УДК 004.43

***Мох Н.С., здобувач,***
***Коршенко В.С., ст. викладач***
*Харківський національний університет ім. В. Н. Каразіна*

# ПОРІВНЯЛЬНИЙ АНАЛІЗ ВИКОРИСТАННЯ ОПЕРАТИВНОЇ ПАМ'ЯТІ МАСИВАМИ NUMPY ТА СТАНДАРТНИМИ СПИСКАМИ PYTHON

Завдяки своїй універсальності, легкості в опануванні та наявності великого вибору бібліотек мова Python сьогодні домінує в багатьох сферах, як-от машинне навчання, робота з великими даними, різного роду автоматизації. Майже в кожному з цих напрямків постійно виникає потреба аналізувати величезні набори чисел, найчастіше представлені у вигляді вбудованих списків або масивів NumPy. У сучасному світі розробка будь-якого програмного рішення повинна зосереджуватися не тільки на вирішенні поставленої проблеми, а й на оптимізації швидкодії та використання пам'яті. Запропонована робота покликана дослідити різницю у використанні оперативної пам'яті при роботі зі стандартними списками Python і масивами NumPy.

Вбудовані списки Python є динамічно типізованими [1]. Це означає, що кожен елемент списку може мати власний тип даних. Забезпечується це парадигмою "Все є об'єктом", що використовується у Python: елементи списку є об'єктами, які мають унікальні властивості; а сам список — це лише набір вказівників на комірки пам'яті, де ці об'єкти зберігаються [3]. Недоліком такого підходу є потреба зберігати в пам'яті не лише дані, а й різну додаткову інформацію (наприклад, вказівники).

Масиви ж NumPy з іншого боку є C-подібними й однотипними, а їхні елементи розташовані в пам'яті послідовно [2]. Така архітектура дозволяє NumPy уникнути накладних витрат на зберігання метаданих для кожного елемента окремо. Відповідно, для рівнозначних наборів числових даних масиви NumPy теоретично повинні займати значно менше оперативної пам'яті, ніж списки Python.

Гіпотеза є такою, що об'єктна модель CPython та спосіб представлення даних, зокрема для списків, а також вся архітектура, що стоїть за ними, може значно збільшувати використання оперативної пам'яті комп'ютера.

Для перевірки гіпотези було написано програму, у якій формувалися три списки Python (на 10 тис., 100 тис. та 1 млн елементів) випадкових чисел від 1000 до 9999 за допомогою модуля random, а також три аналогічні масиви NumPy. За допомогою модуля sys та методу getsizeof() вимірювали фактичне використання оперативної

пам'яті [1]. Вимірювання проводили п’ять разів для усереднення результатів. Запускали програму у хмарному середовищі Google Colab (Python 3 Google Compute Engine).

Вибір діапазону чисел не є випадковим: інтерпретатор Python використовує так зване кешування малих цілих чисел (Small Integer Caching), яке передбачає створення в пам'яті чисел від -5 до 256 одразу в момент запуску програми. Таким чином всі наступні цілочисельні об'єкти в цих межах не будуть створюватися — у їхні змінні просто записуватимуться адреси на вже наявні числа. Така оптимізація може суттєво змінити результати, тому для чистоти експерименту було використано числа більших розмірів.

Також варто зазначити, що sys.getsizeof() не показує реальне використання пам’яті для списків, адже не враховує вкладені об’єкти. Для усунення цієї проблеми було обчислено не лише розмір списку і вказівників у ньому, а й окремо його елементи, після чого визначали їх сумарний обсяг пам’яті.

Отримано наступні усереднені результати для списків: список на 10 тис. чисел займає 365176 байт пам'яті, на 100 тис. — 3600984 байт, а на 1 млн — 36448728 байт. Для масивів: 10 тис. чисел займають 80 тис. байт, 100 тис. — 800 тис., а 1 млн чисел — 8 млн байт відповідно.

Отже, залежність між розміром структури та обсягом пам'яті є майже лінійною для обох варіантів, що підтверджує стабільність і передбачуваність отриманих вимірювань.

Із отриманих результатів видно, що масиви NumPy мають перевагу в 4,5 рази. На відміну від списків для кожного числа NumPy використовує стільки пам'яті, скільки йому потрібно — а саме 8 байт (для dtype=int64). Натомість вбудовані списки використовують набагато більше пам'яті для різної додаткової інформації, а не для важливих користувачу даних.

Таким чином, результати узгоджуються із гіпотезою: об’єктна модель CPython та спосіб представлення даних у списках Python використовують у рази більше оперативної пам'яті — це ціна обраної парадигми.

**Список використаних джерел:**

1. Python Software Foundation. Python 3.14.3 documentation. 2026. URL: https://docs.python.org/3.14/ (дата звернення: 22.03.2026).

2. NumPy Developers. NumPy documentation. 2026. URL: https://numpy.org/doc/ (дата звернення: 22.03.2026).

3. VanderPlas, J. Python Data Science Handbook: Essential Tools for Working with Data / J. VanderPlas. – O'Reilly Media, 2016.