Функціональний Python

21 жовтня 2025 р.

Лекція з функціонального програмування

План лекції

Модуль 1: Філософія ФП — Навіщо?

Модуль 2: Структури даних для ФП

Модуль 3: "Двигун"ФП в Python

Модуль 4: Функціональний інструментарій

Модуль 5: Просунуті патерни ФП

Модуль 6: Додаток — Точність

Модуль 1: Філософія ФП — Навіщо?

Що таке Функціональне Програмування (ФП)?

- **Визначення:** Парадигма, що розглядає обчислення як оцінку математичних функцій.
- **Ключова ідея:** Мінімізація **побічних ефектів** та **змінюваного стану (mutable state)**.
- Замість того, щоб *змінювати* дані, ми створюємо *нові* дані шляхом трансформації.

Два стовпи ФП

1. **Чисті функції (Pure Functions)**
   - Завжди повертають однаковий результат для однакових вхідних даних.
   - Не мають побічних ефектів (не змінюють зовнішній стан).
   - **Переваги:** Надійність, легке тестування та паралелізація.

2. **Незмінність даних (Immutability)**
   - Практика створення нових структур даних замість модифікації наявних.
   - Якщо нам потрібні інструменти для ФП, вони мають підтримувати цю філософію.

Модуль 2: Структури даних для ФП

namedtuple: Незмінність і Читабельність

- **Чому він "функціональний":** Це **незмінна (immutable)** структура даних.
- **Перевага:** Доступ до елементів за іменем, а не лише за індексом, що покращує читабельність коду.
- Це чиста структура для "передачі даних"без ризику, що її хтось змінить.

Класичні структури: Стек (LIFO)

- **Принцип LIFO:** "Останнім прийшов — першим вийшов"(Last In, First Out).
- **Аналогія:** стопка тарілок або колода карт.
- Елементи додаються та видаляються лише з одного кінця, який називається "вершиною".

Основні операції

- `push`: додати елемент на вершину.
- `pop`: видалити елемент з вершини.

Зв'язок з ФП
Стек визначає чіткий, передбачуваний **інтерфейс** для послідовної обробки даних.

Класичні структури: Черга (FIFO)

- **Принцип FIFO:** "Першим прийшов — першим вийшов"(First In, First Out).
- **Аналогія:** черга людей в магазині.
- Елементи додаються в один кінець ("хвіст") і видаляються з іншого ("голова").

Основні операції

- `enqueue`: додати елемент у хвіст.
- `dequeue`: видалити елемент з голови.

Зв'язок з ФП

Черга також надає строгий інтерфейс для обробки потоків даних у порядку їх надходження.

deque: Ефективна реалізація в Python

- **deque** (вимовляється "дек") — це двостороння черга (double-ended queue).
- Це високопродуктивна реалізація і для стеків, і для черг у Python.

Чому ‘deque‘ кращий за ‘list‘ для черг?

- Операція `list.pop(0)` є дуже повільною (складність O(n)), оскільки вимагає зсуву всіх наступних елементів.
- Операція `deque.popleft()` є дуже швидкою (складність O(1)).

Висновок
Для побудови ефективних конвеєрів, що обробляють потоки даних, `deque` є основним інструментом.

Управління пам'яттю: Стек (The Call Stack)

- Комп'ютерна пам'ять для виконання програми ділиться на дві основні частини: **Стек** та **Купа**.
- **Стек викликів** — це структура даних LIFO, якою керує сам процесор.
- Він **дуже швидкий**, але **малого, фіксованого розміру**.

Що тут зберігається?

- **Локальні змінні** (числа, булеві значення).
- **Адреси повернення** з функцій.
- **Посилання** на об'єкти, що знаходяться в Купі.

Важливо
Коли ми говоримо про структуру даних "Стек ми імітуємо цю фундаментальну поведінку управління пам'яттю.

Управління пам'яттю: Купа (The Heap)

- **Купа** — це велика, гнучка, але **повільніша** область пам'яті.
- Тут зберігаються всі "великі"об'єкти Python:

Що тут зберігається?

- Списки (`list`)
- Словники (`dict`)
- Двосторонні черги (`deque`)
- Всі об'єкти класів, які ви створюєте.

Управління
У Python Купою керує **Збирач сміття (Garbage Collector)**, який автоматично видаляє об'єкти, на які більше немає посилань (напр., зі Стеку).

Різниця в пам'яті: list vs. deque

'list' — це Динамічний Масив (Array)

- **Модель пам'яті:** Неперервний блок пам'яті, що містить *посилання* на об'єкти в Купі.
- **Проблема** `pop(0)` **(O(n)):** Щоб видалити перший елемент, Python змушений **зсунути всі (n-1) посилання** на одну позицію вліво. Це дуже повільно для великих списків.

'deque' — це Двозв'язний Список Блоків

- **Модель пам'яті:** Двозв'язний список, де кожен вузол — це невеликий масив (блок) з посиланнями.
- **Перевага** `popleft()` **(O(1)):** Щоб видалити перший елемент, Python просто **оновлює один вказівник** ("голову"списку) на наступний елемент. Жодного масового зсуву не відбувається.

deque: Ефективні потоки даних

- Концепції LIFO (Стек) та FIFO (Черга) є чіткими *інтерфейсами* для обробки даних.
- **deque** (Двостороння черга) — це ефективна реалізація в Python.
- На відміну від `list`, `deque` оптимізована для швидких операцій `popleft()` та `appendleft()`.
- Це робить її ідеальним інструментом для потокової обробки даних функціями.

Інструменти для агрегації (Результат ФП-операцій)

defaultdict

- Розв'язання проблеми `KeyError` при групуванні.
- Декларує *намір* (напр., групувати у списки) замість перевірок.
- Дозволяє уникнути складних перевірок `if key not in dict` всередині циклу.

Counter

- Спеціалізований інструмент для підрахунку (кінцевий етап конвеєра).
- Надає корисні методи, як `.most_common()`.

Модуль 3: "Двигун"ФП в Python

Функції як об'єкти першого класу

- **Визначення:** У Python з функціями можна поводитися як зі звичайними об'єктами.
- Це дозволяє:
  - Присвоювати функцію змінній.
  - Зберігати у структурах даних (списки, словники).
  - **Передавати як аргументи** в інші функції.
  - **Повертати як результат** з інших функцій.
- Це є основою для **Функцій Вищого Порядку (HOF)**.

Ефективність: Ліниві обчислення (Генератори)

- **Концепція** `yield`**:** Функція, що "заморожує"свій стан і повертає значення "на вимогу".
- **Перевага:** Величезна економія пам'яті. Ми обробляємо дані по одному елементу, що ідеально для великих потоків даних (напр., читання файлів).
- Генератор — це *ітератор*, який можна використати лише один раз.

Модуль 4: Функціональний інструментарій

Анонімні функції: lambda

- **Синтаксис:** `lambda arguments: expression`.
- **Призначення:** Створення маленьких, однорядкових функцій, які часто передаються як аргументи у функції вищого порядку (HOF).
- Їх не прийнято зберігати у змінні (для цього існує `def`).
- Ідеально для коротких операцій: `sorted(..., key=lambda x: -x)`.

Класичні HOF: map та filter

map(function, iterable)

- **"Трансформація"**. Застосовує `function` до кожного елемента.
- Повертає ітератор (генератор), тобто є "лінивим".

filter(function, iterable)

- **"Вибірка"**. Залишає елементи, для яких `function` повернула `True`.
- Також повертає "лінивий"ітератор.

"Пітонічний"шлях: Comprehensions

- Більш читабельна та часто ефективніша альтернатива `map` та `filter` з `lambda`.
- Вони є "жадібними"(eager) — одразу створюють нову колекцію в пам'яті.

List Comprehension (замість map+filter)
`[expression for item in iterable if condition]`

Set & Dictionary Comprehensions
`{expression for item in iterable if condition}` `{key: value for item in iterable if condition}`

Агрегатори та Перевірки: any / all

any(iterable)

- Повертає `True`, якщо **хоча б один** елемент істинний.
- "Лінива"— зупиняється на першому `True`.
- Корисно для перевірки умов: `any(x < 0 for x in nums)`.

all(iterable)

- Повертає `True`, якщо **всі** елементи істинні.
- "Лінива"— зупиняється на першому `False`.
- (Повертає `True` для порожнього iterable!)

Модуль 5: Просунуті патерни ФП

Замикання (Closures)

- **Визначення:** Внутрішня функція, що "пам'ятає"стан свого оточення (змінні зовнішньої функції) навіть після того, як зовнішня функція завершила роботу.
- Логічний наслідок HOF.
- Дозволяє створювати "фабрики функцій"та функції зі станом.
- Потребує ключового слова `nonlocal` для зміни "запам'ятованих"змінних.

Каррування (Currying)

- **Техніка:** Перетворення функції `f(a, b, c)` на `f(a)(b)(c)`.
- **Практика:** Створення "спеціалізованих"функцій з частково застосованими аргументами.
- Це дозволяє "зафіксувати"один аргумент (напр., відсоток знижки) і отримати нову функцію, яка чекає на решту аргументів (напр., ціну).

Декоратори

- **Патерн:** Розширення функціонала наявної функції, не змінюючи її код.
- **Синтаксис `@`:** "Синтаксичний цукор"для `func = decorator(func)`.
- **Як це працює:** Декоратор — це HOF, що приймає функцію, визначає "обгортку"(використовуючи замикання) і повертає цю обгортку.
- **Ключовий елемент:** `@functools.wraps` — обов'язковий для збереження метаданих оригінальної функції (`__name__`, `__doc__`).

Модуль 6: Додаток — Точність

Додаток: Забезпечення коректності обчислень

Проблема `float`

- Комп'ютер робить обчислення у бінарному вигляді (двійкові дроби).
- Десяткові дроби, як `0.1`, не можуть бути точно представлені бінарно.
- Це призводить до помилок округлення: `0.1 + 0.2 != 0.3`.

Рішення: `decimal`

- Клас для десяткової арифметики з контрольованою точністю.
- Критично для фінансових розрахунків та будь-де, де точність є вимогою.
- Важливо створювати з рядка: `Decimal("0.1")`, а не `Decimal(0.1)`.

Дякую за увагу!

Запитання?