СЕРІАЛІЗАЦІЯ ОБ’ЄКТІВ У МОВІ PYTHON. КОНСЕРВУВАННЯ ОБ’ЄКТІВ

Питання 10.3

Серіалізація та десеріалізація

- Для персистентного зберігання Python-об’єктів необхідно конвертувати його в байти та записати їх у файл.
  - Цю операцію називають ***серіалізацією***, або ***маршалінгом*** (marshaling, deflating, encoding).
- Кожну зі схем серіалізації також можна назвати фізичним форматом даних.
  - Слід відрізняти його від логічного формату даних – простого впорядкування за допомогою пробілів, яке змінює послідовність байтів, проте не значення об’єкту.
- За винятком CSV, такі представлення націлені на збереження одного Python-об’єкта.
  - Хоч один об’єкт може бути списком об’єктів, він буде конкретного розміру.
  - Для обробки одного з об’єктів потрібно ***десеріалізувати*** весь список.

Серіалізаційні представлення

- ***JavaScript Object Notation (JSON)***: поширене представлення. Стандартний модуль json постачає класи та функції, необхідні для завантаження та збереження (dump) даних тільки у JSON-форматі.
- ***Yet Ain't Markup Language (YAML)***: розширення JSON, яке дещо спрощує серіалізований вивід. Спеціальний пакет PyYaml передбачає багато можливостей для підтримки персистентності Python.
- ***pickle:*** модуль pickle має власне, специфічне для Python, представлення даних. Недоліком формату є слабкі можливості для обміну даними з програмами, написаними не мовою Python. Цей формат є базовим для модуля shelve та черг повідомлень (message queues).
- ***Comma-Separated Values (CSV):*** поширений формат, проте може бути незручним для представлення складних Python-об'єктів. CSV дозволяє виконувати інкрементоване представлення колекцій Python-об'єктів, що не поміщаються в пам'ять.
- ***XML:*** дуже поширений формат, тому його парсинг має велике значення. Модулів для цієї задачі багато, часто користуються ElementTree.

Python-об’єкти можуть жити, поки працює відповідний процес

- Поки на них є посилання в просторі імен.
  - Якщо бажаємо мати об’єкт, що існуватиме за межами процесу чи простору імен, необхідно зробити його ***персистентним***.
  - Більшість ОС пропонують підтримку персистентності в формі файлової системи.
- Складність: об’єкти можуть посилатись на інші об’єкти.
  - Об’єкт посилається на свій клас, клас посилається на свій метаклас чи базовий клас.
  - Об’єкт може бути контейнером та посилатись на інші об’єкти.
  - Оскільки розташування в пам’яті не фіксовані, зв’язки між об’єктами порушуються після спроби збереження та відновлення байтів з пам’яті без переписування адрес у певного виду незалежний від розташування (location-independent) ключ.

Проблема схеми міграції (Schema Migration Problem)

- Багато об’єктів у павутині посилань у значній мірі статичні.
  - Наприклад, опис класів змінюється повільніше за значення змінних, а в ідеалі, не змінюється зовсім.
  - Проте на рівні класу можуть бути присутні instance variables.
  - Управління зміною структури (класу) даних називають Schema Migration Problem.
- Python надає формальну відмінність між змінними об’єкта та іншими атрибутами класу.
  - Визначаємо object's instance variables, щоб коректно відобразити динамічний стан об’єкта.
  - Атрибути рівня класу використовують для інформації, що буде спільною для об’єктів цього класу.
  - Якщо можна зберегти на постійній основі лише динамічний стан об’єкта—відокремлений від класу та павутини посилань з опису класу—це буде робоче рішення для серіалізації та персистентного (постійного) зберігання.

Модуль pickle реалізує потужний алгоритм серіалізації / десеріалізації об’єктів Python

- «Pickling» (консервування) – процес перетворення об’єкта Python у потік байтів, а «unpickling» (розконсервація) – зворотна операція.

```
>>> import pickle
>>> data = {
...     'a': [1, 2.0, 3, 4+6j],
...     'b': ("character string", b"byte string"),
...     'c': {None, True, False}
... }
>>>
>>> with open('data.pickle', 'wb') as f:
...     pickle.dump(data, f)
...
>>> with open('data.pickle', 'rb') as f:
...     data_new = pickle.load(f)
...
>>> print(data_new)
{'c': {False, True, None}, 'a': [1, 2.0, 3, (4+6j)], 'b': ('character string', b'byte string')}
```

Поширена термінологія для Python (dump і load)

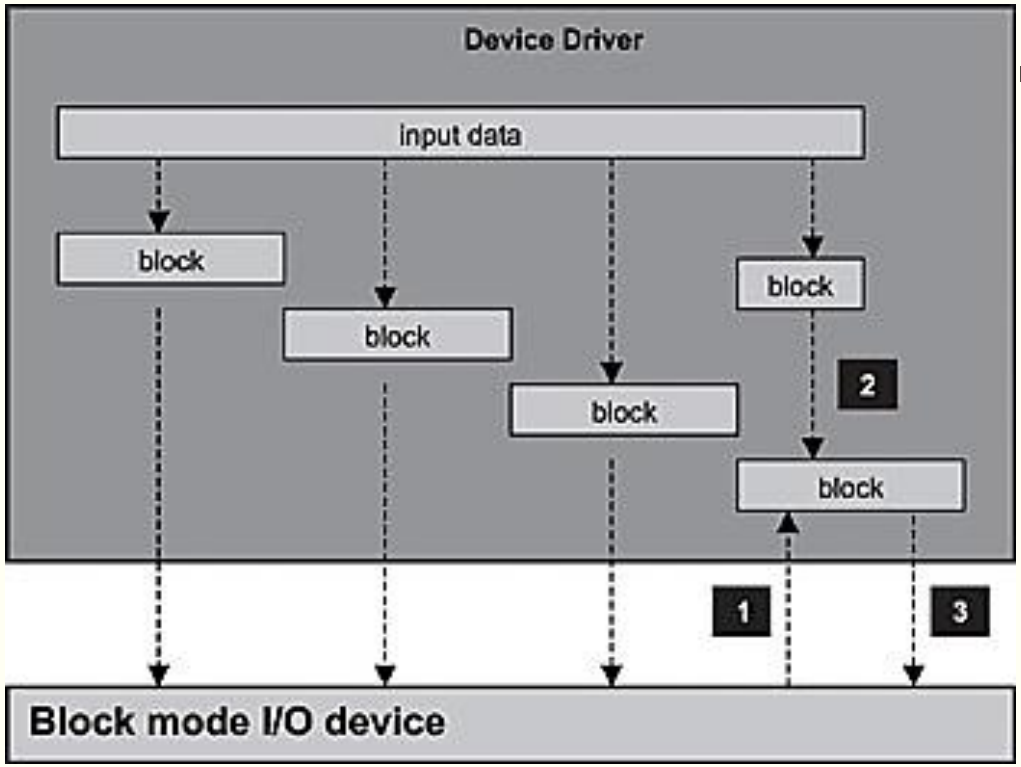
- Більшість класів у даній темі визначають наступні методи:
  - dump(object, file): запис (dump) об’єкту *object* у заданий файл
  - dumps(object): записує об’єкт, при цьому повертаючи рядкове представлення
  - load(file): завантажує об’єкт із вказаного файлу, повертає сконструйований об’єкт
  - loads(string): завантажує об’єкт з рядкового представлення, повертає сконструйований об’єкт.
- Загалом файл, що використовувався для dump/load, може бути будь-яким файлоподібним об’єктом.
  - Можна використовувати об’єкти io.StringIO, як і об’єкти urllib.request у якості джерел даних для завантаження.
  - Аналогічно, dump накладає кілька вимог на джерело даних.

Файлова система та мережа

- Оскільки файлова система ОС (і мережа) працюють з байтами, необхідно представляти значення полів об'єкта у якості серіалізованих потоків байтів.
  - Часто застосовують двоетапне перетворення:
  - Стан об'єкта представляється у вигляді рядка.
  - Засоби Python конвертують рядки в байти.
- У контексті файлових систем виділяють два класи пристроїв:
  - ***Блочні пристрої (Block-mode devices)*** також називають ***seekable-пристроями***, оскільки ОС підтримує операції пошуку (seek), які можуть отримати доступ до будь-якого байту файлу.
  - ***Символьні пристрої (Character-mode devices)*** не є seekable; це інтерфейси, в які байти послідовно передаються. Пошук може бути зворотним.
- Відмінності між символьним та блочним режимами можуть вплинути на представлення стану складного об'єкта або колекції об'єктів.
  - Далі розглянемо підходи до серіалізації найпростішого набору даних – впорядкованого потоку (stream) байтів. Вони збережуть байтовий потік або в символьний, або блочний файл.

Символьні пристрої дозволяють передачу неструктурованих даних

- Зазвичай передача відбувається послідовно, по байту.
  - Це прості пристрої, на зразок послідовного інтерфейсу чи клавіатури: драйвер буферизує дані у випадках, коли швидкість передачі від системи до пристрою вища, ніж пристрій може обробити.



- Блочні пристрої передають дані блоками, наприклад, по 1Кб.
  - Розмір блоку визначається апаратним забезпеченням.
  - Дані можуть бути або в деякій мірі структуровані, або передаватись по деякому протоколу передачі.
  - Інакше ймовірна помилка.
  - Тому інколи необхідно для драйверу блочного пристрою виконувати додаткову роботу для операції зчитування чи запису.

Модуль pickle виконує нативне постійне зберігання об’єктів

- Модуль pickle може перетворити складний об’єкт у байтовий потік, і навпаки.
  - Найбільш очевидна річ, яку можна зробити з байтовими потоками – записати їх у файл, проте також поширена відправка їх по мережі або збереження в базу даних.
  - Це не формат обміну даними, як JSON, YAML, CSV або XML.
- Модуль pickle тісно інтегрований з Python багатьма шляхами.
  - Наприклад, методи __reduce__() та __reduce_ex__() класу існують для підтримки pickle-процесингу.
- Консервування об’єкту travel у файл з дескриптором target:

```
import pickle
with open("travel_blog.p","wb") as target:
    pickle.dump( travel, target )
```

  - Файл записано як сирі (raw) байти, тому функція open() використовує режим "wb".
  - Можна відновити консервований об’єкт так:

```
with open("travel_blog.p","rb") as source:
        copy= pickle.load( source )
```

Проєктування класу для надійного pickle-процесингу

- Метод __init__() класу не використовується to unpickle об'єкта.
  - Метод __init__() обходиться шляхом використання методу __new__() та задання значень для консервування напряму в __dict__ об'єкта.
  - Це важливо, коли оголошення класу включає деяку обробку в __init__(). Наприклад, якщо __init__() відкриває зовнішні файли, створює деякі частини GUI-інтерфейсу або виконує деяке зовнішнє оновлення БД, все це не буде виконуватись протягом розконсервації.
- Якщо обчислювати новий екземпляр протягом обробки __init__(), проблеми немає.
  - Наприклад, об'єкт BlackjackHand, який обчислює суму з екземплярів класу Card при створенні Hand.
  - Звичайна консервація збереже цю обчислену змінну екземпляра (instance variable) та не виконуватиме повторне обчислення при розконсервації.
- Клас з обробкою даних в методі __init__() повинен здійснити налаштування, щоб переконатись в коректності початкового процесингу даних. Можемо зробити таке:
  - Замість негайного startup processing в __init__() виконувати одноразову initialization processing. Наприклад, операції із зовнішнім файлом повинні відкладатись до вимоги.
  - Визначити методи __getstate__() і __setstate__(), які pickle використає, щоб зберігати та відновлювати стан об'єкта. Метод __setstate__() може потім викликати той же метод, що __init__() викликає для одноразової initialization processing у звичайному Python-коді.

Приклад: початкові екземпляри Card, роздані в Hand логуються для аудиту в методі __init__()

- Версія Hand, яка нормально не працює під час розконсервації.

```
class Hand_x:
    def __init__( self, dealer_card, *cards ):
        self.dealer_card= dealer_card
        self.cards= list(cards)
        for c in self.cards:
            audit_log.info( "Initial %s", c )
    def append( self, card ):
        self.cards.append( card )
        audit_log.info( "Hit %s", card )
    def __str__( self ):
        cards= ", ".join( map(str,self.cards) )
        return "{self.dealer_card} | {cards}".format( self=self,
cards=cards )
```

- Присутні 2 місця для логування (logging locations): в методах __init__() та append().
- Обробка в __init__() працює неузгоджено між початковим створеннням об'єкту та розконсервацією для його повторного відтворення.

Виконуємо початкове налаштування для ведення логу

```
import logging,sys
audit_log= logging.getLogger( "audit" )
logging.basicConfig(stream=sys.stderr, level=logging.INFO)
```

- В ході нього створюється лог та перевіряється, щоб рівень ведення логу був доречним для перегляду аудитної інформації.
- Невеликий скрипт, що збирає, консервує та розконсервовує об’єкт типу Hand_x:

```
h = Hand_x( FaceCard('K','♦'), AceCard('A','♣'), Card('9','❤') )
data = pickle.dumps( h )
h2 = pickle.loads( data )
```

- При виконанні скрипту записи з логу, зроблені під час роботи __init__(), не виписано при розконсервації Hand.
- Для того, щоб ретельно записати лог аудиту для розконсервації, можна внести ліниві (lazy) тести ведення логу по всьому класу.
- Наприклад, розширити метод __getattribute__(), щоб записувати початкові записи з логу незалежно від того, чи до довільного атрибуту цього класу були звернення.
- Це призведе до логування зі збереженням стану (stateful logging).

Краще рішення – змінити стандартний для pickle спосіб збереження та відновлення стану

```
class Hand2:
    def __init__( self, dealer_card, *cards ):
        self.dealer_card= dealer_card
        self.cards= list(cards)
        for c in self.cards:
            audit_log.info( "Initial %s", c )
    def append( self, card ):
        self.cards.append( card )
        audit_log.info( "Hit %s", card )
    def __str__( self ):
        cards= ", ".join( map(str,self.cards) )
        return "{self.dealer_card} | {cards}".format( self=self,
cards=cards )
    def __getstate__( self ):
        return self.__dict__
    def __setstate__( self, state ):
        self.__dict__.update(state)
        for c in self.cards:
            audit_log.info( "Initial (unpickle) %s", c )
```

- Метод __getstate__() використовується під час консервування, щоб зібрати дані про поточний стан об'єкта.
  - Цей метод може повернути будь-що.
  - У випадку об'єктів, які мають внутрішню мемоізацію (різновид кешування), кеш може не консервуватись, щоб зберегти час та простір.
  - Ця реалізація використовує внутрішній __dict__ без змін.
- Метод __setstate__() використовується при розконсервації, щоб змінити (reset) значення об'єкту на значення за умовчанням.
  - Ця версія об'єднує стан у внутрішню змінну __dict__, а потім записує доречні записи з логу.

Проблеми при роботі

- Під час розконсервації глобальні імена в pickle-потоці можуть вести до виконання довільного коду.
  - Глобальні імена загалом є назвами класів або функцій.
  - Проте є можливість включити глобальне ім'я, яке буде функцією з модулів (на зразок os або subprocess).
  - Це дозволить атакувати додатки, які спробують передати законсервовані об'єкти по Інтернету без строгого SSL-контролю на місці.
  - Для локальних файлів такої проблеми немає.
- Щоб уникнути виконання довільного коду, необхідно успадковуватись від класу pickle.Unpickler.
  - Замістимо метод find_class(), щоб замінити реалізацію методу чимось безпечнішим.
- Потрібно враховувати декілька інших проблем розконсервації:
  - Необхідно уникати використання вбудованих функцій exec() та eval().
  - Потрібно уникати застосування модулів та пакетів, що можуть не вважатись безпечними. Зокрема, sys чи os.
  - Необхідно дозволити використання модулів нашого додатку.

Приклад з накладанням деяких обмежень

```
import builtins
class RestrictedUnpickler(pickle.Unpickler):
    def find_class(self, module, name):
        if module == "builtins":
            if name not in ("exec", "eval"):
                return getattr(builtins, name)
        elif module == "__main__":
            return globals()[name]
        # elif module in any of our application modules...
        raise pickle.UnpicklingError(
        "global '{module}.{name}' is forbidden".format(module=module,
name=name))
```

- Дана версія класу Unpickler допоможе уникнути великої кількості потенційних проблем.
  - Дозволяє застосування будь-яких вбудованих функцій, за винятком exec() and eval().
  - Дозволяє використовувати класи, визначені тільки в __main__.
  - У решті випадків викидає виняток.

ДЯКУЮ ЗА УВАГУ!

Наступне питання: Робота з серіалізаційним представленням json