**Лабораторна робота №6**

**Тема. Включення і генератори.**

**Мета роботи**. Освоїти спискові включення і інструменти

функціонального програмування.

**Зміст**.

1. Вивчення відомостей про спискові включення. Генератори.
2. Виконання роботи.
3. Отримання результату.

**Ключові положення.**

Інструменти map і filter - головні члени раннього інструментального набору для функціонального програмування на Python. Вони відображають

операції на ітеруємі об'єкти і накопичують результати. Через те, що ця задача настільки поширена при написанні з'явився новий вираз - спискові включення, які володіють навіть більшою гнучкістю, ніж інші інструменти.

**Формальний синтаксис включень**

Насправді спискові включення навіть більш універсальні. Їх найпростіша форма передбачає зазначення виразу, який накопичується і одиночної конструкції for:

[*Вираз for мета in ітеруємий\_об'ект*]

Незважаючи на необов'язковість всіх інших частин, вони дозволяють висловлювати розвиненіші ітерації - в списковому включенні допускається записувати будь-яку кількість вкладених циклів for, кожний з яких може мати необов'язкову асоційовану перевірку if, що діє як фільтр. Загальна структура спискових включень виглядає наступним чином:

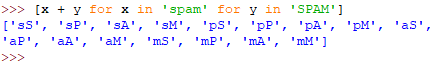
*[Вираз for мета1 in ітеруємий\_об'єкт1 if умова1*

*for мета2 in ітеруємий\_об'ект2 if умова 2. . .*

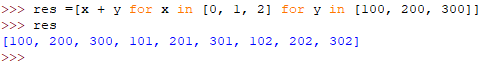
*for мета № in ітеруємий\_об'ект№ if умова № ]*

Такий самий синтаксис успадкований включеннями множин і словників, а також генераторними виразами, які з'явилися пізніше, хоча вони використовують інші символи (фігурні дужки або часто необов'язкові круглі дужки), а включення словника починається з двох виразів, розділених двокрапкою (для ключа і значення).

Наприклад:



Наприклад:



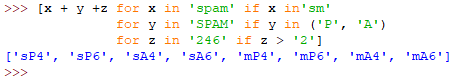
Кожна конструкція for може мати асоційований фільтр if незалежно від

того, наскільки глибоко вкладені цикли.

Наприклад:



Наприклад:



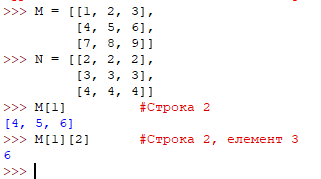
Вираз комбінує парні числа від 0 до 4 з непарними числами від 0 до 4. Конструкції if фільтрують елементи на кожній ітерації.

Наприклад:



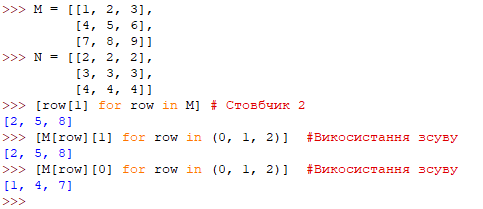
**Спискові включення і матриці**

Наприклад, в наступному коді визначаються дві матриці 3 х 3 у вигляді списків, що складаються з вкладених списків:



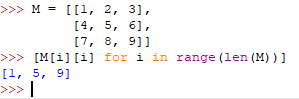
Спискові включення є потужним інструментом для обробки

таких структур як матриці, тому що вони автоматично переглядають рядки і стовпці.

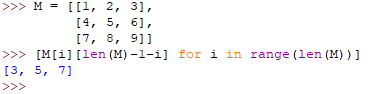


За заданими позиціями ми також можемо виконувати завдання на зразок вилучення діагоналі. У першому з наведених далі виразів із застосуванням функції range генерується список зсувів і проводиться індексація з однаковими номерами рядків і стовпців, що вибирає М [0] [0], потім М [1] [1] і т.д. У другому рядку індекс стовпця врівноважується для вилучення М [0] [2], М [1] [1] і т.д. Ми припускаємо, що матриця має рівну кількість рядків і стовпців.

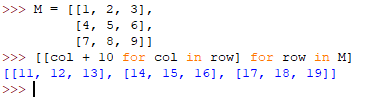
Наприклад:



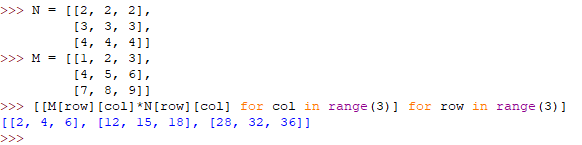
Витягування зворотної діагоналі:



Приклад зміни кожного члена матриці на 10:



Приклад перемноження матриць:



**Генераторні функції і вирази**

**Генераторні функції** записуються як нормальні оператори def, але в них застосовуються оператори yield, щоб повертати по одному результату за раз, призупиняти виконання зі збереженням стану і відновлювати його між видачами.

В великих програмах генератори можуть бути краще в плані пам'яті і продуктивності. Вони дозволяють функціям уникнути виконання всієї роботи заздалегідь, що особливо корисно, коли результуючі списки великі або отримання кожного значення вимагає тривалих обчислень. Генератори розподіляють час, необхідний для виробництва серії значень, за всіма итерациям циклу. Крім того, для ускладнених сценаріїв застосування генераторів здатне запропонувати більш просту альтернативу ручному збереженню стану між ітераціями. В об'єктах класів - генератори забезпечують автоматичне збереження і відновлення змінних, доступних в області видимості функцій.

**Генераторні функції** **yield або return**

**Генераторні функції** тісно пов'язані з поняттям протоколу ітерації в Python. Об'єкти ітераторів визначають метод, який або повертає черговий елемент в ітерації, або ініціює спеціальне виключення Stopiteration для закінчення ітерації. Итератор ітеруємого об'єкта спочатку витягується за допомогою вбудованої функції iter, хоча цей крок нічого не робить для об'єктів, які самі є ітераторами.

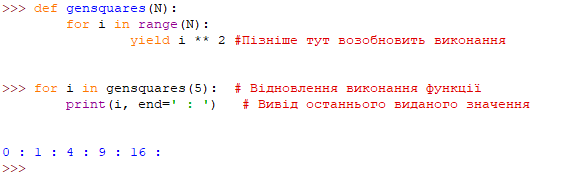
Цикли for в Python і всі інші ітераційні контексти використовують протокол ітерації для проходу по генератору послідовностей або значень, якщо протокол підтримується (якщо ні, тоді ітерація натомість робить багаторазову індексацію послідовностей). Будь-який об'єкт, що підтримує такий інтерфейс, працює з усіма ітераційним інструментами.

Для підтримки протоколу ітерації функції, що містять оператор yield, компілюються в особливий спосіб як генератори - вони не будуть нормальними функціями, а будуються з метою повернення об'єкта з очікуваними методами з протоколу ітерації. При наступному виклику вони повертають об'єкт генератора, який підтримує інтерфейс ітерації з автоматично створеним методом по імені, призначеним для запуску або відновлення виконання.

Генераторні функції можуть також мати оператор return, який поряд з переміщенням за кінець блоку def просто припиняє генерацію значень - формально за рахунок ініціювання виключення Stopiteration після всіх звичайних дій по виходу з функції. З точки зору викликаючого коду, метод генератора відновлює виконання функції до тих пір, поки або не повернеться наступний результат yield, або до виникнення виключення StopIteration.

Сукупний ефект в тому, що генераторні функції, які записані як оператори def, що містять оператори yield, автоматично робляться доступними для протоколу ітерації і тому можуть застосовуватися в будь-якому ітераційному контексті, щоб виробляти результати з плином часу і за запитом.

У наступному коді визначається генераторная функція, яку можна застосовувати для генерації квадратів серії чисел з плином часу:



Функція gensquares видає значення і тому повертає управління коду який викликається на кожній ітерації циклу; при поновленні її виконання відновлюється попередній стан, включаючи останні значення змінних i і N, а управління знову підхоплюється безпосередньо після оператора yield. Скажімо, коли gensquares використовується в тілі циклу for, перша ітерація починає функцію

і отримує перший результат; потім на кожній ітерації циклу управління повертається функції після оператора yield.

Щоб закінчити генерацію значень, функції або застосовують оператор return без значення, або дозволяють потоку управління дійти до кінця тіла функціі.

**Генераторні вирази: ітеруємі об'єкти які зустрічаються з включеннями**

**Генераторні вирази** схожі на спискові включення, але вони не будують результуючий список, а повертають об'єкти, які виробляють результати за запитом. Синтаксично генераторні вирази схожі на нормальні спискові включення і підтримують весь їх синтаксис, в тому числі фільтри if і вкладення циклів, але вони розміщаються в круглі дужки, а не в квадратні (подібно кортежам круглі дужки часто необов'язкові).

У точності як генераторні функції, генераторні вирази забезпечують оптимізацію витрат пам'яті - вони не вимагають створення відразу всього результуючого списку, що відбувається в разі спискового включення в квадратних дужках. Також подібно генераторним функцій вони поділяють роботу з випуску результатів на невеликі тимчасові інтервали - результати видаються поступово замість того, щоб викликати код та очікувати створення повного набору в єдиному виклику.

З іншого боку, на практиці генераторні вирази можуть виконуватися трохи повільніше спискових включень, а тому їх найкраще застосовувати для дуже великих результуючих наборів або в додатках, які не можуть чекати генерації повних результатів.

Наприклад:

Генераторний вираз будує інтеруємий об’єкт.





Для примусового формування всіх результатів відразу використовуємо вбудовану функцію list.

**Лабораторне завдання**

1. Написати програму, яка комбінує непарні числа від 0 до 7 з парними числами від 0 до 7 за допомогою списків включень.
2. Для матриці А [[11, №, 44, 55],[1, 2, №, 4],[11, 12, 13, 14],[21, 31, 41, №]]. Вивести на екран: головну діагональ, зворотну діагональ, другий стовпчик матриці. Збільшити кожен член матриці на №. Помножити матрицю А на матрицю М де M = [[1, 2, 3,6], [4, 5, 6,0], [7, 8, 9,7], [71, 8, 93,7]]. Де №- остання цифра номеру студента у списку групи.
3. За допомогою генераторної функції та **yield** написатикод для генерації кубів серії чисел з плином часу. Вивести № сгенерованих чисел. Де №- остання цифра номеру студента у списку групи.