Programowanie I R

Zadania – seria 10.

Wyjątki, generatory, algebra liniowa z NumPy.

Zadanie 1. `prudent_text_analyzer` – **Wyjątki**
Kluczowym w pisaniu funkcjonalnych kodów jest obsługa wyjątków, tak by przygotować się na wiele potencjalnych problemów, pochodzących na przykład z wadliwego pliku wsadowego, lub źle podanej komendy przez użytkownika. Do obsługi wyjątków służy wbudowana klasa `Exception`. Napisz program `prudent_text_analyzer` do zliczania słów, bezpiecznie obsługujący błędy wejścia/wyjścia.

Zdefiniuj wyjątek `EmptyFileError` dziedziczący ze standardowej klasy `Exception`. Zaimplementuj w nim metody `__init__`, który wywoła inicjalizację klasy matki i przekaże argumenty formatujące komunikat błędu. Następnie utwórz klasę `TextAnalyzer` inicjalizującą w konstruktorze pusty słownik na statystyki wyrazów.

Napisz metodę `load_file(self, filepath)`. Zastosuj blok `try-except-finally` do odczytu pliku. Przechwyć błąd `FileNotFoundError` i zwróć go ze stosownym komunikatem. Jeśli wczytany tekst jest pusty, podnieś ręcznie `EmptyFileError` za pomocą instrukcji `raise`. W bloku `finally` zamknij otwarty plik.

Zaimplementuj metodę `get_word_count`, która zwróci liczbę wystąpień danego słowa w tekście. Zamiast instrukcji warunkowej sprawdzającej obecność klucza, zwróć wartość bezpośrednio w bloku `try`, a brak klucza obsłuż przechwytując `KeyError` i zwracając 0. Przetestuj program na dużym pliku tesktowym, na przykład Panu Tadeuszu. Sprawdź również jego zachowanie podając ścieżkę do pliku nieistniejącego oraz pustego.

Opracowanie: Bartosz Kasza.

Zadanie 2. `fibonacci_generator` – **Generatory**
Napisz program `fibonacci_generator`, który przyjmuje jako argument wywołania liczbę naturalną N. Napisz klasę `FibIterator`, implementującą nieskończony iterator ciągu Fibonacciego. Zdefiniuj stan początkowy w konstruktorze `__init__`, zwracaj instancję obiektu w metodzie `__iter__`, a kolejne wyrazy ciągu obliczaj i zwracaj w metodzie `__next__`. Korzystając z pętli `for x in FibIterator()` oraz instrukcji `break`, znajdź i wypisz pierwszy wyraz ciągu większy od N.

Napisz generator `filter_gen(iterator, select)`, który pobiera wartości z dowolnego iteratora za pomocą pętli `for x in iterator` i zwraca przez `yield` tylko te, dla których funkcja `select(x)` zwraca `True`. Wykorzystaj go do znalezienia pierwszych N parzystych liczb Fibonacciego:

```
evens = filter_gen(FibIterator(), lambda x: x % 2 == 0)
vals = [next(evens) for _ in range(N)]
```

Opracowanie: Bartosz Kasza.

Zadanie 3. `linear_algebra` – **Algebra liniowa z NumPy.**
Do obliczeń numerycznych w Pythonie kluczową biblioteką jest NumPy. Operuje ona na obiektach `numpy.ndarray`, które są wielowymiarowymi tablicami, łatwymi do zmapowania na macierze dla obliczeń fizycznych. W tym zadaniu zaimplementuj podstawowe operacje na macierzach, znane z kursu algebry. Napisz program `linear_algebra`, który w argumencie wywołania przyjmował będzie rozmiar n macierzy kwadratowej $n \times n$, oraz opcjonalnie listę liczb naturalnych będących wymiarem batcha macierzy kwadratowych, a następnie wykonywał poniższe obliczenia.

Podstawowe operacje macierzowe. Zdefiniuj funkcję `matrix` wykonującą poniższe obliczenia i wypisania.
Wygeneruj dwie losowe macierze zespolone A i B rozmiaru $n \times n$:

```
A = np.random.randn(n,n) + 1j*np.random.randn(n,n)
```

Oblicz sumę $A + B$ oraz iloczyn macierzowy $A \times B$ za pomocą operatora @ oraz na indeksach za pomocą `np.einsum`. Mnożenie macierzy w notacji indeksowej dane jest jako:

$$C_{ij} = \sum_k A_{ik} B_{kj}$$

Sprawdź, czy mnożenie macierzy jest przemienne za pomocą `np.allclose`.

Tensory W przestrzeni operatorów, w naszym przypadku w reprezentacji macierzowej, możemy zdefiniować komutator $[A, B] = AB - BA$. Ze względu na działanie macierzą A można traktować go jako liniowe odwzorowanie cztero-indeksowym tensorem $(n \times n \times n \times n)$ (superoperatorem) $\mathcal{L}_A[B]$ działające na przestrzeni macierzy $n \times n$. Jego macierzowa postać w bazie zwektoryzowanych macierzy wynosi:

$$\mathcal{L}_A \;=\; A \otimes \mathbb{1} \;-\; \mathbb{1} \otimes A^T, \tag{1}$$

gdzie $\otimes$ oznacza iloczyn Kroneckera. Napisz funkcję `superop` wykonującą poniższe obliczenia i wypisania:
Zbuduj macierz superoperatora $\mathcal{L}_A$ o wymiarach $n^2 \times n^2$ używając `np.kron` i `np.eye`.
Zwektoryzuj macierz B poleceniem `B.flatten()`, zadziałaj $\mathcal{L}_A$ (mnożenie macierzowe), a następnie odtwórz wynik do kształtu $n \times n$ za pomocą `reshape`.
Zweryfikuj poprawność, porównując wynik z bezpośrednim obliczeniem $AB - BA$.
Oblicz ten sam komutator za pomocą `np.einsum` i porównaj wynik z poprzednimi metodami.

Batch macierzy kwadratowych Powtórz wszystkie obliczenia dla wektora losowych macierzy $a \times n \times n$. W jaki sposób należy zapisać operacje w `numpy`, by działały poprawnie na arrayu dowolnego kształtu $a \times \cdots \times z \times n \times n$?

Test macierzy hermitowskich Zdefiniuj funkcję `hermitian` wykonującą poniższe obliczenia. Napisz w niej funkcję `is_hermitian(M, tol)`, która sprawdza, czy macierz kwadratowa jest hermitowska $M = M^\dagger$ (`np.conj(M.T)`) z precyzją `tol`. Wygeneruj losowy batch macierzy hermitowskich $H = \frac{1}{2}(M + M^\dagger)$ i sprawdź ich hermitowskość z pomocą funkcji `hermitian`. Oblicz wartości własne λ_i macierzy H za pomocą `np.linalg.eigvalsh`. Sprawdź, że wszystkie są rzeczywiste. Zweryfikuj tożsamość $\mathrm{Tr}(H) = \sum_i \lambda_i$ używając `np.trace`. W jaki sposób policzyć ślad z wykorzystaniem `np.einsum`?

Opracowanie: Bartosz Kasza.