**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)

1. a
2. b
3. 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]

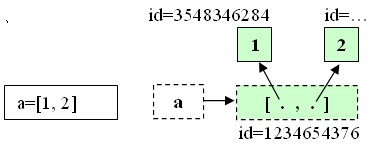
>>> del a[:]

>>> print(a)

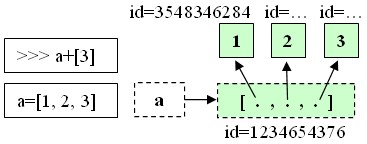
[]

### 7.1.4. Змінюваність типу список

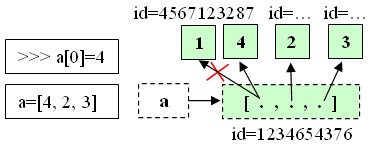
Змінна, визначена як список, містить посилання на область в пам'яті, яка в свою чергу містить посилання на елементи (об'єкти) цього списку. На відміну від числових типів даних, список є змінюваним типом даних і тому вміст списку може бути змінений, розширений чи зменшений (Рис. 7.1).



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*11  *a*21  *Am**n*    ...    *am*1 | *a*12  *a*22  ...  *am*2 | ...  ...  ... | *a*1*n*   *a*2*n*  ...  *amn*  |

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 4х5 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 4х5 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=s\*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: ci= ai2 + bi2. 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)]

## 7.2. Кортежі

Кортеж (tuple) - це незмінна структура даних, яка за своєю будовою дуже схожа на список. Інколи навіть кажуть, що кортеж – це незмінюваний список. Так само, як і список, кортеж може містити елементи різних типів. Кортеж записується, як перелік елементів, розділених комою та взятих в круглі дужки: (1, 3, 5, 'Hello').

Існує кілька причин, коли варто використовувати кортежі замість списків. Першою причиною є можливість захисту даних від випадкової зміни (захист від дурня). Якщо ми отримали набір даних, і є необхідність опрацьовувати його без зміни даних, то це як раз той випадок, коли доцільно використати кортеж.

Другою причиною є те, що кортежі в пам'яті займають менший об'єм у порівнянні зі списками.

>>> lst=[1, 2, 3]

>>> tpl=(1, 2, 3)

>>> print (lst.\_\_sizeof\_\_()) 32

>>> print (tpl.\_\_sizeof\_\_())

24

Третьою причиною є приріст продуктивності, який пов'язаний з тим, що кортежі працюють швидше, ніж списки (наприклад, операції перебору елементів). Четвертою причиною є можливість використання кортежів в якості ключа у словнику.

>>> d = {(1, 1, 1) : 1}

>>> d = {[1, 1, 1] : 1}

Traceback (most recent call last): File "<pyshell#3>", line 1, in <module> d = {[1, 1, 1] : 1}

TypeError: unhashable type: 'list'

### 7.2.1. Задання кортежів

Для задання порожнього кортежу можна скористатися однією з наступних команд:

>>> a=()

>>> b=tuple()

Задання кортежу з наперед заданим набором елементів:

>>> a=(1, 3, 5, 'Hello')

Окрім того, будь-який набір різних об’єктів, розділених комами та не виділених будь-якими дужками (тобто ні квадратними, ні фігурними, ні круглими) за замовчуванням буде вважатися кортежем:

>>> b=1, 3, 5, 'Hello'

>>> print(b)

(1, 3, 5, 'Hello')

Проте створення кортежу з одного елемента має певні особливості. Так, ввівши команду a = (3) , отримаємо в змінній a лише число, а не кортеж

>>> a=(3)

>>> print(a)

3

Для створення кортежу з одного елемента можна записати:

>>> a=(3,)

>>> print(a)

(3,)

## Створення кортежів з інших структур даних

Кортеж можна отримати з елементів об’єкта, що може ітеруватися (діапазон, рядок, словник, множина, кортеж, файл і т.д.), використавши функцію **tuple([iterable])**:

>> b=tuple(range(1,10,2))

>>> print(b)

(1, 3, 5, 7, 9)

>>> b=tuple('Hello')

>>> print(b)

('H', 'e', 'l', 'l', 'o')

## Об’єднання кортежів

Нові кортежі можуть бути створені методом конкатенації (об’єднання) декількох кортежів. Для цього використовується перевизначена операція додавання («+»), яка використовується як операція конкатенації кортежів. >>> b=(1, 3, 5, 7, 9)

>>> b=b + (10,)

>>> print(b)

(1, 3, 5, 7, 9, 10)

## Багаторазове повторення елементів

Аналогічно до перевизначеної операції («+») для кортежів, в мові Python перевизначена і операція множення «\*». Якщо виконати операцію «\*» кортежу b на ціле число n, то в результаті буде отриманий кортеж, що складається з n повторень кортежу b:

>>> b=(0,)\*5

>>> print(b)

(0, 0, 0, 0, 0)

**7.2.2. Виконання дій над кортежами та їхніми елементами**

## Доступ до елементів кортежу

Доступ до елементів кортежу здійснюється аналогічно доступу до елементів списку – за їхніми індексами. Тобто для того, щоб звернутися до елемента кортежу, необхідно вказати ім’я змінної кортежу та в квадратних дужках індекс необхідного елемента (ім’я\_кортежу[індекс]).

>>> a=(1, 3, 5, 7) >>> print(a[0])

1

Те ж саме стосується і зрізів кортежів. Задати зріз можна одним з двох варіантів: **tuple[start: stop]**.

**tuple[start: stop: step]**.

>>> print(a[1:3])

(3, 5)

Можна перевірити приналежність деякого елемента до кортежу, використовуючи оператор in (значення in ім’я\_кортежу).

>>> a=(1, 3, 5, 7)

>>> 3 in a

True

>>> 3 in a

False

Але, як вже було сказано - змінювати елементи кортежу не можна. При спробі змінити чи вилучити елемент кортежу виникне виняток TypeError.

>>> b=(1, 2, 3)

>>> b[1]=15

Traceback (most recent call last): File "<pyshell#1>", line 1, in <module> b[1]=15

TypeError: 'tuple' object does not support item assignment

>>> del b[1]

Traceback (most recent call last): File "<pyshell#49>", line 1, in <module> del b[1]

TypeError: 'tuple' object doesn't support item deletion

## Вилучення кортежів

Як було зазначено, вилучити окремий елемент з кортежу неможливо, але можна видалити кортеж повністю, скориставшись командою del (del ім’я\_кортежу).

>>> a=(1, 3, 5, 7)

>>> del a

>>> print(a)

Traceback (most recent call last): File "<pyshell#27>", line 1, in <module> print(a)

NameError: name 'a' is not defined

## Перетворення кортежів у список і назад

На основі існуючого кортежу можна створити список, як і навпаки - зі списку можна створити кортеж. Тобто, використовуючи кортеж для захисту даних, за необхідності можна перетворити його в список, внести необхідні зміни і знову повернути в кортеж.

>>> b=(1,2,3)

>>> b=list(b)

>>> b[1]=5

>>> b=tuple(b)

>>> print(b)

(1, 5, 3)

## Методи кортежів

Враховуючи незмінюваність кортежів, вони мають лише методи:

tuple.index(x[, start[, end]]) та tuple.count(x)призначення яких аналогічне до призначенням однойменних методів списків.

## 7.3. Словники

Звичайні списки являють собою набір пронумерованих елементів, отже, для звернення до деякого елемента списку необхідно вказати його номер (індекс). Номер елемента в списку однозначно ідентифікує сам елемент. Проте досить часто ідентифікувати дані лише за числовими індексами не завжди зручно.

Словник (dict) – це структура даних, призначена для зберігання довільних об'єктів з доступом за довільним ключем. Дані в словнику зберігаються в форматі ключ=значення. Ключі в межах словника мають бути унікальними, тобто двох однакових ключів в словнику бути не може. Ключ повинен мати незмінюваний тип даних: ціле або дійсне число, рядкок, кортеж. Враховуючи зазначену структуру, словник інколи ще називають асоціативним масивом.

Словник записується, як перелік пар ключ : значення, розділених комою та взятих в фігурні дужки: {'A1':2, 'A2':3}.

Словник є змінюваним типом даних: в нього можна додавати нові елементи з довільними ключами і вилучати вже існуючі елементи. При цьому розмір використовуваної пам’яті пропорційний розміру словника. Доступ до елементів словника виконується хоча і повільніше, ніж до звичайного списку, але в цілому достатньо швидко.

### 7.3.1. Створення словників

Для задання порожнього словника можна скористатися однією з наступних команд:

>>> a={}

>>> b=dict()

Задання словника з наперед заданим набором елементів:

>>> a={'A1':2, 'A2':3}

>>> b=dict(id1=4, id2=8)

>>> print(b)

{'id1': 4, 'id2': 8}

Варто відмітити, що у випадку створення словника, в якого ключем має виступати число або кортеж, можливе використання лише першої команди:

>>> a={1: 'A1', 2: 'A2'}

>>> a={(1,2,3):2, 'A2':3}

Крім того, словник можна створити використовуючи генератори:

>>> d={x: x\*\*2 for x in (2, 4, 6)}

>>> print(d)

{2: 4, 4: 16, 6: 36}

### 7.3.2. Виконання дій над елементами словника

В цьому пункті буде розглянуте виконання дій над словниками та іхніми елементами без застосування методів словників, які подані в наступному пункті.

## Доступ до значень словника

Враховуючи те, що елементом словника є пара ключ=значення, то, як такого, доступу до елемента словник не має. В словнику передбачена можливість доступу до значення елемента словника за його ключем.

Для того, щоб звернутися до значення елемента словника, необхідно вказати ім’я змінної словника та в квадратних дужках ключ необхідного елемента (ім’я\_словника[ключ]).

>>> e={'A1':2, 'A2':3}

>>> print(e['A2'])

3

При спробі доступу за неіснуючим у словнику ключем виникає виняток KeyError.

>>> print(e['A3'])

Traceback (most recent call last): File "<pyshell#99>", line 1, in <module> print(e['A3'])

KeyError: 'A3'

Можна перевірити приналежність деякого ключа до словника, використовуючи оператор in (ключ in ім’я\_словника).

>>> e={'A1':2, 'A2':3}

>>> 'A1' in e

True

>>> 'A3' in e

False

Також можна перевірити наявність певного значення в словнику, проте така перевірка буде відбуватися за результатом методу values().

>>> e={'A1':2, 'A2':3} >>> 2 in e.values()

True

За необхідності можна організувати перебір ключів усіх елементів словника.

d={'A1': 1, 'A2': 3, 'A3': 5, 'A4': 7} for k in d: print(k, d[k])

## Зміна значень словника

Враховуючи, що словник є змінюваним типом даних, за необхідності можна змінювати значення його елементів. Для зміни значення елемента словника необхідно вказати ім’я змінної словника та в квадратних дужках ключ та виконати присвоєння нового значення

(ім’я\_словника[ключ]=нове\_значення).

>>> e={'A1':2, 'A2':3}

>>> e['A1']=5

>>> print(e)

{'A1': 5, 'A2': 3}

## Додавання елементів до словника

Щоб додати елемент до словника, потрібно вказати ім’я змінної словника та в квадратних дужках новий ключ та виконати присвоєння нового значення (ім’я\_словника[новий\_ключ]=значення).

>>> e={'A1':2, 'A2':3}

>>> e['A3']=4

>>> print(e)

{'A1': 2, 'A2': 3, 'A3': 4}

## Вилучення елемента зі словника

Для видалення елемента зі словника можна скористатися командою del (del ім’я\_словника[ключ]).

>>> e={'A1':2, 'A2':3}

>>> del e['A1']

>>> print(e)

{'A2': 3}

### 7.3.3. Методи словників

**dict.fromkeys(iterable [, value])**. Створює новий словник, ключами якого будуть елементи з iterableі однаковим для всіх значенням value.

>>> d.fromkeys(['a', 'b', 'c'],12)

{ 'a': 12, 'b': 12, 'c': 12}} **dict.update([other])**. Доповнює словник dict парами

(ключ=значення) з словника other, якщо ключ вже присутній в словнику, то його значення оновлюється.

>>> d={'A1': 1, 'A2': 3, 'A3': 5, 'A4': 7}

>>> s={'A1':13, 'A5':15}

>>> d.update(s)

>>> print(d)

{'A1': 13, 'A2': 3, 'A3': 5, 'A4': 7, 'A5': 15} **dict.copy()**. Повертає копію словника dict.

>>> d={'A1': 1, 'A2': 3, 'A3': 5, 'A4': 7}

>>> e=d.copy()

>>> print(e)

{'A1': 1, 'A2': 3, 'A3': 5, 'A4': 7} **dict.get(key[, default])**. Повертає значення зі словника dict за ключем key. У випадку відсутності елемента з ключем key повертається значення default.

>>> d={'A1': 1, 'A2': 3, 'A3': 5, 'A4': 7}

>>> d.get('A2', 10)

3

>>> d.get('A8', 10)

10 **dict.setdefault(key[, default])**. Повертає значення зі словника dict за ключем key. У випадку відсутності елемента з ключем key повертається значення default, і до словника додається елемент з ключем key і значенням default.

>>> d={'A1': 1, 'A2': 3, 'A3': 5, 'A4': 7}

>>> d.setdefault('A1', 13)

1

>>> d.setdefault('A5', 13)

13

>>> print(d)

{'A1': 1, 'A2': 3, 'A3': 5, 'A4': 7, 'A5': 13} **dict.keys()**. Повертає ключі елементів словника dict у вигляді

об’єкту перегляду словника, що забезпечують динамічний перегляд записів словника.

>>> d={'A1': 1, 'A2': 3, 'A3': 5, 'A4': 7}

>>> d.keys()

dict\_keys(['A1', 'A2', 'A3', 'A4']) **dict.values()**. Повертає значення елементів словника dict у вигляді об’єкту перегляду словника, що забезпечують динамічний перегляд записів словника.

>>> d={'A1': 1, 'A2': 3, 'A3': 5, 'A4': 7}

>>> d.values()

dict\_values([1, 3, 5, 7]) **dict.items()**. Повертає ключі та значення елементів словника dict у вигляді об’єкту перегляду словника, що забезпечують динамічний перегляд записів словника. Елементи словника подаються в вигляді кортежів (ключ, значення).

>>> d={'A1': 1, 'A2': 3, 'A3': 5}

>>> d.items() dict\_items([('A1', 1), ('A2', 3), ('A3', 5)]) **dict.pop(key[, default])**. Вилучає зі словника dict елемент з ключем key та повертає його значення, як результат виконання функції. У випадку відсутності елемента з ключем key повертається значення default. Якщо default не вказаний, і елемент з ключем key відсутній, то виникає виняток KeyError.

>>> d={'A1': 1, 'A2': 3, 'A3': 5, 'A4': 7}

>>> d.pop('A2')

3

>>> print(d)

{ 'A1': 1, 'A3': 5, 'A4': 7} **dict.popitem()**. Вилучає і повертає пару (ключ, значення) зі словника dict, як результат виконання функції. Пари повертаються в порядку LIFO (last-in first-out). Якщо словник порожній, то виникає виняток

KeyError.

>>> d={'A1': 1, 'A2': 3, 'A3': 5, 'A4': 7}

>>> print(d)

{'A1': 1, 'A2': 3, 'A3': 5, 'A4': 7}

>>> d.popitem()

('A4', 7)

>>> print(d)

{'A1': 1, 'A2': 3, 'A3': 5} **dict.clear()**. Очищує словник dict (вилучає всі елементи зі

словнику).

>>> d={'A1': 1, 'A2': 3, 'A3': 5, 'A4': 7}

>>> print(d)

{'A1': 1, 'A2': 3, 'A3': 5, 'A4': 7}

>>> d.clear()

>>> print(d)

{}

### 7.3.4. Приклади розв’язування задач

**Приклад**. Написати програму для знаходження відстані між двома точками заданими своїми координатами. Для збереження даних точки скористатися словником.

p1={'x':0, 'y':0} p2={'x':0, 'y':0} print('введіть координати 1-ї точки: ') p1['x']=float(input('x:')) p1['y']=float(input('y:')) print('введіть координати 2-ї точки: ') p2['x']=float(input('x:')) p2['y']=float(input('y:')) import math import math lenP=math.sqrt((p1['x']-p2['x'])\*\*2+(p1['y']p2['y'])\*\*2)

print('відстань між точками ({}; {}) та ({}; {}) = {:.3}'.format(p1['x'], p1['y'], p2['x'], p2['y'],lenP))

## 7.4. Рядкові величини

З точки зору комп’ютера, текст – це набір символів, кожен з яких має свій код. Аналогічно і в мовах програмування, найменшою текстовою одиницею є символ. Символ – це або буква, або цифра, або інший спеціальний символ (символ пропуску, символ новго рядка і т.д.). А уже весь текст подається у вигляді рядка, який є послідовністю символів. Ми вже зустрічалися з рядками, а саме з рядковими літералами.

Як і більшість мов програмування, мова Python часто використовується для опрацювання текстів: пошуку в тексті, заміни окремих частин тексту і т.д. Для роботи з текстом у мові Python передбачений спеціальний рядковий тип даних str.

Рядковий тип або просто рядки в Python – це послідовність символів, яка використовується для зберігання і подання текстових даних. Користуючись рядковим типом, можна працювати з усім, що може бути подано в текстовому вигляді.

У мові Python символами рядка можуть бути будь-які символи, що підтримуються стандартом Unicode. У Python 3 немає ASCII-рядків, якщо необхідно отримати рядок строго в кодуванні ASCII, необхідно скористатися відповідним методом перекодування.

Варто відмітити, що в мові Python немає типу для символа рядка. Кожен символ рядка є також рядком. Також рядковий тип є незмінюваний.

### 7.4.1. Рядкові літерали та їх задання

В мові Python існує декілька способів задання рядкових літералів. Наприклад, можна розмістити текст в одинарних або подвійних лапках, і такий запис буде сприйматися, як текстовий літерал:

>>> 'Привіт' 'Привіт'

>>> "Привіт"

'Привіт'

Наявність двох таких варіантів забезпечує створення рядків, які будуть містити одинарні чи подвійні лапки всередині тексту і однозначно визначати кінці рядка.

>>> "Ім'я"

"Ім'я"

>>> 'Національний університет "Чернігівський колегіум".'

'Національний університет "Чернігівський колегіум".'

Іншим методом додавання в рядок одинарних чи подвійних лапок є так зване екранування символу. Для екранування символу перед ним записується символ оберненої косої риски (зворотній слеш, бекслеш, backslash) «**\**».

>>> 'Ім\'я'

"Ім'я"

>>> "Національний університет \"Чернігівський колегіум\"."

'Національний університет "Чернігівський колегіум".'

Рядкові літерали можна присвоювати змінним, які матимуть рядковий тип, і в подальшому використовувати їх:

>>> s='Привіт світ!'

>>> print(s) Привіт світ!

## Багаторядкові текстові блоки (Багаторядкові літерали)

При роботі з текстовими даними інколи виникає необхідність в заданні багаторядкового текстового блоку. Такий текстовий блок може використовуватися як багаторядковий коментар до програми чи анотація (синтаксис, на якому будуються підказки в мові Python) до функції.

Для того, щоб деякий текст, записаний в декілька рядків, вважався єдиним текстовим літералом, його необхідно взяти в три одинарні чи три подвійні лапки (''' або """). Окрім того в середині такого рядка можна використовувати одинарні чи подвійні лапки (головне, щоб не було трьох лапок підряд).

>>> '''Це дуже довгий текст, записаний в декілька рядків'''

'Це дуже довгий \nтекст, записаний \nв декілька рядків' >>> print(c) Це дуже довгий текст, записинай в декілька рядків

### 7.4.2. Задання рядків

Для задання порожнього рядку можна скористатися однією з наступних команд:

>>> s=''

>>> s=str()

## Використання рядкових літералів

Як зазначалося раніше, рядки можна створювати, використовуючи рядкові літерали:

>>> s1='Привіт'

>>> s2="Привіт" >>> c='''Це дуже довгий текст, записаний в декілька рядків'''

## Введення рядків

Аналогічно до введення числових даних, введення рядкових даних відбувається з використанням функції input().

>>> s=input('Введіть рядок: ')

Введіть рядок: Привіт

>>> print(s) Привіт

## Приведення до рядкового типу

Будь який стандартний об’єкт мови Python можна привести до рядкового типу, який йому відповідає, або отримати для нього «неформальне» рядкове подання. Для цього необхідно скористатися функцією str(object=''), передавши в якості параметра об’єкт, значення якого необхідно привести до рядкового типу.

>>> s=str(5)

>>> print(s)

'5'

>>> s=str(True)

>>> print(s)

'True'

## Об’єднання рядків

Нові рядки можуть бути створені методом конкатенації (об’єднання) декількох рядків. Для цього використовується перевизначена операція додавання («+»), яка використовується як операція конкатенації рядків.

>>> s1='Привіт '

>>> s2='світ!'

>>> s=s1+s2

>>> s

'Привіт світ!'

Незважаючи на незмінюваність рядкового типу, рядок може бути розширеним, тобто до рядка можна додати деякий інший, використовуючи операцію конкатенації.

>>> s='Привіт'

>>> id(s)

10388768

>>> s=s+' Світ!'

>>> print(s) Привіт Світ!

>>> id(s)

10388768

Враховуючи строгу типізацію мови Python і неможливість проводити операції у виразах з даними різних несумісних типів (рядок та число є несумісними типами), виконати об’єднання змінних рядкового та числового типів без додаткових перетворень не є можливим. Проте до проведення такого об’єднання можна привести числове значення до рядкового, а вже потім провести операцію конкатенації для двох рядкових значень.

>>> s1='Вавілон '

>>> n=15

>>> s2=str(n)

>>> print(s1+s2)

'Вавілон 15'

## Багаторазове повторення рядка або дублювання рядка

Аналогічно до перевизначеної операції («+»), в мові Python перевизначена і операція множення «\*». Якщо виконати операцію «\*» рядка s на ціле число n, то в результаті буде отриманий рядок, що складається з n повторень рядка s:

>>> s="СПАМ" \* 5

>>> print(s)

'СПАМСПАМСПАМСПАМСПАМ'

### 7.4.3. Доступ до символів рядку. Зрізи

Кожен символ рядка має свій індекс (порядковий номер). Доступ до символів рядка відбувається за їх індексами. Як і в багатьох інших мовах програмування, нумерація символів рядка починається з нуля. Для того, щоб звернутися до певного символу рядка, необхідно вказати ім’я рядкової змінної та в квадратних дужках індекс необхідного символу (рядок[індекс]).

>>> s='Привіт!'

>>> s[0]

'П'

Індекси можуть бути від’ємними, в такому випадку нумерація буде відбуватися з кінця (кількість символів рядка + від’ємний індекс).

>>> s[-1]

'!'

Якщо ж розглянути всі індекси для рядка 'Привіт!' , отримаємо:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Рядок s | П | р | и | в | і | т | ! |
| Індекс | s[0] | s[1] | s[2] | s[3] | s[4] | s[5] | s[6] |
| Індекс | s[-7] | s[-6] | s[-5] | s[-4] | s[-3] | s[-2] | s[-1] |

Якщо ж вказати індекс, який виходить за межі рядка (наприклад, s[8] або s[-9]), то генерується виняток: IndexError: string index out of range.

Можна перевірити приналежність деякого символу (підрядка) до рядка, використовуючи оператор in (підрядок in рядкок).

>>> a=[1, 3, 5, 7]

>>> 3 in a

True

>>> 3 in a False

## Зрізи (slice)

Досить часто, необхідно отримати не один якийсь символ (за індексом), а деякий набір символів за певними простими правилами. Наприклад: перші 5 символів, останні 3 символи, кожен другий символ. В таких завданнях замість перебору в циклі набагато зручніше використовувати так званий зріз (slice, slicing). Зріз – отримання з даного рядка набору з його символів. Зріз рядка також є рядком. Отримання одного символу рядка є найпростішим варіантом зрізу.

Слід пам'ятати, що беручи символ за індексом або зрізом, ми ніяк не змінюємо початковий рядок, ми просто скопіювали його частину для подальшого використання.

Задати зріз можна одним з двох варіантів: **str[start: stop]**.

**str[start: stop: step]**.

Для рядку str береться зріз від символу з індексом start, до символу з індексом stop (не включаючи його), з кроком step (тобто будуть взяті символи з індексами start, start + step, start + 2 \* step і т. д.). 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 символів рядку, step = 1.

>>> S='Привіт!'

>>> S[2:5]

'иві'

Якщо опустити другий параметр (залишивши двокрапку), то зріз береться до кінця рядка. Наприклад, щоб отримати зріз без перших двох символів, можна записати S[2:]. Якщо опустити перший параметр, то отримаємо зріз, який містить вказану кількість символів, що йдуть від початку рядка.

>>> S[2:]

'ивіт!'

>>> S[:3]

'При'

При заданні значення третього параметра, рівному 2, в зріз потрапить кожний другий символ рядку.

>>> S[::2]

'Пиі!'

>>> S[1::2]

'рвт'

Можна бачити, що взяття зрізу схоже на створення діапазону (range()).

Якщо значення параметру stop буде перевищувати кількість символів в рядку, воно буде проігнороване.

>>> S[2:50]

'ивіт!'

У випадку, якщо параметри start і stop мають від’ємні значення, то нумерація відбувається з кінця (кількість символів рядка + від’ємний індекс). Наприклад, S[1: -1] - це рядок без першого і останнього символу (зріз починається символом з індексом 1 і закінчується символом з індексом -1, не включаючи його).

>>> S[1:-1]

'ривіт'

>>> S[:-4] 'При'

>>> S[-4:]

'віт!'

Якщо параметр step має від’ємне значення, то зріз береться зправа наліво.

>>> S[::-1]

'!тівирП'

>>> S[-2::-1] 'тівирП'

>>> S[1:4:-1]

''

В останньому прикладі був отриманий порожній рядок, так як start < stop, а step від’ємний.

**7.4.4. Виконання дій над рядками та їхніми елементами**

## Визначення довжини рядка

Як зазначалося раніше, рядки складаються з окремих символів. Для того щоб дізнатися кількість символів у рядку (довжину рядка), необхідно скористатися функцією len(рядок).

>>> len('Привіт!')

7

## Зміна елементів рядка

Як було зазначено раніше, рядковий тип є незмінюваним, тобто зміна значення символу рядка чи його вилучення є неможливими.

>>> s='Привіт!'

>>> s[0]='п'

Traceback (most recent call last): File "<pyshell#48>", line 1, in <module> s[0]='п'

TypeError: 'str' object does not support item assignment

За необхідності можна записати оператори для зміни чи вилучення окремих символів рядка, проте в результаті ми отримаємо новий рядок.

>>> s='Hello'

>>> s=s[:1]+'a'+s[2:]

>>> print(s)

Hallo

>>> s=s[:1]+s[3:]

>>> print(s) Hlo

## Отримання коду символу та символу за кодом

Як було зазначено раніше, кожен символ текстового рядка має свій код. Інколи виникає необхідність отримання цього коду для певного символу, або зворотна операція – отримання символу за його кодом. При виконанні таких дій варто пам’ятати, що символи в мові Python є символами Unicode.

Для отримання коду символу можна скористатися функцією ord(c), за якою повертається ціле число, яке відповідає коду цього символу.

>>> ord('a')

97

>>> ord('€') 8364

>>> ord('½')

189

Зворотною до функції ord() є функція chr(n), за якою для цілого числа n (від 0 до 1114111=0x10FFFF) повертається символ (рядок, що є єдиним символом), для якого n є його кодом.

>>> chr(97)

'a'

>>> chr(8364)

'€'

>>> chr(189)

'½'

### 7.4.5. escape-послідовності

В попередньому пункті при виведенні рядкової змінної (без використання функції Print), яка містила багаторядковий текст, в місцях, де мали бути переходи на новий рядок, можна було бачити пару символів «\n». Ця пара символів починається з зворотного слеша і називається escape-послідовністю або керувальною послідовністю. Така послідовність в мові Python не єдина.

Використовуючи escape-послідовності, в рядки можна вставляти символи, які важко ввести з клавіатури. Серед найбільш вживаних escape-послідовностей можна виокремити:

\f – перехід на нову сторінку;

\n – перехід на новий рядок;

\r – переведення каретки;

\t – горизонтальна табуляція;

\v – вертикальна табуляція;

\0 – символ Null (не є ознакою кінця рядка);

>>> print('Це речення \nв два рядки') Це речення в два рядки

Отже, за функцією print() було розпізнано escape-послідовність і виконано перехід на новий рядок.

Проте інколи є необхідність, щоб символ зворотнього слеша не сприймався, як початок escape-послідовністі. Наприклад, це є важливим при заданні шляху до файлу в операційній системі Windows. Для такого відключення аналізу escape-послідовністей необхідно перед відкриваючою лапкою вставити символ r (у будь-якому регістрі),

>>> S = r'C:\newt.txt'

>>> print(S)

C:\newt.txt

Проте рядки з відключеним аналізом escape-послідовністей не можуть закінчуватися символом зворотнього слеша. Але це можна обійти:

>>> S = r'\n\n' + '\\'

>>> print(S)

\n\n\

>>> S = r'\n\n\\'[:-1]

>>> print(S)

\n\n\

### 7.4.6. Методи рядків

Для роботи з рядками в Python передбачена велика кількість вбудованих методів.

При роботі з методами рядків необхідно пам'ятати, що рядковий тип в Python є незмінюваним типом, тому всі рядкові методи повертають новий рядок, не змінюючи даного. Для того, щоб змінити сам рядок, необхідно викликати для нього потрібний метод і присвоїти його результат (рядкове значення) тій же змінній, в якій зберігається поточний рядок: str=str.метод().

**str.find(substr [, start [,end]])**. Повертає найменший індекс, за яким знаходиться початок підрядка substr в зрізі str[start:end] (необов’язкові параметри start та end інтерпретуються як нотації зрізу). Тобто знаходиться перше входження підрядка в рядку. Значення, що повертається, є індексом рядка str. Якщо підрядок не

|  |  |
| --- | --- |
| знайдено, то повертається значення -1. |  |
| >>> 'habrahabr'.find('r') | # 3 |
| >>> 'habrahabr'.find('r',4) | # 8 |
| >>> 'habrahabr'.find('abr') | # 1 |
| >>> 'habrahabr'.find('Abr') | # -1 |

**str.rfind(substr [, start [,end]])**. Повертає найбільший індекс, за яким знаходиться початок підрядка substr в зрізі str[start:end] (необов’язкові параметри start та end інтерпретуються як нотації зрізу). Тобто знаходиться останнє входження підрядка в рядку. Значення, що повертається, є індексом рядка str. Якщо підрядок не знайдено, то повертається значення -1.

>>> 'habrahabr'.rfind('abr') # 6

>>> 'habrahabr'.rfind('abr',7) # -1 **str.index(substr [, start [,end]])**. Повертає найменший індекс, за яким знаходиться початок підрядка substr в зрізі str[start:end] (необов’язкові параметри start та end інтерпретуються як нотації зрізу). Тобто знаходиться перше входження підрядка в рядку. Значення, що повертається, є індексом рядка str. Якщо підрядок не знайдено, то виникає виняток ValueError.

**str.rindex(substr [, start [,end]])**. Повертає найбільший індекс, за яким знаходиться початок підрядка substr в зрізі str[start:end] (необов’язкові параметри start та end інтерпретуються як нотації зрізу). Тобто знаходиться останнє входження підрядка в рядку. Значення, що повертається, є індексом рядка str. Якщо підрядок не знайдено, то виникає виняток ValueError.

**str.startswith(prefix[, start[, end]])**. Повертає True, якщо зріз str[start:end] (необов’язкові параметри start та end інтерпретуються як нотації зрізу) починається з префіксу prefix (prefix може бути кортежем, в такому випадку як префікс перевіряються всі елементи кортежу), інакше – False.

|  |  |
| --- | --- |
| >>> 'habrahabr'.startswith('ha') | # True |
| >>> 'habrahabr'.startswith('ha',1) | # False |
| >>> 'habrahabr'.startswith('ab',1) | # True |

>>> 'habrahabr'.startswith(('ha','bla')) # True

>>> 'blablacar'.startswith(('ha','bla')) # True **str.endswith(suffix[, start[, end]])**. Повертає True, якщо зріз str[start:end] (необов’язкові параметри start та end інтерпретуються як нотації зрізу) закінчується на суфікс suffix (suffix може бути кортежем, в такому випадку як суфікс перевіряються всі елементи кортежу), інакше – False.

|  |  |
| --- | --- |
| >>> 'habrahabr'.endswith('abr') | # True |
| >>> 'habrahabr'.endswith('abr',1,4) | # True |
| >>> 'habrahabr'.endswith('abr',1,8) | # False |

>>> 'habrahabr'.endswith(('abr','car')) # True

>>> 'blablacar'.endswith(('abr','car')) # True **str.count(substr [, start [,end]])**. Повертає кількість входжень підрядка sub в зріз str[start:end] (необов’язкові параметри start та end інтерпретуються як нотації зрізу) без самоперетинів.

|  |  |
| --- | --- |
| >>> 'habrahabr'.count('ha') | # 2 |
| >>> 'hahahahahah'.count('hah') | # 3 |
| >>> 'hahahahahah'.count('ha') | # 5 |

>>> 'hahahahahah'.count('ha',1,6) # 2 **str.isalpha()**. Повертає True, якщо рядок є непорожнім і складається лише з алфавітних символів, інакше – False. Алфавітні символи – це символи, які належать до категорії Юнікоду "Letter" ("Lm", "Lt", "Lu", "Ll", "Lo").

|  |  |
| --- | --- |
| >>> 'Habr'.isalpha() | # True |
| >>> ''.isalpha() | # False |
| >>> 'Habr1'.isalpha() | # False |
| >>> 'Habr%'.isalpha() | # False |

**str.isdecimal()**. Повертає True, якщо рядок є непорожнім і складається лише з десяткових цифр (десяткових символів), інакше – False.

Десяткові символи – це символи, які належать до категорії Юнікоду "Number, Decimal Digit" ("Nd").

|  |  |
| --- | --- |
| >>> '123'.isdecimal() | # True |
| >>> ''.isdecimal() | # False |
| >>> '1.1'.isdecimal() | # False |

**str.isdigit()**. Повертає True, якщо рядок є непорожнім і складається лише з цифр, інакше – False. Цифри – це символи, які належать до категорій Юнікоду "Number, Decimal Digit" та "Number, Other" ("Nd",

"No"). Наприклад, до них належать цифри надрядкового знаку.

>>> '2'.isdigit() # True

**str.isnumeric()**. Повертає True, якщо рядок є непорожнім і складається лише з числових символів, інакше – False. Числові символи – це символи, які належать до категорії Юнікоду "Number" ("Nd", "No", "Nl").

>>> 'VII'.isnumeric() # True

**str.isalnum()**. Повертає True, якщо рядок є непорожнім і

складається лише з літеро-числових символів, інакше – False. Літерочислові символи – це символи, які належать до категорій Юнікоду "Letter" та "Number" ("Lm", "Lt", "Lu", "Ll", "Lo", "Nd", "No", "Nl").

>>> 'Habr1'.isalnum() # True

**str.islower()**. Повертає True, якщо рядок містить принаймні одну літеру, і всі літери записані в нижньому регістрі, інакше – False.

|  |  |
| --- | --- |
| >>> 'habra habr'.islower() | # True |
| >>> 'habra habr 1'.islower() | # True |
| >>> 'habra Habr'.islower() | # False |
| >>> '1'.islower() | # False |

**str.isupper()**. Повертає True, якщо рядок містить принаймні одну літеру, і всі літери записані в верхньому регістрі, інакше – False.

|  |  |
| --- | --- |
| >>> 'HABRAHABR'.isupper() | # True |
| >>> 'HABRAHABR 1'.isupper() | # True |
| >>> 'HaBRAHABR 1'.isupper() | # False |
| >>> '1'.isupper() | # False |

**str.istitle()**. Повертає True, якщо рядок містить принаймні одну літеру, і літери, записані в верхньому регістрі, не йдуть безпосередньо після літер в нижньому чи верхньому регістрі, а перед групою літер в нижньому регістрі завжди стоїть літера в верхньому регістрі, інакше – False.

|  |  |
| --- | --- |
| >>> 'Habra Habr'.istitle() | # True |
| >>> '1Habra %Habr'.istitle() | # True |
| >>> 'Habra habr'.istitle() | # False |
| >>> 'hAbra Habr'.istitle() | # False |

**str.isspace()**.

Повертає True, якщо рядок є непорожнім і складається лише з символів пропуску (whitespace), інакше – False. Whitespace символи – це символи, які в базі даних Unicode визначені як «Other» або «Separator». Наприклад, до таких символів належать: пробіл, перехід на нову сторінку (\f), перехід на новий рядок (\n), переведення каретки (\r), горизонтальна та вертикальна табуляції (\t та \v).

|  |  |
| --- | --- |
| ' '.isspace() | # True |
| '\n\t'.isspace() | # True |
| ''.isspace() | # False |
| 'Habr!'.isspace() **str.isprintable()**. | # False |

Повертає True, якщо рядок містить лише символи, що друкуються (можуть бути виведені при друці), інакше – False. До символів, що друкуються, не входять символи пропуску (whitespace) окрім пробілу.

''.isprintable() # True

'Habra habr!'.isprintable() # True

'\n\t'.isprintable() # False

'Habra\nhabr'.isprintable() # False

**str.upper()**. Повертає копію рядка, в якому всі літери, записані в

нижньому регістрі, будуть приведені до верхнього регістру.

>>> 'Habra Habr 5%2=1'.upper()

'HABRA HABR 5%2=1'

**str.lower()**. Повертає копію рядка, в якому всі літери, записані в

верхньому регістрі, будуть приведені до нижнього регістру.

>>> 'Habra Habr 5%2=1'.lower()

'habra habr 5%2=1'

**str.swapcase()**. Повертає копію рядка, в якому всі літери, записані в верхньому регістрі, будуть приведені до нижнього регістру, а нижньому – до верхнього.

>>> 'Habra Habr 5%2=1'.swapcase()

'hABRA hABR 5%2=1'

**str.title()**. Повертає копію рядка, в якому перша літера кожного слова буде приведена до верхнього регістру, а всі інші – до нижнього. Першою літерою слова вважається літера, перед якою не міститься інші літери.

>>> 'hAbra Habr 5%2=1'.title()

'Habra Habr 5%2=1'

**str.capitalize()**. Повертає копію рядка, в якому перший символ, якщо він є літерою, буде приведений до верхнього регістру, а всі інші літери до нижнього.

>>> 'hAbra Habr 5%2=1'.capitalize() 'Habra habr 5%2=1'

>>> '5%2=1 hAbra Habr'.capitalize()

'5%2=1 habra habr' **str.replace(old, new[, count])**. Повертає копію рядка, в якому всі входження підрядка old будуть замінені на новий підрядок new.

Якщо задано парметри count, то буде виконано не більше чим count замін.

>>> 'habrahabr'.replace('a','A')

'hAbrAhAbr'

>>> 'habrahabr'.replace('a','A',2)

'hAbrAhabr' **str.lstrip([chars])**. Повертає копію рядка з вилученими початковими символами, вказаними в рядку chars. Якщо параметр chars відсутній або None, то вилучаються пропуски.

>>> ' habrahabr '.lstrip()

'habrahabr '

>>> 'www.habr.com'.lstrip('cmowa.')

'habr.com'

**str.rstrip([chars])**. Повертає копію рядка з вилученими кінцевими символами, вказаними в рядку chars. Якщо параметр chars відсутній або None, то вилучаються пропуски.

>>> ' habrahabr '.rstrip()

' habrahabr'

>>> 'www.habr.com'.rstrip('cmowa.')

'www.habr' **str.strip([chars])**. Повертає копію рядка з вилученими початковими та кінцевими символами, вказаними в рядку chars. Якщо параметр chars відсутній або None, то вилучаються пропуски.

>>> ' habrahabr '.strip()

'habrahabr'

>>> 'www.habr.com'.strip('cmowa.')

'habr' **str.expandtabs(tabsize=8)**. Повертає копію рядка, в якому всі символи табуляції (\t) замінюються декількома пропусками, в залежності від стовпця табулювання і заданого розміру табуляції tabsize.

>>> '\t1\t10\t100'.expandtabs()

' 1 10 100'

>>> '\t1234567\t10'.expandtabs()

' 1234567 10'

>>> '\t12345678\t10'.expandtabs()

' 12345678 10'

>>> '\t1\t10\t100'.expandtabs(4)

' 1 10 100' **str.split(sep=None, maxsplit=-1)**. Повертає список слів, які отримуються розбиттям рядка за роздільником рядком sep. Якщо параметр sep=None, то роздільником буде виступати пропуск, і результуючий список не буде містити порожніх елементів. Якщо задано параметр maxsplit, то буде виконано не більше чим maxsplit розбиттів (результуючий список буде мати не більше maxsplit+1 елемент).

>>> 'habrahabr'.split('a')

['h', 'br', 'h', 'br']

>>> 'haabrahabr'.split('a')

['h', '', 'br', 'h', 'br']

>>> 'habrahabr'.split('ab')

['h', 'rah', 'r']

>>> 'habrahabr'.split('a',2)

['h', 'br', 'habr'] **str.join(iterable)**. Повертає рядок, який є результатом конкатенації всіх рядків з iterable. Під час конкатенації між рядковими елементами iterable буде розміщений рядок str. Якщо iterable містить принаймні одне нерядкове значення, то генерується виняток TypeError.

>>> '..'.join(['1', '2'])

'1..2'

>>> '..'.join('hello')

'h..e..l..l..o' **str.partition(sep)**. Повертає кортеж з трьома рядковими значеннями, які є частинами рядку str. Першим елементом є частина рядка str, що міститься до першого входження роздільника sep. Другим елементом є сам роздільник sep. Третім елементом є частина рядка str, що міститься після роздільника sep. Якщо роздільника sep в рядку str не знайдено, то першим елементом кортежу буде сам рядок str, а другий та третій елементи будуть порожніми рядками.

>>> 'habrahabr'.partition('ra')

('hab', 'ra', 'habr')

>>> 'habrahabr'.partition('a')

('h', 'a', 'brahabr')

>>> 'habrahabr'.partition('car')

('habrahabr', '', '') **str.rpartition(sep)**. Повертає кортеж з трьома рядковими значеннями, які є частинами рядку str. Першим елементом є частина рядка str, що міститься до останнього входження роздільника sep. Другим елементом є сам роздільник sep. Третім елементом є частина рядка str, що міститься після роздільника sep. Якщо роздільника sep в рядку str не знайдено, то першим елементом кортежу буде сам рядок str, а другий та третій елементи будуть порожніми рядками.

>>> 'habrahabr'.rpartition('a')

('habrah', 'a', 'br') **str.ljust(width, fillchar=" ")**. Повертає копію рядка str, доповненого справа символами fillchar до довжини width. Якщо довжина рядка більша або рівна width, повертає оригінальний рядок.

>>> 'habr'.ljust(7)

'habr '

>>> 'habr'.ljust(7,'\_')

'habr\_\_\_' **str.rjust(width, fillchar=" ")**. Повертає копію рядка str ,доповненого зліва символами fillchar до довжини width. Якщо довжина рядка більша або рівна width, повертає оригінальний рядок.

>>> 'habr'.rjust(7)

' habr'

>>> 'habr'.rjust(7,'\_')

'\_\_\_habr' **str.center(width[, fillchar])**. Повертає копію рядка str, доповненого зліва та справа символами fillchar до довжини width, таким чином, щоб рядок був вирівняний за центром результуючого рядка.

Якщо довжина рядка більша або рівна width, повертає оригінальний рядок.

>>> 'habr'.center(8,'\_') '\_\_habr\_\_'

>>> 'habr'.center(7,'\_')

'\_\_habr\_' **str.zfill(width)**. Повертає копію рядка str, доповненого символами «0» до довжини width. Якщо рядок починається зі знаку («+» або «-»), то доповнюючи символи вставляються після знаку. Якщо довжина рядка більша або рівна width, повертає оригінальний рядок.

>>> '9'.zfill(4)

'0009'

>>> '-9'.zfill(4)

'-009'

>>> '+9'.zfill(4)

'+009' **str.format(\*args, \*\*kwargs)**. Повертає копію рядка str, відформатованого відповідним чином. Детальні відомості щодо цього методу викладено в додатку 3

### 7.4.7. Приклади розв’язування задач

**Приклад**. Написати програму за якою буде визначатися, чи є задане користувачем слово паліндромом (паліндром – слово, що читається однаково зліва направо і навпаки, наприклад, "шалаш"). s=input('Введіть рядок: ') if s==s[::-1]:

print('Паліндром') else:

print('Не паліндром')

**Приклад**. Дано рядок, який складається із слів розділених пропусками. Написати програму за якою буде обраховуватися кількість слів в рядку.

Якщо припустити, що пропуски містьться лише між словами іїх там строго по одному, то розвязок задачі може мати вигляд: s=input('Введіть рядок: ') n=s.count(' ')+1 print(n)

Якщо ж пропусків між словами може бути скільки завгодно, то розвязок задачі може мати вигляд:

s=input('Введіть рядок: ') a=list(s.split(' ')) b=[] for i in a: if i!='' : b=b+[i] print(len(b))

Останій цикл можна переписати в вигляді генератору спсику:

b =[i for i in a if i!='']

**Приклад**. Дано рядок. Написати програму за якою буде відбуватися поділ рядка на дві рівні частини (якщо довжина рядка непарна то перша частина має бути на 1 символ меншою) та виведення їх окремими рядками.

При розв’язання задачі не використовувати інфструкцію if. s=input('Введіть рядок: ') print(s[:len(s)//2]) print(s[len(s)//2:])

**Приклад**. Дано рядок, який складається із слів розділених пропусками та розділовими знаками (,.!?). Написати програму за якою буде виведено всі слова окремими рядками.

s=input('Введіть рядок: ') a=list(s.split(' ')) for i in a: if i!='':

print(i.strip(',.!?'))

Останій цикл можна переписати в вигляді генератору спсику:

[print(i.strip(',.!?')) for i in a if i!='']

**Приклад**. Дано рядок, який складається із слів розділених пропусками та розділовими знаками (,.!?). Написати програму за якою буде виведено всі слова парної довжини окремими рядками. s=input('Введіть рядок: ') a=list(s.split(' ')) a=[i.strip(',.!?') for i in a if i!='']

[print(i) for i in a if len(i)%2==0]

**Приклад**. Дано рядок, який складається із слів розділених пропусками та розділовими знаками (,.!?). Написати програму за якою буде виведено всі слова що не містять повторюваних символів. s=input('Введіть рядок: ') a=list(s.split(' ')) a=[i.strip(',.!?') for i in a if i!=''] for i in a: flag=True for j in i:

if i.count(j)!=1: flag=False if flag: print(i)

## 7.5. Множини

Множина - це структура даних, що містить невпорядкований набір унікальних елементів. Множини в мові Python досить подібні до множин у математиці. Використання множин є доцільним у тому випадку, коли присутність елемента в наборі важливіша порядку слідування елементів та того, скільки разів даний елемент там зустрічається.

Множина може містити елемененти різних типів, проте ці елементи можуть бути лише незмінюваних типів даних: числа, рядки, кортежі. Вимога незмінності елементів множини накладається особливостями подання множини в пам'яті комп'ютера.

Сама множина є змінюваним типом даних, тому до множин можна додавати нові та видаляти існуючі елементи. Як і у випадку математичних множин, у мові Python передбачено виконання операцій над множинами: об'єднання, перетину, різниці, симетричної різниці.

На відміну від масивів, де елементи зберігаються у вигляді послідовного списку, у множинах порядок зберігання елементів невизначений. Це дозволяє виконувати операції типу "перевірити приналежність елемента множині" швидше, ніж просто перебираючи всі елементи множини.

Множини записуються, як перелік елементів, розділених комою та взятих в фігурні дужки: {1, 2, 3, 'Hello'}.

### 7.5.1. Задання множини

Задання порожньої множини виконується з використанням функції set():

>>> a = set()

Використання «{}» призведе до створення порожнього словника:

>>> s={}

>>> type(s)

<class 'dict'>

Проте можна задати множину, перерахувавши її елементи, взяті в фігурні дужки «{}»:

>>> s1 = {1, 2, 3}

>>> s2 = {'a', 'b', 'c', 5}

## Створення множини з інших структур даних

Множину можна отримати з елементів об’єкта, що може ітеруватися (діапазон, список, рядок, словник, кортеж, файл і т.д.), використавши функцію **set([iterable])**. Проте варто пам’ятати, що до множини будуть включені лише унікальні елементи.

>>> a = set('hello') >>> a

{'e', 'o', 'l', 'h'}

>>> set(range(5)

{0, 1, 2, 3, 4}

### 7.5.2. Виконання дій над елементами множини

Як зазначалося раніше, можна перевірити приналежність деякого елемента до множини, використовуючи оператор in (значення in ім’я\_множини).

>>> a = {1, 2, 3}

>>> 2 in a

True

>>> 2 not in a

False

Відповідно для перебору всіх елементів множини (в невизначеному порядку) необхідно скористатися циклом for: s = {2, 5, 7, 11, 4} for num in s:

print(num)

### 7.5.3. Порівняння множин

Множини можна порівнювати між собою. Порівняння множин зводиться до перевірки, чи є множини рівними, або чи є певна множина підмножиною іншої.

>>> a={1,2,3}

>>> b={1,2,3}

>>> d={1,2,3,4}

>>> c={4,5,6} **set == other**. Перевірка, чи множини set та other рівні. Повертає True, якщо всі елементи множини set належать множині other, і всі елементи множини other належать множині set, інакше – False.

>>> a == b True

>>> a == d

False

**set != other**. Перевірка, чи є множини set та other не рівними. Повертає True, якщо приаймні один елемент множини set не належать множині other, або приаймні один елемент множини other не належать множині set, інакше – False.

>>> a != b

False

>>> a != d

True **set <= other**. Перевірка, чи є множина set підмножиною множини other. Повертає True, якщо всі елементи множини set належать множині other, інакше – False.

>>> a<=d True

>>> a<=b

True **set < other**. Повертає True, якщо всі елементи множини set належать множині other, але не всі елементи множини other належать множині set, інакше – False.

>>> a<d True

>>> a<b

False **set.isdisjoint(other)**. Повертає True, якщо множини set і

other не мають спільних елементів, інакше – False.

>>> a.isdisjoint(c) True

>>> c.isdisjoint(d)

False **set.issubset(other)**.Перевірка, чи є множина set підмножиною множини other. Аналогічно до set <= other. **set.issuperset(other)**. Перевірка чи є множина other

підмножиною множини set. Аналогічно до set >= other.

### 7.5.4. Методи множин

**set.add(x)**. Додає елемент x до множини set.

>>> s = {1, 2, 3}

>>> s.add(4)

>>> print(s)

{1, 2, 3, 4} **set.remove(x)**. Вилучає елемент x із множини set. Якщо такого

елемента в множині немає, то виникає виняток KeyError.

**set.discard(x)**. Вилучає елемент x із множини set. Якщо такого

елемента в множині немає, то нічого не відбувається **set.pop()**. Вилучає «перший» елемент з множини set та повертає його значення, як результат виконання функції. Так як множина – це невпорядкований набір, то не можна точно передбачити який з елементів буде взятий як перший. Якщо множина порожня, то виникає виняток KeyError.

**set.clear().** Очищує множину set (вилучає всі елементи з

множини).

## Операції з множинами

>>> d={1,2,3,4}

>>> c={4,5,6} **set.union(\*other)** або **set | other | ...**. Повертає об’єднання множин set і other. Множина-результат буде містити як елементи множини set, так і елементи множини other.

>>> d | c

{1, 2, 3, 4, 5, 6} **set.intersection(\*other)** або **set & other & ...**.

Повертає перетин множин set і other. Множина-результат буде містити елементи, які належать як множині set, так і множині other.

>>> d & c

{4} **set.difference(\*other)** або **set - other - ...**. Повертає різницю множин set і other. Множина-результат буде містити елементи множини set, які не належать множині other.

>>> d - c

{1, 2, 3}

>>> c - d

{5, 6} **set.symmetric\_difference(\*other)** або **set ^ other**. Повертає симетричну різницю множин set і other. Множина-результат буде містити елементи, які належать множинам set та other, але не належать обом множинам. Аналогічно до (set | other)-( set & other).

>>> d^c

{1, 2, 3, 5, 6}

Як можна бачити, вказані методи не призводять до зміни множини, а за ними відбувається формування нової множини. Проте для множин передбачені методи, шо будуть призводити до зміни початкової множини.

**set.update(\*other)** або **set |= other.** Додає до множини set всі елементи множини other. Множина set буде містити об’єднання множин set і other.

**set.intersection\_update(\*other)** або **set &= other.**

Вилучає з множини set всі елементи, які не входять до множинаи other.

Множина set буде містити перетин множин set і other. **set.difference\_update(\*other)** або **set -= other.**

Вилучає з множини set всі елементи, які входять до множини other.

Множина set буде містити різницю множин set і other.

**set.symmetric\_difference\_update(other)** або

**set ^= other.** Множина set буде містити симетричну різницю множин set і other.

### 7.5.5. Приклади розв’язування задач

**Приклад**. Дано два рядки. Написати програму за якою буде створено дві множини (перша з символів першого рядка, друга з символів другого рядка) та виведено перетин, об’єднання, різницю цих множин. st1=input('Введіть 1-й рядок') st2=input('Введіть 2-й рядок') s1=set(st1) s2=set(st2) o=s1|s2 p=s1&s2 r1=s1-s2 r2=s2-s1 print('Множина 1:',s1) print('Множина 2:',s2) print("Обє'днання 1:",o) print('Перетин 1:',p) print('Різниця між 1-ю і 2-ю множинами:',r1) print('Різниця між 2-ю і 1-ю множинами:',r2)

**Приклад**. Дано рядок, який складається із слів розділених пропусками та розділовими знаками (,.!?). Написати програму за якою буде виведено всі слова що не містять повторюваних символів.

s=input('Введіть рядок: ') a=list(s.split(' ')) a=[i.strip(',.!?') for i in a if i!=''] for i in a: if len(i)==len(set(i)):

print(i)

Останій цикл можна переписати в вигляді генератору спсику:

[print(i) for i in a if len(i)==len(set(i))]