Національний університет біоресурсів і природокористування України

Читання, запис і аналіз текстової інформації засобами Python

Лектор: к.ф.-м.н., доцент Кириченко Віктор Вікторович

Поняття файлу

Що таке файл?

- Файл — іменована область на носії інформації
- Містить дані певного формату
- Має унікальне ім'я та розширення
- Зберігає інформацію між запусками програми

Важливість файлів у програмуванні

- Зберігання даних для подальшого використання
- Обмін даними між програмами
- Конфігурація програмних систем
- Обробка великих обсягів інформації

Ключова ідея

Робота з файлами дозволяє програмам зберігати та отримувати дані поза оперативною пам'яттю, що є основою для створення стійких до перезавантаження додатків.

Типи файлів: текстові та бінарні

Текстові файли

- Містять символи, що читаються людиною
- Кодуються за допомогою таблиць символів (ASCII, Unicode)
- Розширення: .txt, .csv, .json, .xml, .html
- Легко редагуються та аналізуються

Бінарні файли

- Містять дані у вигляді послідовності бітів
- Не читаються людиною без спеціальних програм
- Розширення: .jpg, .png, .mp3, .exe, .pdf
- Ефективне зберігання складних даних

Ключові відмінності

Текстові файли:

- Портативність між системами
- Простота аналізу

Бінарні файли:

- Компактність зберігання
- Швидкість обробки

Читання даних з файлу

Основні операції

- Відкриття файлу: `open()`
- Читання даних: `read()`, `readline()`, `readlines()`
- Закриття файлу: `close()`
- Важливість закриття файлу для звільнення ресурсів

Приклад читання файлу

```
file = open("example.txt", "r")
# Читання всього файлу
content = file.read()
file.close()

# Читання по рядках
file = open("example.txt", "r")
lines = file.readlines()
file.close()
```

Важливий аспект

Завжди закривайте файли після використання! Невикористані відкриті файли можуть призвести до втрати даних та перевантаження системи.

Контекстні менеджери для роботи з файлами

Переваги контекстних менеджерів

- Автоматичне закриття файлу
- Безпечна обробка виняткових ситуацій
- Читабельний та лаконічний код
- Запобігання витоку ресурсів

Синтаксис та приклад

```python
# Стандартний підхід
file = open("example.txt", "r")
try:
    content = file.read()
finally:
    file.close()

# З контекстним менеджером
with open("example.txt", "r") as file:
    content = file.read()
    # Файл автоматично закриється
```

Ключова ідея

Контекстний менеджер `with` гарантує, що файл буде закрито навіть у разі виникнення помилки під час виконання коду.

Запис даних у файл

Режими запису

- `'w'` - запис (перезапис)
- Створює новий файл або очищує існуючий
- `'a'` - додавання
- Додає дані в кінець існуючого файлу

Приклади запису

```
# Перезапис файлу
with open("data.txt", "w") as file:
    file.write("Нові дані\n")

# Додавання до файлу
with open("data.txt", "a") as file:
    file.write("Додаткові дані\n")

# Запис списку рядків
lines = ["Рядок 1\n", "Рядок 2\n"]
with open("data.txt", "w") as file:
    file.writelines(lines)
```

Важливий аспект

Будьте обережні з режимом `'w'` - він повністю перезаписує файл, стираючи всі попередні дані!

Режими відкриття файлів

Текстові режими

- `'r'` - читання (за замовчуванням)
- `'w'` - запис (очищує файл)
- `'a'` - додавання в кінець файлу
- `'r+'` - читання та запис

Бінарні режими

- `'rb'` - читання бінарних даних
- `'wb'` - запис бінарних даних
- `'ab'` - додавання бінарних даних
- `'r+b'` - читання та запис бінарних даних

Відмінності та призначення

Текстові режими:

- Для роботи з текстовими файлами
- Автоматичне перетворення символів нового рядка
- Кодування/декодування символів

Бінарні режими:

- Для роботи з бінарними файлами
- Без обробки символів нового рядка
- Пряма робота з байтами

Метод split() для роботи з рядками

Опис методу split()

- Розділяє рядок на підрядки
- Повертає список рядків
- Синтаксис: `str.split(sep=None, maxsplit=-1)`
- `sep` - роздільник (за замовчуванням пробіл)
- `maxsplit` - максимальна кількість розділень

Приклади використання

```
# Базове використання
text = "Python для аналізу даних"
words = text.split()
# Результат: ['Python', 'для', 'аналізу', 'даних']

# З роздільником
data = "ім'я,прізвище,вік"
fields = data.split(",")
# Результат: ["ім'я", "прізвище", "вік"]

# З обмеженням кількості розділень
path = "/home/user/documents/file.txt"
parts = path.split("/", 2)
# Результат: ['', 'home', 'user/documents/file.txt']
```

Практичне застосування

Метод `split()` особливо корисний при обробці CSV-файлів, логів, парсингу текстових даних та розділенні рядків на складові частини для подальшого аналізу.

Метод join() для роботи з рядками

Опис методу join()

- Об'єднує елементи послідовності в один рядок
- Синтаксис: `str.join(iterable)`
- `str` - роздільник між елементами
- `iterable` - список, кортеж тощо
- Зворотна операція до `split()`

Приклади використання

```
# Об'єднання з пробілом
words = ["Python", "для", "аналізу", "даних"]
sentence = " ".join(words)
# Результат: "Python для аналізу даних"

# Об'єднання з комою
data = ["ім'я", "прізвище", "вік"]
csv_line = ",".join(data)
# Результат: "ім'я,прізвище,вік"

# Об'єднання з переносом рядка
lines = ["Рядок 1", "Рядок 2", "Рядок 3"]
text = "\n".join(lines)
# Результат: "Рядок 1\nРядок 2\nРядок 3"
```

Практичне застосування

Метод `join()` незамінний при створенні CSV-файлів, форматуванні виводу, генерації SQL-запитів та об'єднанні даних для запису у текстові файли.

Методи strip(), lstrip(), rstrip() для роботи з рядками

Опис методів

- `strip()` - видаляє символи з обох кінців
- `lstrip()` - видаляє символи з лівого кінця
- `rstrip()` - видаляє символи з правого кінця
- Синтаксис: `str.strip([chars])`
- `chars` - набір символів для видалення

Приклади використання

```
# Видалення пробілів
text = " Приклад тексту "
cleaned = text.strip()
# Результат: "Приклад тексту"

# Видалення специфічних символів
data = "**Важливі дані**"
clean_data = data.strip("*")
# Результат: "Важливі дані"

# Видалення з різних кінців
path = "//home/user//"
left_clean = path.lstrip("/")
right_clean = path.rstrip("/")
# Результати: "home/user//" та "//home/user"
```

Практичне застосування

Методи `strip()`, `lstrip()` та `rstrip()` корисні при очищенні даних, введених користувачем, обробці файлів з невідомим форматуванням та підготовці рядків для подальшої обробки.

Метод replace() для роботи з рядками

Опис методу replace()

- Замінює підрядок в рядку
- Синтаксис: `str.replace(old, new, count=-1)`
- `old` - підрядок для заміни
- `new` - новий підрядок
- `count` - максимальна кількість замін

Приклади використання

```
# Проста заміна
text = "Python - чудова мова"
new_text = text.replace("чудова", "потужна")
# Результат: "Python - потужна мова"

# Обмежена кількість замін
data = "1, 2, 3, 4, 5"
modified = data.replace(",", ";", 2)
# Результат: "1; 2; 3, 4, 5"

# Видалення символів
messy = "Текст***з***зайвими***символами"
clean = messy.replace("***", " ")
# Результат: "Текст з зайвими символами"
```

Практичне застосування

Метод `replace()` незамінний при очищенні текстових даних, форматуванні виводу, виправленні помилок та перетворенні даних між різними форматами.

Форматування рядків у Python

% Оператор %

- Старий стиль форматування
- Схожий з функцією printf

```
name = "Олексій"
age = 25
result = "Ім'я: %s, Вік: %d" % (name, age)
# Результат: "Ім'я: Олексій, Вік: 25"
```

Метод format()

- Гнучкіший підхід
- Підтримка іменованих аргументів

```
name = "Олексій"
age = 25
result = "Ім'я: {}, Вік: {}".format(name, age)
# Результат: "Ім'я: Олексій, Вік: 25"
```

f-рядки (Python 3.6+)

- Найсучасніший підхід
- Найшвидший та найчитабельніший

```
name = "Олексій"
age = 25
result = f"Ім'я: {name}, Вік: {age}"
# Результат: "Ім'я: Олексій, Вік: 25"
```

Практичне застосування

Форматування рядків використовується для створення звітів, генерації текстових файлів, виводу даних користувачу та формування запитів до баз даних. F-рядки є рекомендованим підходом для сучасного коду на Python.

Практичне завдання: читання текстового файлу

Завдання

- Прочитати вміст текстового файлу
- Обробити дані: розділити на рядки
- Видалити зайві пробіли та порожні рядки
- Підготувати дані для подальшого аналізу

Приклад реалізації

```python
# Читання файлу та базова обробка
def read_text_file(file_path):
    try:
        with open(file_path, 'r', encoding='utf-8') as file:
            lines = file.readlines()
            # Обробка рядків
            clean_lines = []
            for line in lines:
                stripped = line.strip()
                if stripped:
                    clean_lines.append(stripped)
            return clean_lines
        except FileNotFoundError:
            return ["Файл не знайдено"]
```

Поради для ефективного читання файлів

Використання кодування

Завжди вказуйте кодування (utf-8) для коректної роботи з українськими символами

Обробка винятків

Використовуйте try-except для обробки помилок (файл не існує, проблеми з доступом)

Оптимізація пам'яті

Для великих файлів використовуйте по рядкове читання замість readlines()

Практичне завдання: обробка даних

{} Завдання обробки

- Розділити рядки на окремі поля
- Вилучити та стандартизувати дані
- Застосувати фільтрацію за критеріями
- Підготувати дані для аналізу

<> Приклад обробки даних

```
# Обробка даних з файлу
def process_data(clean_lines):
    processed_data = []
    for line in clean_lines:
        # Розділення рядка на поля
        fields = line.split(",")
        if len(fields) >= 3:
            # Очищення та форматування полів
            name = fields[0].strip().title()
            age = fields[1].strip()
            city = fields[2].strip().replace("м.", "")
            # Фільтрація за віком
            if age.isdigit() and int(age) >= 18:
                processed_data.append({
                    "name": name,
                    "age": int(age),
                    "city": city
                })
    return processed_data
```

Ефективні методи обробки даних

Використання list comprehensions
Для спрощення коду та підвищення продуктивності

Перевірка типів даних
Завжди перевіряйте типи даних перед обробкою

Структурування даних
Використовуйте словники для зберігання структурованих даних

Практичне завдання: збереження результатів

Завдання збереження

- Записати оброблені дані у новий файл
- Вибрати відповідний формат файлу
- Форматувати дані для запису
- Перевірити успішність операції

Приклад збереження результатів

```
# Збереження результатів у файл
def save_results(processed_data, output_file):
    try:
        with open(output_file, 'w', encoding='utf-8') as file:
            # Запис заголовка
            file.write("Ім'я,Вік,Місто\n")
            # Запис даних
            for item in processed_data:
                line = f"{item['name']},{item['age']},
{item['city']}\n"
                file.write(line)
            # Альтернативний варіант з використанням join
            lines = ["Ім'я,Вік,Місто"]
            for item in processed_data:
                lines.append(f"{item['name']},{item['age']},
{item['city']}")
            content = "\n".join(lines)
            file.write(content)
        return True
    except Exception as e:
        print(f"Помилка збереження: {e}")
        return False
```

Поради для ефективного збереження даних

Вибір формату
Використовуйте CSV для табличних даних, JSON для структурованих даних

Обробка помилок
Перевіряйте права доступу до файлу та наявність місця на диску

Оптимізація
Для великих обсягів даних використовуйте буферизацію або пакетний запис

Висновки

Основні висновки

- Python надає потужні інструменти для роботи з текстовими файлами
- Контекстні менеджери забезпечують безпечну роботу з файлами
- Різні режими відкриття файлів дозволяють гнучко працювати з даними
- Рядкові методи Python є основою для обробки текстової інформації

Практичне значення

- Вміння працювати з файлами є необхідним для будь-якого програміста
- Обробка текстової інформації є основою для аналізу даних
- Комбінування методів роботи з рядками дозволяє вирішувати складні завдання
- Практичні навички роботи з файлами формують основу для подальшого вивчення Python

Ключові ідеї

- Ефективна робота з файлами вимагає розуміння їх типів та режимів доступу
- Методи обробки рядків є інструментами для трансформації даних
- Комплексний підхід до читання, обробки та запису даних забезпечує повний цикл роботи з інформацією