – чтение и запись

Позволяет **и читать, и записывать** данные в файл **без удаления содержимого**.

Пример

```python
with open("users.txt", "r+", encoding="utf-8") as file:
    content = file.read()  # Читаем текущие данные
    file.write("\nДобавленный текст")  # Записываем новые данные
```

Особенности:

- Файл **должен существовать**, иначе возникнет ошибка.
- Чтение начинается с начала файла.

- Запись начинается **с текущей позиции указателя**, что может привести к частичной перезаписи данных.

Режим "x" – создание нового файла

Используется, если нужно **гарантированно создать новый файл** и избежать перезаписи существующих данных.

Пример

```python
try:
    with open("new_file.txt", "x", encoding="utf-8") as file:
        file.write("Этот файл создан в режиме 'x'.\n")
except FileExistsError:
    print("Файл уже существует.")
```

Особенности:

- Если файл **уже существует**, возникает `FileExistsError`.
- Полезно, если важно **не перезаписывать существующие данные**.

IT CAREER HUB

Контекстный менеджер с несколькими файлами

Можно открывать **несколько файлов одновременно**.

Пример

```python
with (open("example.txt", "r", encoding="utf-8") as infile,
      open("output.txt", "w", encoding="utf-8") as outfile):
    for line in infile:
        outfile.write(line.upper())  # Записываем в верхнем регистре
```

Особенности:

- Открывает сразу **два файла**.
- Читает из `input.txt` и записывает изменённый текст в `output.txt`.

IT CAREER HUB

Задания для закрепления 2

1. **Какой метод возвращает список строк из файла?**

   a. `read()`
   b. `readline()`
   c. `readlines()`
   d. `splitlines()`

2. Что делает следующий код? Перечислите все варианты.

```python
with open("data.txt", "a", encoding="utf-8") as file:
    file.write("Новая строка\n")
```

a. Перезаписывает файл и добавляет "Новая строка"
b. Добавляет "Новая строка" в конец файла
c. Очищает файл перед записью
d. Создаёт новый файл, если его нет

Посмотреть ответы

IT CAREER HUB

Ответы на задания

Задания на закрепление 1	Вернуться к заданиям
1. Режим работы файла	Ответ: c
2. Параметр, обязательный для open()	Ответ: c
Задания на закрепление 2	Вернуться к заданиям
3. Список строк из файла	Ответ: c
4. Результат выполнения кода	Ответ: b,d

IT CAREER HUB

Практическая работа

1. Подсчёт частоты слов в файле

Напишите программу, которая подсчитывает, сколько раз каждое слово встречается в файле (не учитывая регистр).

- Программа запрашивает имя файла и количество популярных слов для вывода.
- Если указанный файл не существует, программа должна вывести ошибку.

Используйте файл text.txt.

Пример ввода:

```python
Введите имя файла: text.txt
Введите количество популярных слов: 3
```

Пример вывода:

```python
Популярные слова:
python: 4
is: 3
in: 2
```

IT CAREER HUB

Решение:

```python
from collections import Counter

filename = input("Введите имя файла: ")

def count_words_in_file(filename):
    with open(filename, "r", encoding="utf-8") as file:
        text = file.read().lower()

    punctuation = '.,!?;:"-'
    for char in punctuation:
        text = text.replace(char, " ")

    return Counter(text.split())


try:
    word_counts = count_words_in_file(filename)
    num_words = int(input("Введите количество популярных слов: "))

    print("\nПопулярные слова:")
    for word, count in word_counts.most_common(num_words):
        print(f"{word}: {count}")

except FileNotFoundError:
    print("Ошибка: Файл не найден.")
except ValueError:
    print("Ошибка: Введите целое число для количества популярных слов.")
```

2. Удаление пустых строк

Напишите программу, которая удаляет пустые строки из указанного пользователем файла и записывает результат в новый файл.

- Имя нового файла формируется автоматически по шаблону `<oldname>_cleaned.txt`.
- Если указанный файл не существует, программа должна вывести ошибку.

Используйте файл tasks.txt.

Пример ввода:

```
Python
Введите имя файла: tasks.txt
```

Пример вывода:

```
Python
Пустые строки удалены, сохранено в tasks_cleaned.txt.
```

Решение:

```python
import os

filename = input("Введите имя файла: ")

# Генерация имени нового файла
new_filename = f"{os.path.splitext(filename)[0]}_cleaned.txt"
# # Или изученным способом
# old_name = filename.rsplit('.', 1)
# new_filename = f"{old_name[0]}_cleaned.{old_name[1]}"

try:
    with (open(filename, "r", encoding="utf-8") as infile,
          open(new_filename, "w", encoding="utf-8") as outfile):
        for line in infile:
            if line.strip():
                outfile.writelines(line)
    print(f"Пустые строки удалены, сохранено в {new_filename}.")
except FileNotFoundError:

    print(f"Ошибка: файл '{filename}' не найден.")
```