Національний університет біоресурсів і природокористування України

Серіалізація та робота з табличними форматами даних у Python

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

Вступ

Серіалізація та робота з табличними форматами даних є ключовими навичками для ефективної обробки та збереження інформації у Python

Основні теми лекції:

- Серіалізація та десеріалізація об'єктів у Python
- Робота з форматом JSON
- Представлення табличних даних у форматі CSV
- Читання та запис CSV-файлів за допомогою словників
- Створення власного діалекту CSV
- Використання сніффера для розпізнавання діалекту CSV
- Робота з форматом XLSX

Практичні приклади та інструменти для роботи з даними

Серіалізація та десеріалізація в Python

Серіалізація

Процес перетворення об'єкта Python у потік байтів для збереження або передачі

- Збереження стану об'єкта
- Передача даних між системами
- Кешування об'єктів

Десеріалізація

Процес відновлення об'єкта Python з потоку байтів

- Відтворення об'єкта зі збереженого стану
- Завантаження даних з файлів або мережі
- Відновлення структури даних

Приклад серіалізації за допомогою pickle

```
import pickle

# Створення об'єкта
data = {'name': 'John', 'age': 30, 'scores': [85, 92, 78]}

# Серіалізація
with open('data.pkl', 'wb') as f:
    pickle.dump(data, f)

# Десеріалізація
with open('data.pkl', 'rb') as f:
    loaded_data = pickle.load(f)
    print(loaded_data)
```

⚠ **Увага:** pickle не є безпечним для обробки даних з ненадійних джерел

Робота з форматом JSON

Що таке JSON?

- JavaScript Object Notation
- Текстовий формат обміну даними
- Людочитабельний та машиночитабельний
- Підтримує базові типи даних: рядки, числа, булеві значення, масиви, об'єкти

Модуль json в Python

- Стандартний модуль для роботи з JSON
- Перетворення Python-об'єктів у JSON та навпаки
- Основні методи: `dump()`, `dumps()`, `load()`, `loads()`
- Підтримка роботи з файлами та рядками

Відповідність типів даних Python та JSON

Python → JSON

- `dict → object`
- `list, tuple → array`
- `str → string`

JSON → Python

- `object → dict`
- `array → list`
- `string → str`

JSON є стандартом для веб-сервісів та API

Приклади роботи з JSON

Серіалізація в рядок

```python
import json

# Створення словника
data = {
  "name": "Анна",
  "age": 25,
  "courses": ["Python", "Data Science"]
}

# Перетворення в JSON-рядок
json_string = json.dumps(data, indent=4, ensure_ascii=False)
print(json_string)
```

Запис у файл

```python
import json

# Дані для запису
data = {
  "students": [
    {"name": "Іван", "grade": 90},
    {"name": "Олена", "grade": 85}
  ]
}

# Запис у файл
with open('students.json', 'w', encoding='utf-8') as f:
    json.dump(data, f, indent=4, ensure_ascii=False)
```

Читання з файлу

```python
import json

# Читання даних з файлу
with open('students.json', 'r', encoding='utf-8') as f:
    data = json.load(f)

# Робота з даними
for student in data['students']:
    print(f"{student['name']}: {student['grade']}")
```

Парсинг з рядка

```python
import json

# JSON-рядок
json_string = '{"name": "Петро", "age": 30, "city": "Київ"}'

# Перетворення в Python-об'єкт
data = json.loads(json_string)

# Доступ до даних
print(data['name']) # Петро
print(data['age']) # 30
```

Параметр `ensure_ascii=False` дозволяє коректно відображати кириличні символи

Формат CSV: основи

Що таке CSV?

- Comma-Separated Values (значення, розділені комами)
- Текстовий формат для представлення табличних даних
- Кожен рядок відповідає одному запису (рядку таблиці)
- Значення в рядку розділяються роздільником (зазвичай кома)

Модуль csv в Python

- Стандартний модуль для роботи з CSV-файлами
- Основні компоненти: читачі, записувачі, діалекти
- Підтримка різних форматів та роздільників
- Можливість роботи зі словниками для зручності

Приклад структури CSV-файлу

```
# Приклад файлу students.csv
Ім'я,Прізвище,Вік,Спеціальність
Іван,Петренко,20,Інформатика
Олена,Коваль,21,Біологія
Петро,Сидоренко,22,Екологія
```

Переваги

Простота формату
Підтримка в Excel
Людочитабельність

Обмеження

Відсутність типізації даних
Проблеми з кодуванням
Складність з вкладеними структурами

Застосування

Експорт даних з баз
Обмін даними між системами
Аналіз даних

CSV — універсальний формат для обміну табличними даними

Читання CSV-файлів

Основні методи

- `csv.reader()` - читання файлу як список рядків
- `csv.DictReader()` - читання файлу як словників
- Використання контекстного менеджера `with`
- Вказання кодування файлу (`encoding`)

Параметри читання

- `delimiter` - роздільник полів (за замовчуванням ',')
- `quotechar` - символ для цитування (за замовчуванням '"')
- `skipinitialspace` - пропуск пробілів після роздільника
- `encoding` - кодування файлу (напр. 'utf-8')

Приклад читання CSV-файлу

```python
import csv

# Читання файлу за допомогою reader
with open('students.csv', 'r', encoding='utf-8') as file:
    csv_reader = csv.reader(file)
    for row in csv_reader:
        print(row)

# Читання файлу за допомогою DictReader
with open('students.csv', 'r', encoding='utf-8') as file:
    dict_reader = csv.DictReader(file)
    for row in dict_reader:
        print(row['Ім\'я'], row['Прізвище'])
```

DictReader автоматично використовує перший рядок як ключі словника

Запис CSV-файлів

Основні методи

- `csv.writer()` - запис списків у файл
- `csv.DictWriter()` - запис словників у файл
- Методи `writerow()` та `writerows()`
- Використання контекстного менеджера `with`

Параметри запису

- `delimiter` - роздільник полів (за замовчуванням ',')
- `quotechar` - символ для цитування (за замовчуванням '"')
- `quoting` - коли цитувати значення
- `lineterminator` - символ завершення рядка

Приклад запису CSV-файлу

```
import csv

# Дані для запису
data = [
  ['Іван', 'Петренко', 20, 'Інформатика'],
  ['Олена', 'Коваль', 21, 'Біологія']
]

# Запис файлу за допомогою writer
with open('students.csv', 'w', encoding='utf-8', newline='') as file:
    writer = csv.writer(file)
    writer.writerow(['Ім\'я', 'Прізвище', 'Вік', 'Спеціальність'])
    writer.writerows(data)
```

Запис за допомогою DictWriter

```
# Дані для запису у вигляді словників
data = [
  {'Ім\'я': 'Іван', 'Прізвище': 'Петренко', 'Вік': 20, 'Спеціальність': 'Інформатика'},
  {'Ім\'я': 'Олена', 'Прізвище': 'Коваль', 'Вік': 21, 'Спеціальність': 'Біологія'}
]

# Запис файлу за допомогою DictWriter
with open('students_dict.csv', 'w', encoding='utf-8', newline='') as file:
    fieldnames = ['Ім\'я', 'Прізвище', 'Вік', 'Спеціальність']
    writer = csv.DictWriter(file, fieldnames=fieldnames)
    writer.writeheader()
    writer.writerows(data)
```

Робота з CSV за допомогою словників

Переваги використання словників

- Зручний доступ до даних за назвами полів
- Код стає більш читабельним та зрозумілим
- Можливість легко змінювати порядок стовпців
- Легше працювати з неповними даними

Основні класи

- `csv.DictReader` - для читання CSV у словники
- `csv.DictWriter` - для запису словників у CSV
- Метод `writeheader()` для запису заголовків
- Параметр `fieldnames` для визначення полів

Приклад: читання CSV за допомогою DictReader

```
import csv

# Читання файлу
with open('students.csv', 'r', encoding='utf-8') as file:
    reader = csv.DictReader(file)
    for row in reader:
        # Доступ до даних за назвою поля
        name = row['Ім\'я']
        surname = row['Прізвище']
        age = row['Вік']
        specialty = row['Спеціальність']
        print(f"{name} {surname}, {age} років, {specialty}")
```

Приклад: запис CSV за допомогою DictWriter

```
import csv

# Дані для запису
students = [
  {'Ім\'я': 'Петро', 'Прізвище': 'Сидоренко', 'Вік': 22, 'Спеціальність': 'Екологія'},
  {'Ім\'я': 'Марія', 'Прізвище': 'Мельник', 'Вік': 19, 'Спеціальність': 'Агрономія'}
]

# Визначення полів
fieldnames = ['Ім\'я', 'Прізвище', 'Вік', 'Спеціальність']

# Запис у файл
with open('new_students.csv', 'w', encoding='utf-8', newline='') as file:
    writer = csv.DictWriter(file, fieldnames=fieldnames)
    writer.writeheader() # Запис заголовків
```

Створення власного діалекту CSV

Що таке діалект CSV?

- Набір параметрів для визначення формату CSV
- Дозволяє працювати з різними варіаціями формату
- Уніфікує параметри для читання та запису
- Покращує читабельність коду

Основні параметри діалекту

- `delimiter` - роздільник полів
- `quotechar` - символ цитування
- `escapechar` - символ екранування
- `doublequote` - подвоєння символу цитування
- `skipinitialspace` - пропуск початкових пробілів

Створення власного діалекту

```
import csv

# Створення класу для власного діалекту
class MyCSVDialect(csv.Dialect):
    delimiter = ';'
    quotechar = '"'
    escapechar = '\\'
    doublequote = False
    skipinitialspace = True
    quoting = csv.QUOTE_MINIMAL
    lineterminator = '\r\n'

# Реєстрація діалекту
csv.register_dialect('mycsv', MyCSVDialect)

# Альтернативний спосіб створення діалекту
csv.register_dialect('excel_eu', delimiter=';', quotechar='"')
```

Використання власного діалекту

```
# Запис даних з використанням діалекту
data = [
  ['Ім\'я', 'Прізвище', 'Вік'],
  ['Іван', 'Петренко', 20],
  ['Олена', 'Коваль', 21]
]

with open('data.csv', 'w', encoding='utf-8', newline='') as file:
    writer = csv.writer(file, dialect='mycsv')
```

Використання сніффера для розпізнавання діалекту CSV

Що таке CSV Sniffer?

- Клас `csv.Sniffer` для аналізу формату CSV
- Автоматичне визначення параметрів діалекту
- Аналізує роздільники, символи цитування тощо
- Корисний для роботи з невідомими CSV-файлами

Основні методи

- `sniff()` - аналізує CSV і повертає діалект
- `has_header()` - визначає наявність заголовка
- Параметр `sample` - кількість символів для аналізу
- Параметр `delimiters` - можливі роздільники

Приклад використання сніффера

```
import csv

# Відкриваємо файл для аналізу
with open('unknown_format.csv', 'r', encoding='utf-8') as file:
    # Читаємо частину файлу для аналізу
    sample = file.read(1024)
    file.seek(0)

    # Визначаємо діалект
    dialect = csv.Sniffer().sniff(sample)

    # Перевіряємо наявність заголовка
    has_header = csv.Sniffer().has_header(sample)

    # Виводимо інформацію про діалект
    print(f"Роздільник: {dialect.delimiter}")
    print(f"Символ цитування: {dialect.quotechar}")
    print(f"Наявність заголовка: {has_header}")

    # Читаємо файл з визначеним діалектом
    reader = csv.reader(file, dialect=dialect)
    for row in reader:
        print(row)
```

Приклад з вказанням можливих роздільників

```
import csv

with open('data.csv', 'r', encoding='utf-8') as file:
    sample = file.read(1024)
```

Формат XLSX: основи

Що таке XLSX?

- Формат файлів Microsoft Excel (з 2007 року)
- Фактично є ZIP-архівом з XML-документами
- Підтримує багато аркушів, форматування, формули
- Стандарт для бізнес-даних та звітності

Python-бібліотеки для XLSX

- `openpyxl` - читання, запис, форматування
- `xlsxwriter` - створення файлів з нуля
- `pandas` - високорівнева робота з таблицями
- `xlrd/xlwt` - для старих форматів (.xls)

Структура XLSX-файлу

Компоненти файлу

- Робочі книги (Workbook)
- Робочі аркуші (Worksheet)
- Комірки (Cell)
- Рядки та стовпці

Типи даних

- Текст, числа, дати
- Формули
- Форматування
- Графіки та зображення

Переваги

Підтримка форматування

Багато аркушів у файлі

Формули та обчислення

Обмеження

Складніший за CSV

Більший розмір файлу

Потрібні спеціальні бібліотеки

Застосування

Бізнес-звіти

Фінансові розрахунки

Наукові дані

XLSX — стандарт для обміну складними табличними даними

Читання XLSX-файлів

Основні бібліотеки

- `openpyxl` - найпопулярніша для XLSX
- `pandas` - зручна для аналізу даних
- `xlrd` - для старих форматів .xls
- `pyxlsb` - для бінарних форматів

Основні операції

- Відкриття файлу: `load_workbook()`
- Доступ до аркушів: `active`, `sheetnames`
- Читання комірок: `cell()`, `iter_rows()`
- Значення комірок: `.value`

Приклад читання XLSX за допомогою openpyxl

```
from openpyxl import load_workbook

# Завантаження книги
workbook = load_workbook('data.xlsx')

# Отримання активного аркуша
sheet = workbook.active

# Читання значення конкретної комірки
cell_value = sheet['A1'].value
print(f"Значення комірки A1: {cell_value}")

# Читання діапазону комірок
for row in sheet.iter_rows(min_row=1, max_row=5, min_col=1, max_col=3):
    for cell in row:
        print(cell.value, end='\t')
    print()
```

Приклад читання XLSX за допомогою pandas

```
import pandas as pd

# Читання файлу в DataFrame
df = pd.read_excel('data.xlsx', sheet_name='Sheet1')

# Виведення перших 5 рядків
print(df.head())

# Доступ до стовпця
print(df['Назва стовпця'])
```

Запис XLSX-файлів

Основні бібліотеки

- `openpyxl` - для створення та модифікації
- `xlsxwriter` - оптимізована для запису
- `pandas` - для експорту DataFrame
- `xlwt` - для старих форматів .xls

Основні операції

- Створення книги: `Workbook()`
- Створення аркуша: `create_sheet()`
- Запис комірок: `cell()`, `.value`
- Збереження файлу: `save()`

Приклад запису XLSX за допомогою openpyxl

```
from openpyxl import Workbook

# Створення нової книги
workbook = Workbook()

# Отримання активного аркуша
sheet = workbook.active
sheet.title = "Студенти"

# Запис заголовків
headers = ["Ім'я", "Прізвище", "Вік", "Спеціальність"]
for col_num, header in enumerate(headers, 1):
    sheet.cell(row=1, column=col_num, value=header)

# Запис даних
data = [
  ["Іван", "Петренко", 20, "Інформатика"],
  ["Олена", "Коваль", 21, "Біологія"]
]

for row_num, row_data in enumerate(data, 2):
    for col_num, cell_value in enumerate(row_data, 1):
        sheet.cell(row=row_num, column=col_num, value=cell_value)

# Збереження файлу
workbook.save("students.xlsx")
```

Приклад запису XLSX за допомогою pandas

```
import pandas as pd
```

Висновки

Серіалізація

- Серіалізація — ключовий механізм збереження стану об'єктів
- Python пропонує вбудовані модулі для серіалізації
- Вибір формату залежить від типу даних та завдань

Формати даних

- JSON — ідеальний для структурованих даних та API
- CSV — простий формат для табличних даних
- XLSX — потужний формат для складних таблиць

Робота з CSV

- Модуль csv пропонує гнучкі інструменти для роботи
- DictReader та DictWriter спрощують доступ до даних
- Діалекти та сніффер дозволяють працювати з різними варіаціями формату

Робота з XLSX

- Бібліотеки openpyxl та pandas забезпечують повний контроль
- Підтримка форматування, формул та багатьох аркушів
- Ідеально підходить для бізнес-застосувань

Загальні висновки

- Вибір формату даних залежить від конкретних завдань та вимог
- Python надає потужні та гнучкі інструменти для роботи з різними форматами
- Розуміння серіалізації та роботи з табличними даними є необхідним для сучасного розробника