```
            s=s+1/(i+j**2);
print('Сума =', s)
```

7. Структури даних

Для спрощення написання і виконання програми буває зручно окремі дані об'єднувати в певні структури. Від того, наскільки вдало будуть вибрані ці структури, суттєво залежить ефективність програми.

Структури даних - це спосіб організації даних. У мові Python існують вбудовані структури даних, серед яких є: послідовності (списки, кортежі, діапазони), бінарні послідовності, рядки, множини, словники.

Для вбудованих структур даних в мові Python передбачений набір стандартизованих функцій.

`len(iterable)`. Повертає число елементів (довжину) `iterable`.

`max(iterable, *[, default=obj, key=func])`. Повертає максимальний елемент із `iterable`.

```
>>> max([2,3,4])
4
```

`min(iterable, *[, default=obj, key=func])`. Повертає мінімальний елемент із `iterable`.

```
>>> min([2,3,4])
2
```

`sum(iterable[, start])`. Повертає суму членів числового `iterable`, починаючи з елемента з індексом `start`. За замовчуванням `start = 0`.

```
>>> sum([2,3,4])
9
```

`map(func, *iterables)`. Застосовує функцію `func` до кожного елемента із `iterable`. Результатом є об'єкт, що підтримує ітерування (ітератор).

```
>>> list(map(bin, [1,3,5]))
['0b1', '0b11', '0b101']
```

`enumerate(iterable, start=0)`. Повертає кортеж (порядковий_номер_елемента, значення_елемента), отриманий з `iterable`.

```
>>> a=['a','b','c']
>>> for i, v in enumerate(a): print(i, v)
0 a
1 b
2 c
```

7.1. Списки

Список (list) – це структура даних для зберігання елементів (об'єктів) не обов'язково одного типу. Це частково схоже на масиви в інших мовах програмування, але головною особливістю є те, що елементами списку в мові Python можуть бути елементи різних типів. Список є змінюваним типом даних. Списки записуються як перелік елементів, розділених комою та взятих у квадратні дужки: `[1, 2, 3, 'Hello']`.

7.1.1. Задання списків

Для задання порожнього списку можна скористатися однією з наступних команд:

```
>>> a=[]
>>> a=list()
```

Задання списку з наперед заданим набором елементів:

```
>>> a=[1, 2, 3, 4]
>>> b=['Hello', 2, True]
```

Створення списку з інших структур даних

Список можна отримати з елементів об'єкту, що може ітеруватися (діапазон, рядок, словник, множина, кортеж, файл і т.д.) використавши функцію **`list([iterable])`**:

```
>>> b=list('Hello')
>>> print(b)
['H', 'e', 'l', 'l', 'o']
>>> c=list(range(10))
>>> print(c)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Об'єднання списків

Нові списки можуть бути створені методом конкатенації (об'єднання) декількох списків в один. Для цього використовується перевизначена операція додавання («+»).

```
>>> b=[1, 3, 5, 7, 9]
>>> d=[10, 11, 12]
>>> c=b+d
>>> print(c)
[1, 3, 5, 7, 9, 10, 11, 12]
```

Багаторазове повторення елементів

Аналогічно до перевизначеної операції («+»), для списків в мові Python перевизначена і операція множення «*». Якщо виконати операцію «*» списку `a` на ціле число `n`, то в результаті буде отриманий список, що складається з `n` повторень списку `a`:

```
>>> a=[0]*5
>>> print(a)
[0, 0, 0, 0, 0]
```

Введення або генерування елементів списку

Числові елементи списку можна згенерувати випадковим чином, скориставшись функціями модуля random, наприклад:

```
import random
n=int(input('Введіть кількість елементів списку ='))
a=[]
for i in range(n):
     x=random.randint(1,100)
     a.append(x)
```

Для випадку, коли необхідно створити список з елементів, що будуть задаватися користувачем, можна скористатися наступним фрагментом коду:

```
n=int(input('Введіть кількість елементів списку = '))
a=[]
for i in range(n):
     x=int(input('a[{}]='.format(i)))
          # або x=int(input('a['+str(i)+']='))
```

```
    a.append(x)
```

Якщо ж елементи списку будуть задаватися одним рядком через пропуск, то для формування такого списку можна скористатися наступним фрагментом коду:

```
a=list(map(int, input('Введіть елементи списку через
          пропуск: ').split()))
```

Генератори списків

Для задання списків також можуть бути використані так звані ***генераторні списки*** (List Comprehensions), які інколи також називають «абстракція списків» або «спискові включення». Генераторні списки є частиною синтаксису мови Python, яка надає простий спосіб побудови списків.

Генератори списків забезпечують лаконічний спосіб створення списку у тому випадку, коли кожен елемент списку є результатом деякої операції, застосованої до кожного елемента іншого списку (послідовності), або створення списку з тих елементів, які задовольняють конкретну умову.

Генератор списку складається з квадратних дужок, що містять вираз формування елемента списку, і циклу for, за яким відбувається перебір елементів іншої «базової» послідовності. Окрім того, елементи, що перебираються циклом for, можуть бути обмежені наявністю умови if.

Створення списку цілих чисел від 0 до 10:

```
>>> a=[i for i in range(11)]
>>> print(a)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Створення списку, який містить 5 нулів:

```
>>> a=[0 for i in range(5)]
>>> print(a)
[0, 0, 0, 0, 0]
```

Створення списку з елементів, які є квадратами чисел від 1 до 4:

```
>>> a=[i ** 2 for i in range(1, 5)]
>>> print(a)
[1, 4, 9, 16]
```

Створення списку цілих непарних чисел, менших за 10:

```
>>> a=[i for i in range(10) if i%2==1]
```

```
>>> print(a)
[1, 3, 5, 7, 9]
```

Створення списку з 10 елементів, що заповнений випадковими числами від 1 до 9:

```
>>> import random
>>> a=[random.randrange(1, 10) for i in range(10)]
>>> print(a)
[8, 7, 1, 3, 4, 6, 1, 3, 7, 1]
```

Створення списку, елементами якого будуть пари чисел, які є елементами двох інших списків:

```
>>> a=[ (x, y) for x in [1, 2, 3] for y in [4, 5]]
>>> print(a)
[(1, 4), (1, 5), (2, 4), (2, 5), (3, 4), (3, 5)]
```

Попередня команда еквівалентна, до

```
a=[]
for x in [1,2,3]:
     for y in [4, 5]:
         a.append((x, y))
print(a)
[(1, 4), (1, 5), (2, 4), (2, 5), (3, 4), (3, 5)]
```

7.1.2. Доступ до елементів списку. Зрізи

Доступ до елементів списку відбувається за їх індексами. Як і в багатьох інших мовах, нумерація елементів починається з нуля. Для того, щоб звернутися до елемента списку, необхідно вказати ім'я змінної списку та в квадратних дужках індекс необхідного елемента (`ім'я_списку[індекс]`).

```
>>> b=[1, 3, 5, 7, 9]
>>> b[1]
3
```

При спробі доступу до неіснуючого індексу виникає виняток IndexError.

```
>>> b[5]
Traceback (most recent call last):
```

```
  File "<pyshell#27>", line 1, in <module>
    b[5]
IndexError: list index out of range
```

Індекси можуть бути від'ємними, в такому випадку нумерація буде відбуватися з кінця (кількість елементів списку + від'ємний індекс).

```
>>> b[-1]
9
```

Якщо ж розглянути всі індекси для списку [1, 3, 5, 7, 9] , отримаємо:

список b	1	3	5	7	9
Індекс	b[0]	b[1]	b[2]	b[3]	b[4]
Індекс	b[-5]	b[-4]	b[-3]	b[-2]	b[-1]

Можна перевірити приналежність деякого елемента до списку, використовуючи оператор in (`значення in ім'я_списку`).

```
>>> a=[1, 3, 5, 7]
>>> 3 in a
True
>>> 4 in a
False
```

Для протилежної перевірки, що елемент не належить списку, може бути використаний оператор not in.

```
>>> 4 not in a
True
```

Раніше зазначалося, що використовуючи цикл for, можна перебрати всі елементи послідовності, а отже і списку. Тому команда виведення елементів списку окремими рядками буде виглядати так:

```
>>> for i in b:
      print(i)
```

А команда виведення елементів списку через пропуск, так:

```
>>> for i in b:
      print(i, end=' ')
```

Зрізи (slice)

Досить часто необхідно отримати не один елемент списку за індексом, а деякий набір елементів за певним простим правилом. Наприклад: перші 5 елементів, кожен другий елемент. В таких завданнях замість перебору в циклі набагато зручніше використовувати так званий зріз (slice, slicing). Зріз – отримання з даного списку набору з його елементів. Зріз списку також є списком. Отримання одного елемента списку є найпростішим варіантом зрізу.

Задати зріз можна одним з двох варіантів:

`item[start: stop]`.

`item[start: stop: step]`.

Для списку `item` береться зріз від індексу `start`, до `stop` (не включаючи його), з кроком `step`. Також при записі зрізу деякі, а можливо і всі параметри можуть бути опущені (знаки двокрапки в записі все рівно залишаються). У випадку відсутності деяких параметрів їх значення встановлюється за замовчуванням, а саме: start = 0, stop = кількості елементів списку, step = 1.

```
>>> a = [1, 3, 8, 7]
>>> a[1:3]
[3, 8]
```

Якщо опустити другий параметр (залишивши двокрапку), то зріз береться до кінця рядка. Наприклад, щоб отримати зріз без першого елемента, можна записати `a[1:]`. Якщо ж опустити перший параметр, то отримаємо зріз, який містить вказану кількість елементів, що йдуть на початку списку.

```
>>> a[1:]
[3, 8, 7]
>>> a[:3]
[1, 3, 8]
>>> a[:]
[1, 3, 8, 7]
```

При заданні значення третього параметра, рівного 2, у зріз потрапить кожний другий елемент списку.

```
>>> a[::2]
```

```
[1, 8]
>>> a[1::2]
[3, 7]
```

У випадку, якщо параметри start і stop мають від'ємні значення, то нумерація відбуватиметься з кінця (кількість символів рядка + від'ємний індекс). Наприклад, `a[-2:]` - це список з останніх двох елементів.

```
>>> a[-2:]
[8, 7]
>>> a[:-2]
[1, 3]
>>> a[-3:-1]
[3, 8]
```

Якщо параметр step має від'ємне значення, то зріз береться справа наліво.

```
>>> a[::-1]
[7, 8, 3, 1]
>>> a[-2::-1]
[8, 3, 1]
>>> a[1:4:-1]
[]
```

В останньому прикладі був отриманий порожній список, так як `start < stop`, а `step` від'ємний. Те ж саме відбудеться, якщо діапазон індексів виявиться за межами списку.

```
>>> a[10:20]
[]
```

7.1.3. Зміна та вилучення елементів списку

Оскільки списки є змінюваним типом даних, то окремі елементи списку можуть бути змінені чи вилучені.

В цьому пункті буде розглянуто виконання дій над елементами списків без застосування спеціальних методів списків, які подані далі.

Зміна елементів списку

Враховуючи, що список є змінюваним типом даних, то за необхідності можна змінити один чи кілька елементів списку. Для зміни елемента списку необхідно вказати ім'я змінної списку та в квадратних дужках індекс необхідного елемента та виконати присвоєння нового значення (`ім'я_списку[індекс]=нове_значення`).

```
>>> b=[1, 3, 5, 7, 9]
>>> b[2]=10
>>> print(b)
[1, 3, 10, 7, 9]
```

Змінювати також можна не один елемент, а відразу декілька, використовуючи зрізи.

```
>>> b[1:3]=[11,12]
>>> print(b)
[1, 11, 12, 7, 9]
```

Також використовуючи зріз, елементи можна навіть додавати.

```
>>> b[1:3]=[2,3,4,5,6]
>>> print(b)
[1, 2, 3, 4, 5, 6, 7, 9]
>>> b[len(b):]=[11,12,13]
>>> print(b)
[1, 2, 3, 4, 5, 6, 7, 9, 11, 12, 13]
```

Проте варто зауважити, що при спробі додати список елементів не зрізу, а одному елементу, отримаємо заміну цього елемента на окремий список.

```
>>> b[0]=[0, 1]
>>> print(b)
[[0, 1], 2, 3, 4, 5, 6, 7, 9, 11, 12, 13]
```

Вилучення елементів списку

Для вилучення елемента списку за його індексом можна скористатися командою `del` (`del ім'я_списку[індекс]`).

```
>>> a=[1, 3, 5, 1, 3]
>>> del a[2]
>>> print(a)
```

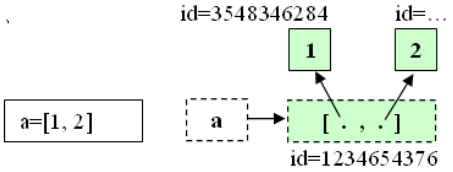
```
[1, 3, 1, 3]
```

Також команда `del` може бути використана для вилучення зрізу із списку чи очищення всього списку

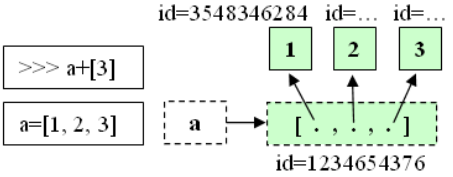
```
>>> del a[1:3]
>>> print(a)
[1, 3]
>>> del a[:]
>>> print(a)
[]
```

7.1.4. Змінюваність типу список

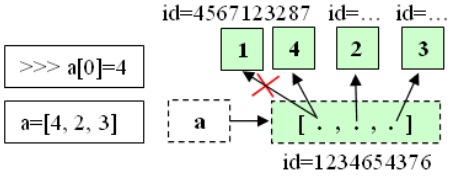
Змінна, визначена як список, містить посилання на область в пам'яті, яка в свою чергу містить посилання на елементи (об'єкти) цього списку. На відміну від числових типів даних, список є змінюваним типом даних і тому вміст списку може бути змінений, розширений чи зменшений (Рис. 7.1).



а) Створення списку



б) Додавання нового елемента до списку



в) зміна елемента списку

Рис. 7.1. Подання дій з списками

Присвоювання списків

При використанні звичайного присвоювання змінної одного списку іншій змінній відбувається присвоювання новій змінній лише посилання на той же об’єкт в пам'яті, на який посилався початковий список. Тобто дві змінні будуть посилатися на один той же об’єкт в пам'яті, і якщо ви будете змінювати один список, то буде змінюватися і інший.

```
>>> a=[1, 3, 5, 7, 9]
>>> b=a
>>> a[1]=2
>>> print(b)
[1, 2, 5, 7, 9]
>>>> id(a)
19733680
>>>> id(b)
19733680
```

Тому для створення нового списку, який буде копією даного, необхідно скористатися зрізом, або функцією list().

```
>>> a=[1, 3, 5, 7, 9]
>>> b=a[:]
>>> a[1]=2
>>> print(b)
[1, 3, 5, 7, 9]
>>> c=list(a)
>>> a[2]=4
>>> print(c)
[1, 2, 5, 7, 9]
>>> id(a)
17285288
>>> id(b)
17343232
>>> id(c)
24285432
```

7.1.5. Методи списків

list.append(x). Додає елемент x в кінець списку list. Аналогічно командам a[len(a):]=[x].

```
>>> a=[1,2]
>>> a.append(3)
>>> print(a)
[1, 2, 3]
```

list.extend(iterable). Розширює існуючий список list за рахунок додавання до нього всіх елементів з iterable. Аналогічно командам a[len(a):]=iterable.

```
>>> a=[1,2]
>>> a.extend([3,4])
>>> print(a)
[1, 2, 3, 4]
```

list.insert(n, x). Вставляє в список list елемент x в позицію n (індекс елемента, після якого буде вставлений елемент).

```
>>> a=[1,2]
>>> a.insert(1, 5)
>>> print(a)
[1, 5, 2]
>>> a.insert(len(a), 9)
>>> print(a)
[1, 5, 2, 9]
```

list.remove(x). Вилучає перше входження елемента x зі списку list.

```
>>> a=[1, 2, 3, 1, 2]
>>> a.remove(1)
>>> print(a)
[2, 3, 1, 2]
```

list.pop([n]). Вилучає з списку list елемент з позиції n та повертає його, як результат виконання функції. Якщо використовувати метод без параметру, то буде вилучений останній елемент списку.

```
>>> a=[1,2,3,4,5]
```

```
>>> print(a.pop(2))
3
>>> print(a.pop())
5
>>> print(a)
[1, 2, 4]
```

list.clear(). Очищує список list (вилучає всі елементи зі списку). Аналогічно до del a[:].

```
>>> a=[1,2,3,4,5]
>>> a.clear()
>>> print(a)
[]
```

list.index(x[, start[, end]]). Повертає індекс першого входження елемента x в зрізі list[start: end] (необов'язкові параметри start та end інтерпретуються як нотації зрізу). Значення, що повертається, є індексом списку list. Якщо елемента в списку не знайдено, виникає виняток ValueError.

```
>>> a=[1, 2, 3, 1, 2, 3, 1, 2, 3]
>>> print(a.index(3))
2
>>> print(a.index(3,3))
5
```

list.count(x). Повертає кількість входжень елемента x в список.

```
>>> a=[1, 2, 3, 1, 2, 3, 1, 2, 3]
>>> print(a.count(2))
3
```

list.sort(key=None, reverse=False). Відсортовує елементи списку (аргументи методу можуть бути використані для налаштування сортування). За замовчуванням сортування відбувається за зростанням. Для сортування в зворотному порядку використовуйте параметр reverse = True. В результаті сортування змінюється сам список.

```
>>> a=[1, 2, 4, 3, 2, 6, 5]
>>> a.sort()
>>> print(a)
```

```
[1, 2, 2, 3, 4, 5, 6]
>>> a.sort(reverse=True)
>>> print(a)
[6, 5, 4, 3, 2, 2, 1]
```

`list.reverse()`. Змінює порядок розташування елементів у списку на зворотний. Змінюється сам список.

```
>>> a=[1, 2, 4, 3, 2, 6, 5]
>>> a.reverse()
>>> print(a)
[5, 6, 2, 3, 4, 2, 1]
```

`list.copy()`. Повертає копію списку. Аналогічно `a[:]`.

```
>>> a=[1, 3, 5]
>>> b=a.copy()
>>> print(b)
[1, 3, 5]
```

7.1.6. Порівняння списків

Списки можна порівнювати між собою. Списки порівнюються поелементно, тому можна порівнювати списки лише в тому випадку, якщо їх відповідні елементи мають однаковий тип чи мають відповідні методи порівняння. При порівнянні використовується лексикографічний порядок: спочатку порівнюються перші два елементи. Якщо вони різні, то вони і визначають результат порівняння; якщо вони рівні, порівнюються наступні два елементи, і т.д., поки одну з двох послідовностей не буде вичерпано. Якщо два порівнюваних елемента самі є списками, то порівняння здійснюється рекурсивно. Якщо всі елементи списків рівні, то списки вважаються рівними. Якщо один список збігається з початком іншого, більш короткий список вважається меншим.

Кілька прикладів порівняння, результатом яких буде істина (True):

```
[1, 2, 3]               < [1, 2, 4]
[1, 2, 3, 4]            < [1, 2, 4]
[1, 2]                  < [1, 2, -1]
[1, 2, 3]              == [1.0, 2.0, 3.0]
[1, 2, ['aa', 'ab']]    < [1, 2, ['abc', 'a'], 4]
```

7.1.7. Вкладені списки

Часто в задачах доводиться зберігати прямокутні таблиці з даними. Такі таблиці називаються матрицями або двовимірними масивами.

В загальному випадку матриця записується так:

$$A_{m\times n}=\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & & a_{mn} \end{bmatrix}$$

За аналогією з одновимірними масивами, які в мові Python подаються списками, двовимірний масив можна подати у вигляді списку (двовимірного списку), кожен елемент якого в свою чергу є списком.

```
>>> a=[[1, 2, 3], [4, 5, 6]]
```

Тобто маємо числовий двовимірний масив з двох рядків (перший вимір) та трьох стовпців (другий вимір).

До елементів такого списку можна звертатися так само, як і до елементів звичайного списку, лише враховуючи, що перший вимір (рядки) також є списком.

```
>>> print(a[0])
[1, 2, 3]
>>> print(a[0][1])
2
>>> a[1][0]=10
>>> print(a)
[[1, 2, 3], [10, 5, 6]]
```

За аналогією можна описати і тривимірні списки, і списки з нескінченною вкладеністю.

Враховуючи, що в Python на списки ніяких кількісних обмежень не накладається, то вкладені списки можуть мати різну кількість елементів, тобто a=[[1, 2], [4, 5, 6]].

Для обробки і виведення вкладених списків, як правило, використовують вкладені цикли, кожен з яких буде проходити по відповідному виміру списку. Так, для опрацювання двовимірних списків

необхідно два цикли: перший цикл буде проходити по елементах основного списку, другий цикл буде проходити по елементах вкладених списків.

Задання двовимірних списків

Заповнити двовимірний список (масив) розмірності 4x5 нулями можна декількома способами.

Перший спосіб. Спочатку створити список з 4-х нульових елементів. Далі розширити кожен елемент списку одновимірним списком з 5-и елементів:

```
a=[0] * 4
for i in range(4):
     a[i]=[0] * 5
```

Другий спосіб. Створити порожній список, потім 4 раз додати до нього новий елемент, який є списком з 5-и нулів:

```
a=[]
for i in range(4):
     a.append([0] * 5)
```

Проте, ще простіше скористатися генератором: створити список з 4-х елементів, кожен з яких буде списком з 5-и нулів:

```
a = [[0] * 5 for i in range(4)]
```

Заповнення двовимірного масиву розмірності 4x5 значеннями, заданими користувачем, можна виконати наступним чином:

```
a=[]
for i in range(4):
     b=[]
     for j in range(5):
          x=int(input('a[{},{}]='.format(i, j)))
          b.append(x)
     a.append(b)
```

Або використовуючи генератори списків:

```
a = [[int(input('a[{},{}]='.format(i, j))) for j in
     range(5)] for i in range(4)]
```

Виведення вкладених списків

Для виведення двовимірного масиву в «природньому» вигляді (елементи першого виміру мають записуватися окремими рядками, а елементи другого виміру в межах одного рядка мають розділятися пропусками) можна скористатися наступним фрагментом програми.

```
for row in a:
     for elem in row:
          print(elem, end=' ')
     print()
```

7.1.8. Приклади розв'язування задач

Приклад. Дано список з 10 випадкових дійсних чисел. Підрахувати суму, добуток і середнє арифметичне елементів списку.

```
import random
v=[random.uniform(1, 100) for i in range(10)]
print('Елементи списку:')
for i in v:
     print(i)
print()
s=sum(v)
d=1
for i in v:
     d=d*i
a=s/len(v)
print('Сума = ',s)
print('Добуток = ',d)
print('Сер. арифм. = ',a)
```

Приклад. Дано списки A, B (кожний з 10 цілих випадкових чисел). Побудувати список C за правилом: $c_i = a_i^2 + b_i^2$.

```
import random
a=[random.randint(1, 100) for i in range(10)]
b=[random.randint(1, 100) for i in range(10)]
print('Елементи списку a: ', a)
print('Елементи списку b: ', b)
```

```
c=[]
for i in range(10):
     c.append(a[i]**2+b[i]**2)
print('Елементи списку c: ', c)
```

Приклад. Дано список з 15 цілих випадкових чисел. Знайти максимальне значення елемента списку та підрахувати кількість елементів списку з таким значенням.

```
import random
a=[random.randint(1, 10) for i in range(15)]
print('Елементи списку a: ', a)
m=max(a)
c=a.count(m)
print('Максимальний елемент списку: ', m)
print('Кількість максимальних елементів: ', c)
```

Приклад. Дано цілочисельну матрицю випадкових цілих чисел розміром m×n (m і n задаються користувачем). Знайти мінімальний елемент матриці та його індекси.

Розглянемо декілька способів розв'язування даної задачі. Проте всі вони матимуть однаковий початок, в якому буде задаватися розмірність матриці, генеруватиметься матриця і виводитиметься в «природному» вигляді.

```
import random
n=int(input('Введіть кількість рядків матриці: '))
m=int(input('Введіть кількість стовпців матриці: '))
a=[]
for i in range(n):
     b=[]
     for j in range(m):
          x=random.randint(10,99)
          b.append(x)
     a.append(b)
print('Згенерована матриця:')
for row in a:
     for elem in row:
```

```
            print(elem, end=' ')
        print()
```

Спосіб 1. Розв'язування методом перебору.

```
. . .
min_el=a[0][0]
min_i=0
min_j=0
for i in range(n):
    for j in range(m):
        if a[i][j]<min_el:
            min_el=a[i][j]
            min_i=i
            min_j=j
print('Мінімальний елемент: ',min_el)
print('Індекс мінімального елемента
          [{}, {}]'.format(min_i,min_j))
```

Спосіб 2. Розв'язування з використаням методів списків.

```
. . .
min_col=list(map(min, a))
min_el=min(min_col)
min_i=min_col.index(min(min_col))
min_j=a[min_i].index(min_el)
print('Мінімальний елемент: ',min_el)
print('Індекс мінімального елемента
          [{}, {}]'.format(min_i,min_j))
```

Якщо ж матриця буде містити декілька рівних елементів, які і будуть мінімальними, то запропоновані способи не підходять.

Спосіб 3. Пошук індексів для всіх мінімумів

```
. . .
min_col=list(map(min, a))
min_el=min(min_col)
print('Мінімальний елемент: ',min_el)
for i in range(n):
    for j in range(m):
```

```
            if a[i][j]==min_el:
                print('Індекс мінімального елемента
              [{}, {}]'.format(i,j))
```

Приклад. Дано цілочисельну матрицю випадкових цілих чисел розміром 4×4. Написати програму для транспонування матриці (переставлення стовпчиків та рядків).

```
import random
a = [[random.randint(10,99) for j in range(4)] for i
in range(4)]
print('Згенерована матриця:')
for row in a:
    for elem in row:
        print(elem, end=' ')
    print()
#Тут має міститися блок транспонування матриці
print('Транспонована матриця:')
for row in t:
    for elem in row:
        print(elem, end=' ')
    print()
```

Спосіб 1. Транспонування вкладеними циклами:

```
t = []
for i in range(4):
    t_row = []
    for row in a:
        t_row.append(row[i])
    t.append(t_row)
```

Спосіб 2. Згортання внутрішнього циклу в генератор списків:

```
t = []
for i in range(4):
    t.append([row[i] for row in a])
```

Спосіб 3. Згортання зовнішнього циклу в генератор списків:

```
t=[[row[i] for row in a] for i in range(4)]
```